SEVENTH FRAMEWORK PROGRAMME

Research Infrastructures

INFRA-2007-2.2.2.1 - Preparatory phase for 'Computer and Data Treatment' research infrastructures in the 2006 ESFRI Roadmap

PRACE

Partnership for Advanced Computing in Europe

Grant Agreement Number: RI-211528

D8.3.2

Final technical report and architecture proposal

Final

Version: 1.0

Author(s): Ramnath Sai Sagar (BSC), Jesus Labarta (BSC), Aad van der Steen (NCF), Iris Christadler (LRZ), Herbert Huber (LRZ)

Date: 25.06.2010

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="6">PRACE Project</td><td colspan="2">Project Ref. №: RI-211528</td></tr>
<tr><td colspan="2">Project Title: Final technical report and architecture proposal</td></tr>
<tr><td colspan="2">Project Web Site: http://www.prace-project.eu</td></tr>
<tr><td colspan="2">Deliverable ID: : <D8.3.2></td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td>Deliverable Level:
PU *</td><td>Contractual Date of Delivery:
30 / 06 / 2010

Actual Date of Delivery:
30 / 06 / 2010</td></tr>
<tr><td colspan="2">EC Project Officer: Bernhard Fabianek</td></tr>
</table>

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

Document	Title: : Final technical report and architecture proposal	
	ID: D8.3.2	
	Version: 1.0	Status: Final
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2003	
	File(s): D8.3.2_addn_v0.3.doc	
Authorship	Written by:	Ramnath Sai Sagar (BSC), Jesus Labarta (BSC), Aad van der Steen (NCF), Iris Christadler (LRZ), Herbert Huber (LRZ)
	Contributors:	Eric Boyer (CINES), James Perry (EPCC), Paul Graham (EPCC), Mark Parsons (EPCC), Alan D Simpson (EPCC), Willi Homberg (FZJ), Wolfgang Gürich(FZJ), Thomas Lippert (FZJ), Radoslaw Januszewski (PSNC), Jonathan Follows (STFC), Igor Kozin (STFC), Dave Cable (STFC), Hans Hacker (LRZ), Volker Weinberg (LRZ), Johann Dobler (LRZ), Christoph Biardzki (LRZ), Reinhold Bader (LRZ), Momme Allalen (LRZ), Jose Gracia (HLRS), Vladimir Marjanovic (BSC), Guillaume Colin De Verdière (CEA), Calvin Christophe (CEA), Hervé Lozach (CEA), Jean-Marie Normand (CEA), Sadaf Alam (CSCS), Adrian Tineo (CSCS), Tim Stitt (CSCS), Neil Stringfellow(CSCS), Giovanni Erbacci (CINECA), Giovanni Foiani (CINECA), Carlo Cavazzoni (CINECA), Filippo Spiga (CINECA), Kimmo Koski (CSC), Jussi Heikonen (CSC), Olli-Pekka Lehto (CSC), Lennart Johnsson (KTH), Lilit Axner (KTH)
	Reviewed by:	Thomas Eickermann (FZJ), Miroslaw Kupzyk (PSNC)
	Approved by:	Technical Board

Document Keywords and Abstract

Keywords:	PRACE, HPC, Research Infrastructure
Abstract:	This document describes the activities in Work Package 8 Task 8.3 (WP8.3) updating and analysing results reported in D8.3.1 for the different WP8 prototypes. The document also suggests potential architectures for future machines, the level of performance we should expect and areas where research efforts should be dedicated.

Table of Contents

Project and Deliverable Information Sheet i
Document Control Sheet ii
Document Keywords and Abstract iii
Table of Contents iv
List of Figures vi
List of Tables ix
References and Applicable Documents x
Executive Summary 1
1 Introduction 2
1.1 Scope and Structure of the Report 3
2 WP8 prototypes and Research Activities 4
2.1 Prototypes 4
2.1.1 eQPACE 4
2.1.2 BAdW-LRZ/GENCI-CINES Phase1 (CINES Part) 5
2.1.3 BAdW-LRZ/GENCI-CINES Phase2 (LRZ Part) 6
2.1.4 Intel Many Integrated Core (MIC) architecture 7
2.1.5 ClearSpeed-Petapath 8
2.1.6 Hybrid Technology Demonstrator 9
2.1.7 Maxwell FPGA 10
2.1.8 XC4-IO 11
2.1.9 SNIC-KTH 12
2.1.10 RapidMind 13
2.2 Research activities 14
2.2.1 PGAS language compiler 14
2.2.2 Research on Power Efficiency 16
2.2.3 Parallel GPU 17
2.2.4 Performance Predictions 18
3 WP8 evaluation results 21
3.1 Performance experiments on prototype hardware 21
3.1.1 Reference performance 21
3.1.2 Numerical issues 25
3.1.3 Accelerated Programming Languages and Compilers 28
3.1.4 FPGA experiments 37
3.1.5 LRZ + CINES (Phase 1) 41
3.1.6 LRZ + CINES (Phase 2) 44
3.1.7 Intel MIC Architecture 48
3.1.8 Petapath experiments 50
3.2 Hybrid Programming Models 53
3.2.1 MPI+OpenMP 53
3.2.2 MPI+CUDA 55
3.2.3 MPI + CellSs 58
3.3 Intra-Node Bandwidth 58
3.3.1 Triads (RINF1) benchmark results (BAdW-LRZ) 58
3.3.2 Random Access Results 59
3.3.3 Host to accelerator bandwidth (NCF) 60
3.4 Inter Node Communication Network 61
3.4.1 eQPACE 61
3.4.2 BAdW-LRZ 64
3.5 PGAS Languages 69
3.5.1 Chapel experiments 69
3.5.2 Co-Array Fortran experiments 70
3.5.3 UPC Experiments 74
3.6 Novel I/O: XC4-IO experiments 77
3.6.1 Lustre Architecture 77
3.6.2 Throughput tests 78
3.6.3 File striping tests 79
3.6.4 Tests on the Metadata device technology 80
3.6.5 Metadata server load tests 81

3.6.6 Parallel I/O libraries test ... 81
3.6.7 Preliminary tests on pNFS ... 83
3.6.8 Conclusion ... 83
3.7 Energy efficiency ... 85
3.7.1 The SNIC-KTH system ... 85
3.7.2 PSNC results ... 91
3.7.3 STFC results ... 100
3.7.4 BAdW-LRZ results ... 102
3.8 Performance predictions ... 102
3.8.1 Impact of basic system components ... 102
3.8.2 Non ideal node level parallelisation ... 104
3.8.3 Prediction for ICE ... 106
3.8.4 Prediction for BG/P ... 107
3.8.5 General analysis ... 110
3.9 Summary of conclusions ... 112
3.9.1 Node/core performance: Accelerators vs. general purpose CPUs ... 112
3.9.2 Memory bandwidth ... 114
3.9.3 Network bandwidth ... 114
3.9.4 Hybrid ... 115
3.9.5 I/O ... 116
3.9.6 Energy Efficiency ... 117
4 Recommendations for next generation Petascale machines ... 118
4.1 Foreseeable architectures ... 118
4.1.1 General architecture ... 118
4.1.2 Some straw man examples ... 120
4.2 Relevant issues ... 121
4.2.1 Power and Energy efficiency ... 121
4.2.2 Programming models and compilers ... 122
4.2.3 Accelerators ... 122
4.2.4 Network Interconnects ... 123
4.2.5 Memory bandwidth and latency ... 124
4.2.6 Memory per node and core ... 124
4.2.7 Performance tools ... 125
4.2.8 Load balance ... 125
4.2.9 Runtime Systems ... 126
4.2.10 Resilience ... 126
4.2.11 Arithmetic ... 127
4.2.12 Benchmarks ... 127
4.2.13 Libraries ... 127
4.2.14 Applications ... 127
5 Conclusions and Final remarks ... 129
6 Annex ... 130
6.1 Benchmarks ... 130
6.1.1 EuroBen – Synthetic Benchmarking Suite ... 130
6.1.2 High Performance LINPACK ... 130
6.1.3 Intel MPI Benchmark (IMB) ... 131
6.1.4 Triads (RINF1) Benchmark ... 131
6.1.5 Random Access Benchmark ... 132
6.1.6 APEX Benchmark ... 132
6.1.7 STREAM Benchmark ... 132
6.1.8 IOR Benchmark ... 133
6.1.9 CPU Burn-in ... 133
6.1.10 CacheBench ... 133
6.1.11 IOzone ... 134
6.2 Applications ... 134
6.2.1 GADGET ... 134
6.2.2 NAMD ... 135
6.2.3 RAxML ... 135
6.2.4 DL-POLY ... 135

List of Figures

Figure 1: QPACE Architecture 4
Figure 2: Prototype configuration and integration layout 6
Figure 3: ClearSpeed CSX 710 card 6
Figure 4: Hybrid system prototype installed at BAdW-LRZ 7
Figure 5: Scheme of Intel MIC architecture prototype 8
Figure 6: x86-based core and associated system blocks 8
Figure 7: Block diagram of vector processing unit (VPU) 8
Figure 8: Clearspeed-Petapath prototype 8
Figure 9: Overview of HMPP components 9
Figure 10: I/O & File System Prototype architectural scheme 11
Figure 11: Prototype's motherboard 12
Figure 12: 10-blade chassis 12
Figure 13: Chassis features 12
Figure 14: Rack of prototype 13
Figure 15: Data-stream processing in RapidMind 13
Figure 16: Connection diagram 17
Figure 17: Gant diagram and efficiency model 20
Figure 18: Model including microscopic load balance and serialization 20
Figure 19: Processor family share in the November 2009 Top500 list 21
Figure 20: Reference performance of dense matrix-matrix multiplication 22
Figure 21: Reference performance of sparse matrix-vector multiplication 23
Figure 22: Reference performance of 1-D complex-to-complex FFT 23
Figure 23: Reference performance of the random number generator 24
Figure 24: Reference performance of dense matrix-matrix and sparse matrix-vector multiplication 25
Figure 25: Reference performance of FFT and random number generation 25
Figure 26: Precision of the CGS Arnoldi Projection of a 10240 square Hilbert matrix 27
Figure 27: Precision of the sparse CGSr process on sparse Andrews matrix 28
Figure 28: Comparison of the CUDA kernels on nVIDIA c1060 GPUs 28
Figure 29: Comparison of the MKL kernels on Nehalem-EP 28
Figure 30: mod2am on a single core Nehalem vs. nVIDIA C1060 30
Figure 31: mod2as on a single core Nehalem vs. nVIDIA C1060 30
Figure 32 : Example illustrating the richness of expression offered by HMPP's set of directives 31
Figure 33: Nehalem-EP single socket mod2am performance versus nVIDIA C1060 using HMPP 31
Figure 34: Nehalem-EP single socket mod2as performance versus nVIDIA C1060 using HMPP 31
Figure 35: Performance of mod2am using CellSs on Maricel 32
Figure 36: Performance of mod2as using CellSs on Maricel 33
Figure 37: Performance of mod2am using GPUSs on nVIDIA Tesla 33
Figure 38: Performance of mod2am on Nehalem EP using SMPSs 34
Figure 39: Performance of mod2as on Nehalem using SMPSs 34
Figure 40: Performance of mod2am using ClearSpeedSs on the ClearSpeed-Petapath prototype 35
Figure 41: RapidMind results for mod2am (GPU) 36
Figure 42: RapidMind results for mod2am (CELL) 36
Figure 43: RapidMind results for mod2am (x86) 36
Figure 44: RapidMind results for mod2as 36
Figure 45: RapidMind results for mod2f 36
Figure 46: Performance of mod2f for different problem sizes 39
Figure 47: CPU/CSX710 (K = 1152) 42
Figure 48: CPU / CX710 across matrix sizes 42
Figure 49: HPL / 32 node + CSXL with varying hostassist 43
Figure 50: HPL / 32 node + CSXL with varying memory size 43
Figure 51: Runtimes and Speedups for synthetic DNA dataset with 50 sequences 44
Figure 52: Runtimes and Speedups for synthetic DNA dataset with 250 sequences 45
Figure 53: Network topology of 9728-core SGI Altix4700 system HLRB II at BAdW-LRZ 46

Figure 54: Scalability of TifaMMy on Many Intel Core Architecture and a recent x86 system. 50
Figure 55: Absolute performance of TifaMMy on Many Intel Core Architecture and a recent x86 system. 50
Figure 56: Performance of mod2am on 1 MTAP 51
Figure 57: Performance of mod2as on 1MTAP 52
Figure 58: Performance for mod2f on 1MTAP 52
Figure 59: Performance for mod2h on 1 MTAP 53
Figure 60: Hybrid MPI+OpenMP results for mod2am 54
Figure 61: Hybrid MPI+OpenMP results for mod2as 54
Figure 62: Speedup of the Linear Algebra subtask of a Car-Parrinello simulation 54
Figure 63: BQCD scaling results on Cray XT5 system 55
Figure 64: BQCD scaling results on IBM BlueGene/P 55
Figure 65: Results of CUDA+MPI mod2am with a reference mxm routine 55
Figure 66: Results of CUDA+MPI mod2am with a CUBLAS mxm routine 56
Figure 67: Results of CUDA+MPI mod2as 56
Figure 68: Runtime of CPU vs. GPU HMMER 57
Figure 69: Performance of mod2am with MPI+CellSs on Maricel 58
Figure 70: Single task performance of the vector triad measured on different processor architectures 59
Figure 71: Measured host-card bandwidth 1 MTAP 60
Figure 72: QPACE network processor 62
Figure 73: Open MPI Modular Component Architecture 63
Figure 74: MPI bisection bandwidths (left figure) and MPI send receive latencies (right figure) 65
Figure 75: MPI Allreduce timings for SGI Altix4700, ICE and UltraViolet 65
Figure 76: Measured Altix ICE MPI bisection bandwidths 66
Figure 77: Measured Altix ICE MPI bisection latencies 66
Figure 78: Measured MPI Allreduce timings 67
Figure 79: Measured MPI Alltoall timings 68
Figure 80: Measured MPI send/receive link bandwidths with and without traffic aware routing 69
Figure 81: Cray XT5 (left) and Cray X2 (right) processing nodes 71
Figure 82: Lustre throughput performances 78
Figure 83: Bandwidth (in MB/s) for Read and Write operations with different striping count 79
Figure 84: Performance (I/O ops/s) with different stripe count with a large number of files 80
Figure 85: Comparison between SSDs and magnetic Hard Disks (I/O ops/s) 80
Figure 86: Comparison between SSDs and magnetic Hard Disks (% cpu utilization of the MDS) 81
Figure 87: IOR results (MB/s) using individual communication 81
Figure 88: IOR results (MB/s) using collective communication 82
Figure 89: RAMSES Test (MB/s) reads 82
Figure 90: RAMSES Test (MB/s) writes 83
Figure 91: Performance ratio of Elpida/Samsung, Supermicro 4-socket blade 86
Figure 92: Performance/W ratio for Elpida/Samsung Supermicro 4-socket blade 86
Figure 93: Stream Copy 87
Figure 94: Near linear relation between tasks and updates for StarRandomAccess 88
Figure 95: Relation between tasks and updates for MPIRandomAccess 88
Figure 96: Comparison of the performance of the dense matrix-matrix multiplication kernel for two different implementations: Fortran, C with ACML and C with MKL 89
Figure 97: Performance of the dense matrix-matrix multiplication with GotoBLAS2 libraries 89
Figure 98: Performance of the sparse matrix-vector multiplication 90
Figure 99: of the random number generator kernel 90
Figure 100: Performance of the fast Fourier transform kernel 90
Figure 101: Summary of Power consumption of tested servers 92
Figure 102: NAMD execution times 92
Figure 103: Power consumption - NAMD appoa1 benchmark 93
Figure 104: Gromacs run time on SiCortex 94
Figure 105: Power consumption of SiCortex and Xeon E5345 95
Figure 106: Performance reported by the dgemm application 96
Figure 107: Performance for the 1k x 1k matrix multiplication (dgemm) 97

Figure 108: Performance for the 6k x 6k matrix multiplication 97
Figure 109: The CPU load while running the four instances of the dgemm test 98
Figure 110: Power consumption reported by the cards while running a single dgemm application 98
Figure 111: Computational performance 98
Figure 112: Server power consumption 99
Figure 113: Performance/power ratio of the test server 99
Figure 114: Impact of node performance and Interconnect bandwidth (in MB/s) for 64 processes ... 103
Figure 115: Impact of node performance and Interconnect bandwidth (in MB/s) for 128 processes . 103
Figure 116: Impact of node performance and Interconnect bandwidth (in MB/s) for 256 processes . 104
Figure 117: Profile of computation regions in GADGET for 128 processors in MareNostrum 105
Figure 118: Impact of bandwidth and acceleration factor applied to major computation bursts representing 93.67 % of the original computation time 105
Figure 119: Impact of bandwidth and acceleration factor applied to major computation bursts representing 97.49 % of the original computation time 105
Figure 120: Impact of bandwidth and acceleration factor applied to major computation bursts representing 99.11 % of the original computation time 105
Figure 121: MPI calls for real run (top) and prediction (bottom) for 128 processes on ICE 106
Figure 122: MPI calls for real run (top) and prediction (bottom) for 128 processes on ICE 107
Figure 123: MPI calls for real run (top) and prediction (bottom) for 128 processes on Jugene..... 108
Figure 124: Histogram of duration of major computation bursts in the real run (on top) and prediction (bottom) 109
Figure 125: Zoom of duration of computation phases (top) and MPI calls (bottom) in exchange region 110
Figure 126: Instantaneous parallelism profile for 64, 128 and 266 processor runs on ICE 111
Figure 127: Instantaneous efficiency in the communication phase of the 256 processor trace from ICE, assuming ideal interconnect 111
Figure 128: Serialized computations pattern 112
Figure 129: Performance relative to reference platform of mod2am 112
Figure 130: Performance relative to reference platform of mod2as 112
Figure 131: Performance relative to reference platform of mod2f 113
Figure 132: current structure and performance of top machines in Top500 list 118
Figure 133: Possible evolution of supercomputer dimensioning 119
Figure 134: Kernel-based performance modeling 125

List of Tables

Table 1: Hardware used in the RapidMind experiments. 35
Table 2: mod2am performance results 37
Table 3: mod2am performance results 38
Table 4: mod2f performance results 39
Table 5: mod2h performance results 40
Table 6: Measured execution time of 256-core GADGET runs 46
Table 7: Memory bandwidth and spatial degradation factors of different processor platforms 59
Table 8: Measured random access memory latencies of different processor architectures 60
Table 9: MPI send/receive benchmark results 64
Table 10: Measured barrier latencies 67
Table 11: Influence of different MPI versions and network pruning on execution time of GADGET 68
Table 12: Test system configurations and roadmap 72
Table 13: STREAM benchmark results for the CAF compiler 73
Table 14: CAF STREAM results (GB/s) on X2 74
Table 15: Impact of alternate image mapping configurations on a CAF code execution 74
Table 16: Two equivalent alternatives to distribute the iterations for array arr between threads according to data locality 75
Table 17: Summary of experiences with CAF and UPC compilers available as part of the CCE framework on the Cray XT5 platform (options added since the previous release are highlighted) 77
Table 18: Power consumption of powered off servers 92
Table 19: Xeon 5130 server - internal power consumption 93
Table 20: NAMD, STMV benchmark, 500 steps 101
Table 21: LINPACK power efficiency on different architectures 101
Table 22: Prediction and error with respect to actual iteration time on the ICE prototype 106
Table 23: Prediction and error with respect to actual iteration time on the Jugene prototype 108
Table 24: Global performance model for GADGET 110
Table 25: Point-to-point performance 114
Table 26: MPI performance of collective calls 115
Table 27: STREAM Benchmark Kernels 132

References and Applicable Documents

[1] http://www.prace-project.eu
[2] D8.3.1 Technical Component Assessment and Development Report
[3] D8.1.4 Signing of AHTP contract
[4] D8.2.2 Architectural specifications from user requirements
[5] D2.7.2 Final report on selection of prototypes
[6] D2.7.2-Addendum Report on selection of 2nd prototypes
[7] D5.4 Report on the Application Benchmarking Results of Prototype Systems
[8] D6.6 Report on Petascale Software Libraries and Programming Models
[9] CAPS Entreprise http://www.caps-entreprise.com/index.php
[10] HARWEST Compiling Environment (HCE): http://www.ylichron.it/index.php
[11] Paraver: http://www.bsc.es/plantillaA.php?cat_id=485
[12] Dimemas: http://www.bsc.es/plantillaA.php?cat_id=475
[13] EuroBen : http://www.euroben.nl/
[14] CUDA Zone: http://www.nvidia.com/object/cuda_home.html
[15] Efficient Sparse Matrix-Vector Multiplication on CUDA" by Nathan Bell and Michael Garland http://www.nvidia.com/object/nvidia_research_pub_001.html
[16] RapidMind developer site, https://developer.rapidmind.com/sample-code/matrix-multiplication-samples/rm-sgemm-gpu-5938.zip
[17] http://hmmer.janelia.org/
[18] http://mpihmmer.org/
[19] NAMD Input set: http://www.ks.uiuc.edu/Research/namd/utilities/apoa1.tar.gz
[20] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. Journal of Computational Chemistry, 26:1781-1802, 2005. http://www.ks.uiuc.edu/Research/namd/
[21] HPL – High-Performance LINPACK, http://www.netlib.org/benchmark/hpl/
[22] "Evaluation of ClearSpeed Accelerators for HPC", I. N. Kozin, STFC technical report DL-TR-2009-1, http://www.cse.scitech.ac.uk/disco/publications/DLTR-2009-001.pdf
[23] Gilles Kempf, Aad J. van der Steen, Christian Caremoli, Wei-Ying Thang: Simulation of Scientific Programs on Parallel Architectures, HPCN Conference, Brussels, 1996.
[24] http://software.intel.com/en-us/articles/intel-mpi-benchmarks/
[25] http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess
[26] https://ftg.lbl.gov/ApeX/ApeX.shtml
[27] http://www.cs.virginia.edu/stream/
[28] http://users.bigpond.net.au/CPUburn/
[29] www.cs.utk.edu/~mucci/DOD/cachebench.ps
[30] http://www.iozone.org/docs/IOzone_msword_98.pdf
[31] V. Springel, "The cosmological simulation code GADGET-2", Mon. Not. R. Astron. Soc. 364, 1105-1134 (2005).
[32] http://www.mpa-garching.mpg.de/gadget/
[33] JAMES C. PHILLIPS et al., "Scalable Molecular Dynamics with NAMD", 2005 Wiley Periodicals, Inc. J Comput Chem 26: 1781–1802, 2005
[34] Alexandros Stamatakis, Michael Ott, and Thomas Ludwig:``RAxML-OMP: An Efficient Program for Phylogenetic Inference on SMPs". In Proceedings of 8th International Conference on Parallel Computing Technologies (PaCT2005), Volume 3606 of Lecture Notes in Computer Science, 288-302, Springer Verlag, September 2005.
[35] http://www.cse.scitech.ac.uk/ccg/software/DL_POLY/
[36] GROMACS : http://www.gromacs.org/

[37] M. Allalen, M. Brehm and H. Stüben, Performance of quantum chromodynamics (QCD) simulations on the SGI Altix 4700, COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY 14(2), 69-75 (2008).
[38] http://arxiv.org/pdf/0911.2174
[39] Green 500 List: http://www.green500.org/lists/2009/11/top/list.php
[40] Matrix Market: http://math.nist.gov/MatrixMarket/
[41] I. Christadler, V. Weinberg, RapidMind: Portability across Architectures and its Limitations, to appear in: Facing the Multicore Challenge (Conference Proceedings), Heidelberg, 2010. http://arxiv.org/abs/1001.1902
[42] D.Brinkers. 2010. Multigrid Algorithms on QPACE. Diploma Thesis, University of Erlangen-Nuremberg.
[43] TifaMMy (TifaMMy isn't the fastest Matrix Multiplication, yet), http://tifammy.sourceforge.net,version 2.2.0.
[44] Heinecke, A., Bader M.: Parallel matrix multiplication based on space-filling curves on shared memory multicore platforms, Proceedings of the 2008 ACM Research Computing Frontiers Conference and collocated workshops: MAW'08 and WRFT'08, p. 385-392, 2008.
[45] Heinecke, A., Bader M.: Towards many-core implementation of LU decomposition using Peano Curves, Proceedings of the 2009 ACM Research Computing Frontiers Conference and co-located workshops: Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop, p. 21-30, 2009.
[46] OpenMP 3.0: The OpenMP API specification for parallel programming, http://openmp.org/, 2009.
[47] Michael Abrash: A First Look at the Larrabee New Instructions (LRBni), Dr Dobbs Journal, April 1, 2009.
[48] http://blogs.intel.com/technology/2010/05/an_update_on_our_graphics-rela.php

List of Acronyms and Abbreviations

ACF	Advanced Computing Facility
ACML	AMD Core Math Library
ADP	Average Dissipated Power
AMD	Advanced Micro Devices
APGAS	Asynchronous PGAS (language)
API	Application Programming Interface
APML	Advanced Platform Management Link (AMD)
ASIC	Application-Specific Integrated Circuit
ATI	Array Technologies Incorporated (AMD)
BAdW	Bayerischen Akademie der Wissenschaften (Germany)
BCO	Benchmark Code Owner
BLAS	Basic Linear Algebra Subprograms
BSC	Barcelona Supercomputing Center (Spain)
BTL	Byte Transfer Layer
CAF	Co-Array Fortran
CAL	Compute Abstraction Layer
CCE	Cray Compiler Environment
ccNUMA	cache coherent NUMA
CEA	Commissariat à l'Energie Atomique (represented in PRACE by GENCI, France)
CGS	Classical Gram-Schmidt
CGSr	Classical Gram-Schmidt with re-orthogonalisation
CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CINES	Centre Informatique National de l'Enseignement Supérieur (represented in PRACE by GENCI, France)
CLE	Cray Linux Environment
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CSR	Compressed Sparse Row (for a sparse matrix)
CSXL	ClearSpeed math Library
CT	Current Transformer
CUDA	Compute Unified Device Architecture (NVIDIA)
DARPA	Defense Advanced Research Projects Agency
DDN	DataDirect Networks
DDR	Double Data Rate
DGEMM	Double precision General Matrix Multiply
DIMM	Dual Inline Memory Module
DMA	Direct Memory Access

DNA	DeoxyriboNucleic Acid
DP	Double Precision, usually 64-bit floating point numbers
DRAM	Dynamic Random Access Memory
EC	European Community
EESI	European Exascale Software Initiative
EoI	Expression of Interest
EP	Efficient Performance, e.g., Nehalem-EP (Intel)
EPCC	Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
EPSRC	The Engineering and Physical Sciences Research Council (United Kingdom)
eQPACE	extended QPACE, name of the FZJ WP8 prototype
ETHZ	Eidgenössische Technische Hochschule Zuerich, ETH Zurich (Switzerland)
EX	Expandable, e.g., Nehalem-EX (Intel)
FC	Fiber Channel
FFT	Fast Fourier Transform
FHPCA	FPGA HPC Alliance
FP	Floating-Point
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
FZJ	Forschungszentrum Jülich (Germany)
GASNet	Global Address Space Networking
GB	Giga (= 2^{30} ~ 10^9) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^9) bits per second, also Gbit/s
GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GDDR	Graphic Double Data Rate memory
GENCI	Grand Equipement National de Calcul Intensif (France)
GFlop/J	Giga (=10^9) Floating point operations (usually in 64-bit, i.e. DP) per Joule
GFlop/s	Giga (= 10^9) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency = 10^9 periods or clock cycles per second
GigE	Gigabit Ethernet, also GbE
GLSL	OpenGL Shading Language
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPU	Graphic Processing Unit
GRU	Global Register Unit (SGI, Altix UV)
GS	Gram-Schmidt
GUI	Graphical User Interface
GUPS	Giga (= 10^9) Updates Per Second
GWU	George Washington University, Washington, D.C. (USA)
HBA	Host Bus Adapter
HCA	Host Channel Adapter
HCE	Harwest Compiling Environment (Ylichron)

HDD	Hard Disk Drive
HE	High Efficiency
HLL	High Level Language
HMM	Hidden Markov Model
HMPP	Hybrid Multi-core Parallel Programming (CAPS enterprise)
HP	Hewlett-Packard
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HPCC	HPC Challenge benchmark, http://icl.cs.utk.edu/hpcc/
HPCS	High Productivity Computing System (a DARPA programme)
HPL	High Performance LINPACK
HT	HyperTransport channel (AMD)
HWA	HardWare Accelerator
IB	InfiniBand
IBA	IB Architecture
IBM	Formerly known as International Business Machines
ICE	(SGI)
IDRIS	Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
IEEE	Institute of Electrical and Electronic Engineers
IESP	International ExasSale Project
IL	Intermediate Language
IMB	Intel MPI Benchmark
I/O	Input/Output
IOR	Interleaved Or Random
IPC	Instruction Per Cycle
IPMI	Intelligent Platform Management Interface
IWC	Inbound Write Controller
JSC	Jülich Supercomputing Centre (FZJ, Germany)
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
KTH	Kungliga Tekniska Högskolan (represented in PRACE by SNIC, Sweden)
LB	Load Balance
LBE	Lattice Boltzmann Equation
LINPACK	Software library for Linear Algebra
LLNL	Lawrence Livermore National Laboratory, Livermore, California (USA)
LOV	Logical Object Volume
LQCD	Lattice QCD
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
LS	Local Store memory (in a Cell processor)
LU	Logical Unit
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MDT	MetaData Target

MFC	Memory Flow Controller
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MGS	Modified Gram-Schmidt, also, ManaGement Server (Lustre file system)
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MIC	Many Integrated Core (Intel)
MIPS	Originally Microprocessor without Interlocked Pipeline Stages; a RISC processor architecture developed by MIPS Technology
MKL	Math Kernel Library (Intel)
ML	Maximum Likelihood
Mop/s	Mega (= 10^6) operations per second (usually integer or logic operations)
MPI	Message Passing Interface
MPP	Massively Parallel Processing (or Processor)
MPT	Message Passing Toolkit
MRAM	Magnetoresistive RAM
MTAP	Multi-Threaded Array Processor (ClearSpead-Petapath)
MTBF	Mean Time Between Failure
MUPS	Mega (= 10^6) Updates Per Second
mxm	DP matrix-by-matrix multiplication `mod2am` of the EuroBen kernels
NAND	Logical operation NOT AND, also NAND flash memory, a type of memory technology
NAS	Network-Attached Storage
NCF	Netherlands Computing Facilities (Netherlands)
NDA	Non-Disclosure Agreement
NoC	Network-on-a-Chip
NFS	Network File System
NIC	Network Interface Controller
NUMA	Non-Uniform Memory Access or Architecture
OFED	OpenFabric Enterprise Distribution
OpenCL	Open Computing Language
OpenGL	Open Graphic Library
Open MP	Open Multi-Processing
OS	Operating System
OSS	Object Storage Server
OST	Object Storage Target
PBS	Portable Batch System
PCIe	Peripheral Component Interconnect express, also PCI-Express
PCI-X	Peripheral Component Interconnect eXtended
PCRAM	Phase-change memory, also PCM or PRAM
PGAS	Partitioned Global Address Space
PGI	Portland Group, Inc.
pNFS	Parallel Network File System
POSIX	Portable OS Interface for Unix

PPE	PowerPC Processor Element (in a Cell processor)
PPL	Parallel Programming Laboratory
PPU	Power Processor Unit (in a Cell processor)
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PSNC	Poznan Supercomputing and Networking Centre (Poland)
PSU	Power Supply Unit
QCD	Quantum Chromodynamics
QCDOC	Quantum Chromodynamics On a Chip
QDR	Quad Data Rate
QPACE	QCD Parallel Computing on the Cell
QR	QR method or algorithm: a procedure in linear algebra to compute the eigenvalues and eigenvectors of a matrix
RAM	Random Access Memory
RDMA	Remote Data Memory Access
RIDS	Reference Input Data Sets
RISC	Reduce Instruction Set Computer
RNG	Random Number Generator
RM	RapidMind language
RMS	Record Management Services (Open VMS)
RPM	Revolution per Minute
SAN	Storage Area Network
SARA	Stichting Academisch Rekencentrum Amsterdam (Netherlands)
SAS	Serial Attached SCSI
SATA	Serial Advanced Technology Attachment (bus)
SDK	Software Development Kit
SDSC	San Diego Supercomputing Center
SGEMM	Single precision General Matrix Multiply, subroutine in the BLAS
SGI	Silicon Graphics, Inc.
SHMEM	SHare MEMory access library (Cray)
SIMD	Single Instruction Multiple Data
SM	Streaming Multiprocessor, also Subnet Manager
SMC	SGI Management Center
SMT	Simultaneous MultiThreading
SMP	Symmetric MultiProcessing
SNIC	Swedish National Infrastructure for Computing (Sweden)
SP	Single Precision, usually 32-bit floating point numbers
SPE	Synergistic Processing Element (core of Cell processor)
SPH	Smoothed Particle Hydrodynamics
spmxv	DP sparse-matrix-by-vector multiplication `mod2as` of the EuroBen kernels
SPU	Synergistic Processor Unit (in each SPE)
SSD	Solid State Disk or Drive
SSE	Streaming SIMD Extensions (Intel)

STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
STRATOS	PRACE advisory group for STRAtegic TechnOlogieS
STT	Spin-Torque-Transfer
TARA	Traffic Aware Routing Algorithm
TB	Tera (= 2^{40} ~ 10^{12}) Bytes (= 8 bits), also TByte
TCB	Theoretical and Computational biophysics Group
TDP	Thermal Design Power
TFlop/s	Tera (= 10^{12}) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
TSMC	Taiwan Semiconductor Manufacturing Company, Limited
TUM	Technical University Munich
UFM	Unified Fabric Manager (Voltaire)
UPC	Unified Parallel C
UV	Ultra Violet (SGI)
VHDL	VHSIC (Very-High Speed Integrated Circuit) Hardware Description Language
VPU	Vector Processing Unit
WP	PRACE Work Package
WP5	PRACE Work Package 5 – Deployment of prototype systems
WP6	PRACE Work Package 6 – Software enabling for Petaflop/s systems
WP8	PRACE Work Package 8 – Future Petaflop/s computer technologies beyond 2010

Executive Summary

The PRACE project [1] started a process in its Work Package 8 (WP8) aiming at the evaluation of components and technologies that can be useful or will be required for future High Performance Computing (HPC) systems. This process will be continued in the permanent pan-European Research Infrastructure. It includes technology surveys and prototype evaluation.

During the first year of the PRACE project, WP8 identified the most promising systems and components that should be evaluated in depth and worked on. Based on the results of this survey, PRACE partners proposed a number of prototypes for detailed evaluation which have been evaluated by external experts to obtain a wide coverage of technologies and to exploit the specific expertise of the partners. Finally twelve WP8 prototypes were selected by PRACE and have been approved by the European Commission (EC) and their HPC technology experts. These prototypes include:

- Programming models/languages such as Chapel, CUDA, RapidMind, HMPP, OpenCL, StarSs(CellSs), UPC and CAF;
- Computing nodes based on NVIDIA as well as AMD GPUs, Cell, ClearSpeed-Petapath, FPGAs, Intel Nights Ferry co-processor and Intel Nehalem and AMD Barcelona and Istanbul processors;
- Systems to assess memory bandwidth, interconnect, power consumption and I/O performance as well as hybrid and power-efficient solutions.

In deliverable D8.3.1 [2] the installation status of the prototypes (hardware and software) and evaluation work performed until mid of September 2009 is described in detail. This deliverable summarises all evaluation results obtained by WP8 during the PRACE PP project and tries to rate the applicability of these different technologies for future European Tier-0 HPC systems.

Two components that have been mainly addressed and evaluated are the programming models and accelerator based nodes. Although some of the prototypes used specific kernels or applications (GADGET, NAMD etc.) in the evaluation, an effort was made to use common benchmark kernels to make the results comparable. WP8 selected 4 kernels from the EuroBen benchmark suite, a dense matrix/matrix multiplication (mod2am), a sparse matrix/vector multiplication (mod2as), a 1-D radix-4 fast Fourier transformation (mod2f) and a random number generator (mod2h) to be ported to different programming languages and processing architectures. These kernels are sufficiently simple to be ported to different programming languages and still do provide significant information concerning ease of use as well as performance.

The experience does show that the performance achievable with the accelerator-based approach is promising but very dependent on the actual coupling between the characteristics of the algorithm and the device. One of the major bottlenecks, the memory bandwidth between the host processor and accelerator has to be addressed in future implementations eventually leading to tightly coupled heterogeneous processing cores in a node. Also the programmability of these devices is still in general far from ideal especially if aiming at very high performance and non trivial codes. At the network level, interconnects with dynamic rerouting capability based on the network load are needed. Also new programming models helping the user to more easily develop and scale parallel applications as well as tools for performance analysis and prediction will be of critical importance in order to be able to efficiently use future European multi-Petascale systems. Equally important will be the development of new scientific libraries which are able to address different optimization criteria such as performance or energy efficiency and support new programming models. Concerning system architecture,

both homogeneous and accelerated clusters with ten thousands compute nodes as well as massively parallel systems with several hundred thousand low power compute nodes seem to be the dominating architectures for the next five years.

1 Introduction

To stay at the forefront of High Performance Computing, PRACE Work Packages WP7 and WP8 have collaboratively started a process to work with technology leaders to prepare for the next generations of multi-Petaflop/s systems. This includes novel components and software technologies to exploit next generation architectures. Close contacts have been established with all major HPC system vendors as well as more than 26 technology providers which are active in areas such as communications network, mass storage and file systems, memory, processing units, programming languages and compilers, system software and tools. The ultimate goal is to engage European industry to develop products for the strategic HPC market and to catch-up and eventually excel in selected fields.

This is not a one-time activity but a continuous process which will be an integral part of the permanent pan-European Research Infrastructure. A major milestone towards the establishment of PRACE as a competent technology partner was the creation of STRATOS (see deliverable D8.1.4 [3]), the PRACE advisory group for Strategic Technologies. STRATOS, founded by twelve PRACE partners and the open Interest Group of European and International users, providers and technology drivers "PROSPECT e.V.", is open to and encourages participation by all PRACE partners but also by other organizations or individuals with an interest to contribute to the evolution of European HPC. With the accession of the French consortium Ter@tec, made up of IT companies, research institutes and industrial users, STRATOS has become a unique European HPC Coordination, Research and Development Consortium consisting of European supercomputing centres and (through PROSPECT e.V. and Ter@tec) over 60 European and world-wide technology providers.

During the first year of the PRACE project, WP8 identified the most promising systems and components that should be evaluated in depth and worked on. With the documentation of these findings in deliverable D8.2.2 [4], a second major milestone of WP8 was reached at the end of 2008. Finally twelve WP8 prototypes were selected by PRACE (see D2.7.2 [5] and its addendum [6]) and have been approved by the European Commission (EC) and their HPC technology experts. These prototypes include:

- Programming models/languages such as Chapel, CUDA, RapidMind, HMPP, OpenCL, StarSs(CellSs), UPC and CAF;
- Computing nodes based on NVIDIA as well as AMD GPUs, Cell, ClearSpeed-Petapath, FPGAs, Intel Nights Ferry co-processor and Intel Nehalem-EP and AMD Barcelona and Istanbul processors;
- Systems to assess memory bandwidth, interconnect, power consumption and I/O performance as well as hybrid and power-efficient solutions.

Their evaluation has led to new insights which will be reported in detail in this document. Due to late delivery of some prototypes the work on these technologies could not be completed in 2009. This deliverable summarizes all work performed by WP8 during the PRACE Preparatory Phase project and it's extension period. The following chapters and subsections of this document contain completely new or updated information:

- Chapters 2.1.3, 2.1.4, 2.1.10, 2.2.1

- Chapter 3.1.6 LRZ + CINES (Phase 2)
- Chapter 3.1.7 Intel MIC Architecture
- Chapter 3.3.1 Triads (RINF1) benchmark results (BAdW-LRZ)
- Chapter 3.3.2 Random Access Results
- Chapter 3.4 Inter Node Communication Network
- Chapter 3.5 PGAS Languages
- Chapter 3.7.1 The SNIC-KTH system
- Chapter 3.7.2 PSNC results
- Chapter 3.7.3 STFC results
- Chapter 3.9.3 Network bandwidth

1.1 Scope and Structure of the Report

This deliverable summarises all evaluation results obtained by WP8 so far and tries to rate the applicability of these different technologies for future European Tier-0 HPC systems. The document also aims at suggesting directions of potential research interest in order to develop European technology towards the Exaflop/s era.

In Chapter 2 each prototype together with its key evaluation objectives are briefly described.

Chapter 3 describes the experiments that have been performed on the different WP8 prototypes. The advanced nature of the WP8 prototypes makes a strongly structured approach for the assessment, as it was taken by WP5/7 impossible. The assessment reports are arranged under several sub-chapters such as “GPGPU Programming Languages and Accelerated Compilers”, “Hybrid Programming Models”, “Intra-node Bandwidth”, “Inter-node Communication Network”, “PGAS Languages”, “Novel I/O” and “Energy Efficiency” as far as their evaluation objective matches with these categories. Since the influence of the communication network and processor performance on application scaling often cannot be tested on real hardware because future Tier-0 systems are expected to contain new hard- and software components which are not available at the time important system design decisions will have to be made, performance prediction tools might become a key system design technology for the PRACE Legal Entity in future. Section 3.8 describes the experiment to identify issues that may be relevant when scaling or porting applications to future architectures. Finally, a short summary of the evaluation results together with concluding remarks are given in Section 3.9.

In Chapter 4 recommendations for future multi-Petascale architectures are given based on the prototype assessment and the PRACE technology surveys.

Chapter 5 gives final conclusions and remarks.

Chapter 6, an annex, gives a brief description of the benchmarks and applications used for the prototype evaluations and for the research activities listed in Section 2.2.

2 WP8 prototypes and Research Activities

2.1 Prototypes

All WP8 prototypes are addressing the area of the emerging technologies for Petascale systems. Their aim is to investigate a specific and vertical set of hardware and software issues. In this section the prototypes acquired for evaluation within Work Package 8 are succinctly described. In addition, the lead partners and the key evaluation objectives for each prototype are listed. Section 2.2 describes common research activities of WP8 parties on WP7 and WP8 prototype hardware.

2.1.1 *eQPACE*

Key Objective: Evaluation and extension of special-purpose cluster architecture QPACE regarding applications beyond QCD.

Lead Partner: FZJ.

QPACE (Quantum Chromodynamics Parallel Computing on the Cell) is a massively parallel and scalable computer architecture optimized for lattice Quantum Chromodynamics (LQCD) which is developed among several academic institutions (SFB TR 55) and the IBM development lab in Böblingen, Germany.

Figure 1: QPACE Architecture

The building block of QPACE is a node card comprising an IBM PowerXCell 8i processor and a custom FPGA-based network processor. The 3D-torus network supports SPE-centric nearest-neighbour communication between the SPE cores of adjacent nodes and data transfer between its Local Stores.

32 node cards are mounted on a single backplane. One dimension of the 3D-torus network is completely routed within the backplane where the nodes are arranged as one partition of 1 × 4 × 8 nodes or as multiple smaller partitions. For larger partitions, cables interconnect several backplanes. Eight backplanes reside in one rack, hosting a total of 256 node cards corresponding to an aggregate peak performance of 26 TFlop/s (DP).

The 3D-torus network interconnects the node cards with nearest-neighbour communication links driven by a lean custom protocol optimized for low latencies. For the physical layer of

the links 10 Gigabit Ethernet PHYs[1] are used providing a bandwidth of 1 GB/s per link and direction.

On each node card, the network processor is also connected to a Gbit-Ethernet transceiver. The Ethernet ports of all node cards connect to standard Ethernet switches housed within the QPACE rack. Depending on the I/O requirements, the Ethernet bandwidth between a QPACE rack and a front-end system can be adjusted by changing the bandwidth of the uplinks of the switches.

The backplanes in the rack, which holds 32 node cards, are connected to a liquid-cooled cold-plate which acts as a heat conductor thus making QPACE a highly energy efficient ecosystem. An upper limit for the power consumption is about 125 W per node card. To remove the generated heat a cost-efficient liquid cooling system has been developed, which enables high packaging densities. The maximum power consumption of one QPACE rack is about 32 kW.

An IBM e1350 cluster serves as front-end system. It comprises login and master nodes as well as six nodes intended to provide a parallel Lustre file system with two meta data and four object storage servers. The disk storage amounts to five enclosures, each with 14 × 72 GB disks.

Goals of eQPACE are:

- Extend the features of QPACE to make the system suitable for a range of applications beyond QCD;
- Leverage the feature that an FPGA is used to implement the torus interconnects and use different versions of FPGA bitstreams;
- Extend the software stack, e.g., by development of a subset of MPI functions for the torus API.

2.1.2 *BAdW-LRZ/GENCI-CINES Phase1 (CINES Part)*

Key Objective: Evaluation of a hybrid thin node system based (Bi Nehalem-EP and ClearSpeed e710).

Lead Partner: GENCI – CINES.

The prototype installed at CINES focuses on assessing the ClearSpeed accelerator, one characteristic of which is energy efficiency. The evaluation process has to take advantage of both node processors and accelerator, by determining the optimal repartition of computing load has to be found.

The platform is integrated in the Tier-1 HPC platform JADE (128 TFlop/s LINPACK), sharing Lustre parallel file system, PBSPro Batch scheduler.

The targeted SGI-ICE / ClearSpeed-Petapath platform (Figure 2) configuration is:

```
SGI hosting platform:
32 Blades XE
64 Processors Intel Nehalem-EP
256 Cores
4 GB per core
Infiniband QDR
Estimated peak performance is 2.53 TFlop/s

ClearSpeed-Petapath accelerators:
32 x e710 card
```

[1] PHY is a chip where the lowest PHYsical layer of a network is implemented

```
One per ICE blade
4 GFlop/s / Watt
Software development toolkit
```

Estimated peak performance is 3 TFlop/s. The total accumulated peak performance is 5.53 TFlop/s.

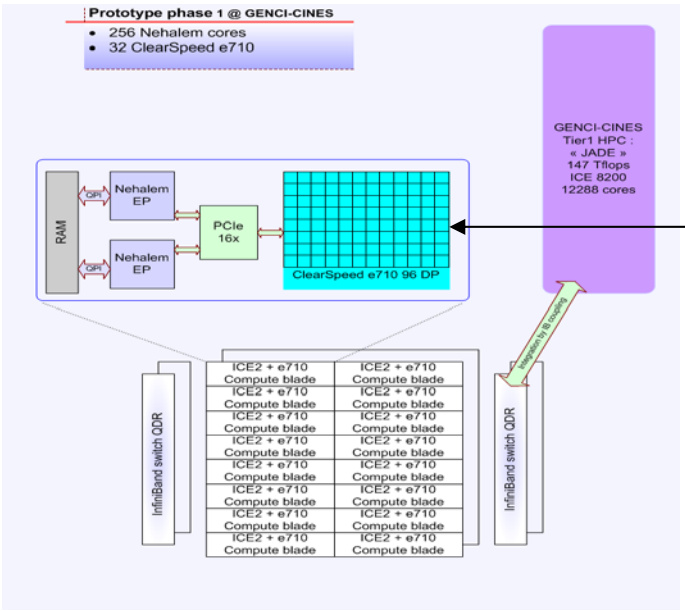

Figure 2: Prototype configuration and integration layout

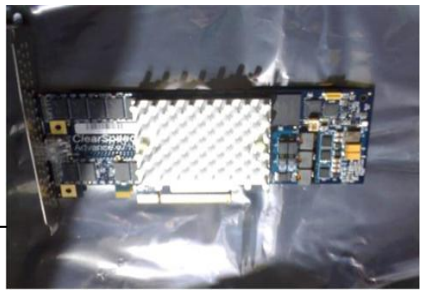

Figure 3: ClearSpeed CSX 710 card

2.1.3 *BAdW-LRZ/GENCI-CINES Phase2 (LRZ Part)*

Key Objective: Evaluation of a hybrid system architecture containing thin nodes, fat nodes and compute accelerators with a shared file system.

Lead Partner: Leibniz Supercomputing Centre (BAdW-LRZ).

The joint prototype proposal from BAdW-LRZ (referred to as LRZ) and GENCI-CINES (referred to as CINES) shown in Figure 4 focuses on the last generation systems from SGI which are able to scale up to peak performances of multi-PFlop/s. It is composed of ccNUMA (UltraViolet, UV) and MPP (Altix ICE 8200LX) systems and accelerated with ClearSpeed-Petapath Advance e710 and nVIDIA Tesla 1060 accelerators[2].

While the focus of the CINES prototype lies in the evaluation of using ClearSpeed-Petapath boards to accelerate real applications and enhance their energy efficiency dramatically, the main objective of the LRZ part is to assess the advantages of hybrid systems for the LRZ workload. The evaluation focus of this approach divides up into the following axes of assessment:

- Hybrid system consisting of Intel Nehalem-EP-based thin nodes and Nehalem-EX-based fat nodes with ClearSpeed-Petapath and nVIDIA compute accelerators;
- Performance/Power efficiency and ease of use for different scientific areas;
- Support for new programming languages like CAF, UPC and RapidMind.

[2] Originally it was planned to evaluate the suitability of Intel Larrabee GPGPUs for accelerating HPC applications. Since Intel changed its product plans [48], LRZ started to assess GPGPU programming with nVIDIA hardware. During the PRACE extension phase, Intel provided LRZ with a prototype software development vehicle for the new Intel Many Integrated Cores (MIC) architecture.

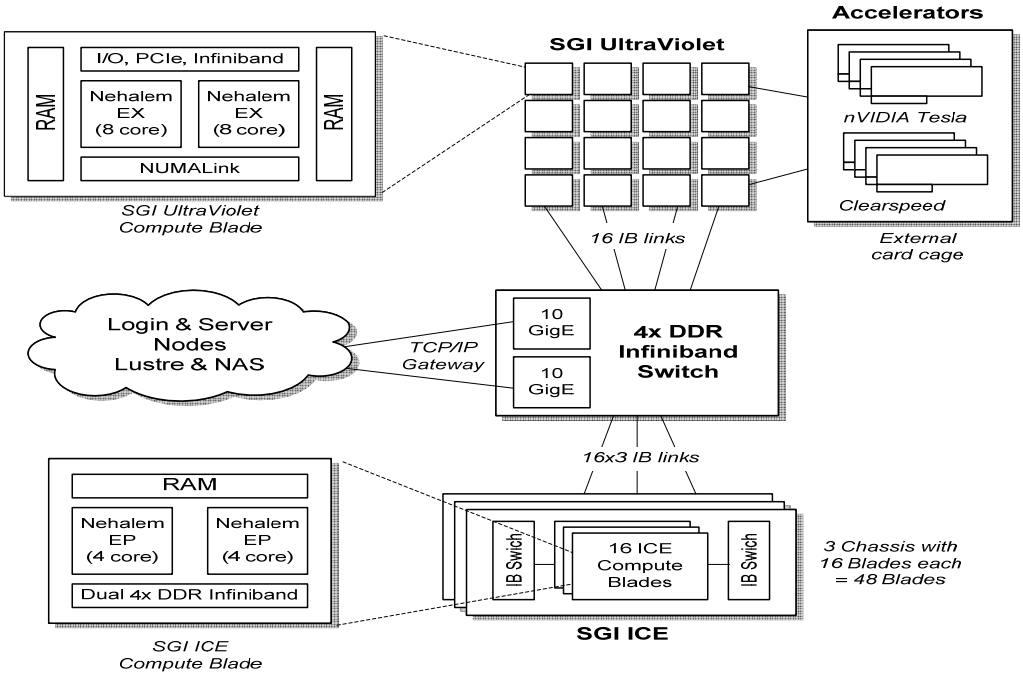

Figure 4: Hybrid system prototype installed at BAdW-LRZ

2.1.4 *Intel Many Integrated Core (MIC) architecture*

Key Objective: Evaluation of the ease of use of the Intel "Many Integrated Core" (MIC) architecture for HPC applications.

Lead Partner: Leibniz Supercomputing Centre (BAdW-LRZ).

Figure 5 shows a scheme of the Intel MIC architecture prototype. It uses 32 in-order x86-based processor cores that are augmented by a 512 bit wide vector processing unit (VPU) (see Figure 6). All cores are interconnected by a bi-directional on-chip ring network. This ring network also provides a path for the L2 caches to access memory. The L2 cache is shared between all cores. The instruction decoder of the Intel MIC architecture supports the standard Pentium x86 instruction set as well as new instructions for explicit cache control and vector type operations.

Each core has access to its 256 KByte local subset of a coherent L2 cache. The L1 cache sizes are 32 KByte for instructions and 32 KByte for data. Furthermore each scalar unit supports 4 threads of execution, with separate register sets per thread. Hence thread switching can be extensively used to cover cases such as pipeline stalls and cache misses.

Figure 7 shows a block diagram of the 16-wide vector processing unit. A set of new VPU instructions allow a variety of instructions on both integer and floating point data types such as:

- Standard arithmetic operation including fused multiply-add;
- Standard logical operations;
- Gather and scatter operations.

The VPU instructions allow up to three source operands, one of which can directly come from the L1 cache. 8-bit and 16-bit integer and floating point numbers can be directly converted to 32-bit integer and floating point numbers without loss of performance. The VPU supports swizzling of register data in a variety of ways and replication of data from memory across the VPU lanes.

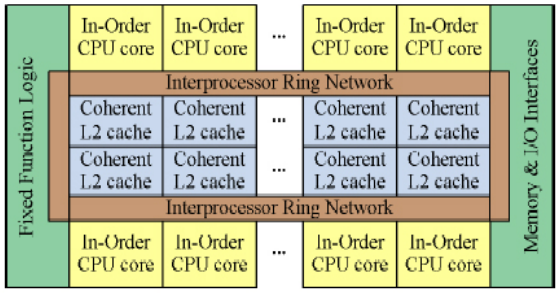

Figure 5: Scheme of Intel MIC architecture prototype.

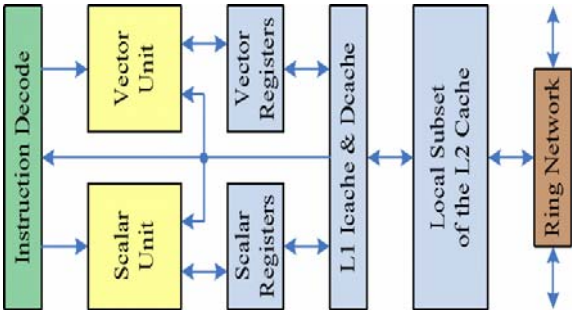

Figure 6: x86-based core and associated system blocks

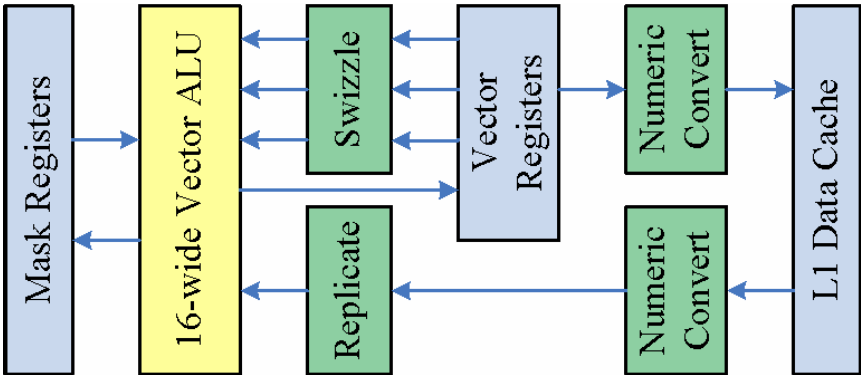

Figure 7: Block diagram of vector processing unit (VPU).

The focus on the assessment of this Intel MIC architecture is to evaluate the ease of use of this parallel SMP cache coherent prototype processor platform for HPC.

2.1.5 *ClearSpeed-Petapath*

Key Objective: Evaluation of the Petapath Feynman e740 and e780 prototype products based on ClearSpeed technology within cluster configuration.

Lead Partner: NCF.

The configuration is given in the Figure 8. The light blue boxes labelled "Feynman e740/e780" contain 4 or 8 ClearSpeed CSX700 processors for a total of 112. The dark blue HP DL170 host nodes each connect to two Feynman boxes by PCI Express Gen. 2 16× (8 GB/s).

The HP host nodes are connected by 4× DDR Infiniband (IB) in order to employ them, and the attached accelerator boxes, in parallel via MPI.

Internally a Feynman e780 connects to its 8 processors through a PLX 8648 PCIe switch. The bandwidth to each individual processor is 1 GB/s. The e740 boxes contain only 4 CSX700 processors but the bandwidth to/from the individual processors is doubled to 2 GB/s. This should help for bandwidth-hungry applications. The peak performances are 768 GFlop/s and 384 GFlop/s in 8-Byte precision (DP) for a Feynman e780 and e740, respectively. The energy efficiency is very high: ≈ 4

Figure 8: Clearspeed-Petapath prototype

GFlop/s/Watt.

The HP DL170h G6 host nodes contain 2 Intel X5550 Nehalem-EP processors at 2.67 GHz and 24 GB of DDR3 memory/node. Furthermore, each host node contains dual disks of 250 and 500 GB, for system and user data, respectively. Access to the system is provided through the HP DL160 G6 head node. The system is divided in a development part (1 DL170 + 2 Feynman e740s) and an application assessment part (7 DL170s + 2 Feynman e740s + 12 Feynman e780s).

2.1.6 *Hybrid Technology Demonstrator*

Key Objective: Evaluating GPGPU and HMPP.

Lead Partner: GENCI-CEA.

The Hybrid technology Demonstrator is a platform to study the impact of GPU computing in terms of programming languages. Based on nVIDIA TESLA servers, this machine can run different programming environments such as CUDA, RapidMind, OpenCL, Ct and HMPP.

On this prototype, we have focused on the HMPP environment provided by [9] to study the impact of a directive-based programming model in terms of ease of development as well as effective performances.

The prototype is a cluster of 5 nodes. The first node is a login node featuring four Intel Nehalem-EP processors. The last four nodes are BULL R422 servers featuring 2 Intel Harpertown CPU with 16 GB of RAM each. The BULL servers are connected through a PCI-Express 16× to two nVIDIA TESLA servers. Interconnects between the nodes are Infiniband DDR.

A TESLA server has 2 PCI-Express 16× links with 2 C1060 graphic boards on each. A C1060 features 30 multiprocessors at 1.3 GHz and has 4 GB of memory. One multiprocessor has 8 single precision (SP) units and 1 double precision (DP) unit. Therefore, a C1060 can have 240 SP (30 DP) units running in parallel. The peak performance of a C1060 is 78 GFlop/s DP and 624 GFlop/s SP.

The main component to be assessed is HMPP since we consider the TESLA as a product which has already been extensively studied. New languages such as OpenCL or Ct were not available at the beginning of this project.

We run the HMPP Workbench software, release 2.1.0sp1 as of the writing of this report. We started our evaluation with release 1.5.3. Since HMPP is a source code generator (a pre-processor) we use the usual Intel compiler (version 11.1) for the host program and the CUDA 2.3 environment for the TESLA part of the software.

HMPP relies on directives which are in the form of special comments in the source code for FORTRAN or pragmas for C. HMPP has the ability to generate code for different types of targets. Here we will focus only on the CUDA one as it is the native environment for a nVIDIA TESLA platform. A future study of different targets (for example CAL/IL) needs to be done.

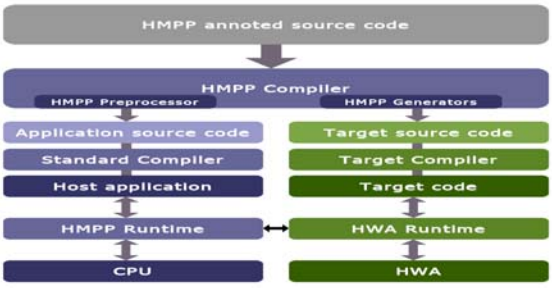

Figure 9: Overview of HMPP components

Our work will consist in adding directives to the proposed codes, evaluate the difficulty of this task and then modify the codes to get the best performances with the HMPP/CUDA driver.

For comparison sake, we have concentrated our efforts on two of the EuroBen benchmarks. We have selected mod2am and mod2as because they are very frequent kernels in scientific codes. Mod2am implements a matrix multiplication which can be easily compared with some vendor libraries (here the CUBLAS one). Mod2as implements a reduction which is well known for being hard to parallelize. Therefore it appeared to us that these two kernels would be good challenges for the HMPP software.

2.1.7 *Maxwell FPGA*

Key Objective: Evaluate the performance and usability of HARWEST Compiling Environment.

Lead Partner: EPSRC-EPCC.

Objective

Our objective is to evaluate the performance and usability of state-of-the-art FPGA programming tools. We plan to port the 4 EuroBen kernels to the FPGA hardware in Maxwell using a C-to-gates compiler, the Harwest Compilation Environment from Ylichron [10].

We will evaluate the performance of the ported kernels, compared to the original kernels running in software. Where possible we will also compare their performance to that of a hand-coded VHDL (Very-High Speed Integrated Circuit Hardware Description Language) implementation of similar algorithms. Finally, we will evaluate the usability of the compiler (how much effort it takes to port a C program to it, how much specialist hardware knowledge is required to get an adequate result, etc.).

Architecture

In 2007, the FPGA High Performance Computing Alliance (FHPCA) completed work on the Maxwell – a pre-production technology demonstrator system built using 32 accelerator cards from Alpha Data Ltd and 32 from Nallatech Ltd using Virtex-4 FPGAs supplied by Xilinx Corp. These accelerator cards are hosted in a 32 node IBM HS21-based BladeCenter system. Each node therefore consists of a single-core 2.8 GHz Xeon processor with two accelerator cards connected on a PCI-X bus. The main novelty in the design of the system is its use of the RocketIO connections on the FPGAs to create a nearest neighbour two-dimensional mesh interconnect between the FPGAs. This can allow a complete numerical algorithm to be executed in parallel with no need for the FPGAs to communicate over the (relatively) slow PCI-X bus.

Major components to be assessed

FPGA acceleration of high performance codes has generated a lot of interest in recent years due to the potential gains in speed and reductions in power consumption. However, the traditional method of programming FPGAs using hardware description languages requires much specialist knowledge and understanding, and is therefore not typically usable by scientists, being challenging even for experienced software engineers.

Various tools exist which attempt to simplify the process. Many of them are C-to-gates compilers, which take some subset of the C programming language (so as to be as familiar as possible to scientists and programmers) and attempt to compile it into a hardware representation. In this project we aim to assess one such tool, the Harwest Compilation Environment (HCE) from Ylichron, to determine:

- How usable it is by programmers without specialist hardware knowledge;
- How the performance of the code it produces compares to software (and, where possible, traditional hand-coded FPGA designs).

2.1.8 *XC4-IO*

Key Objective: Evaluate the performance of I/O and Lustre file system and advantages of using SSD technology for metadata.

Lead Partner: CINECA.

The prototype mainly investigates innovative aspects of I/O in two directions:

- Advantages given by accessing meta data using SSD technology;
- Comparison and performance benchmarks using Lustre file system in its native configuration or in combination with NFS and pNFS over RDMA technologies.

The architecture of the prototype is composed by three main parts, as represented in Figure 10: A Storage Area Network, a cluster of I/O servers equipped with 8 blades (each blade equipped with double link FC HBA 4 Gb/s), and an HPC Blade System cluster (HP Woodcrest dual core Xeon Linux Cluster - XC4).

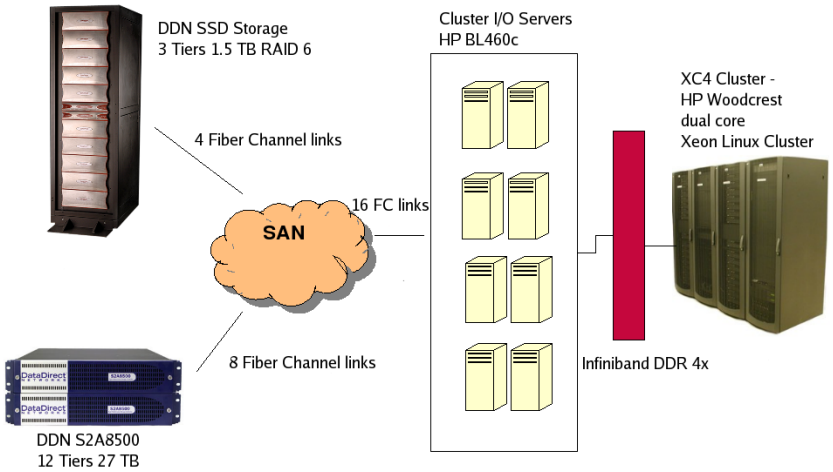

Figure 10: I/O & File System Prototype architectural scheme

The SAN is composed of two different types of devices: one for metadata storing (DDN S2A9900 with SSD storage technology) and the other one in charge for the actual data infrastructure, used by the applications (DDN storage S2A8500). These devices are double linked with a cluster of I/O servers using Fiber Channel connections, ensuring a theoretical bandwidth up to 64 Gb/s from I/O cluster to the storage infrastructure; each Fiber Channel (FC) provides a bandwidth of 4 Gb/s.

One of the most important aspects of this prototype is metadata management analysis. Meta data are stored in Intel X25-E Extreme SATA Solid-State Drives (SSD) and the access to this storage structure is controlled by DDN S2A9900. The whole infrastructure is composed by 30 disks for a global storage amount of 1.5 TB only for metadata. The S2A9900 manages a coherent flow of data throughout the SAN from users to storage, transferring data at speeds of up to 3 GB/s. Each drive controlled by S2A9900 is an Intel X25-E Extreme SATA SSD with 64 GB of capacity. These drives are designed using the latest-generation native SATA interface with an advanced architecture employing 10 parallel NAND flash channels equipped with single-level cell NAND flash memory.

Data used by applications are actually stored in a DDN S2A8500 device. In the prototype architecture, DDN S2A8500 is presented in coupled configuration, organized in 12 tiers for a whole amount of 27 TB of data. The DDN S2A8500 is equipped with eight 2 Gb/s Fiber Channel ports, for a overall bandwidth of 16 Gb/s viewed from the cluster of I/O servers.

In order to manage requests from clients applications, the I/O subsystem is completed using a cluster equipped with 8 blades (I/O server), each blade is an HP ProLiant BL460c G6 server blade (Intel Xeon Processor X5570 2.93 GHz, 16 GB RAM).

To perform computational activities the prototype uses an HP BladeSystem cluster (XC4): HP Woodcrest dual core Xeon Linux Cluster (HP XC6000) equipped with 1024 cores at 3.0 GHz, 2 TB RAM, Infiniband.

2.1.9 *SNIC-KTH*

Key Objective: Study of energy efficiency achievable using un-accelerated nodes based entirely on commodity parts and commodity interconnects for cost efficiency and minimal impact on programming model.

Lead Partner: SNIC-KTH.

Figure 11: Prototype's motherboard

Figure 12: 10-blade chassis

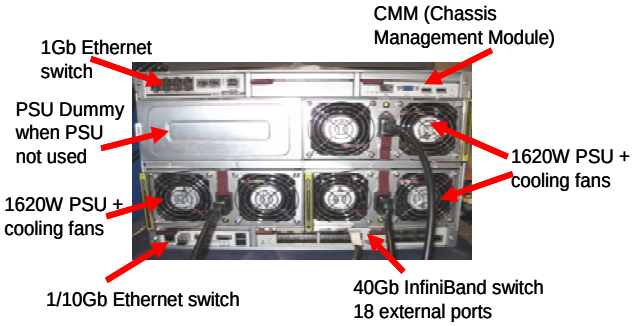

Figure 13: Chassis features

The prototype consists of new blades and a new chassis designed by Supermicro for the prototype. The blades have 4 CPU sockets each with 4 DIMM slots, support for PCI Express-Gen2 and AMD's APML (Advanced Platform Management Link). Each blade has four 6-core, 2.1 GHz, 6 MB cache, 55 W ADP High Efficiency (HE) CPUs and two 4 GB DIMMs/socket. Each 7U chassis is equipped with 10 blades achieving a density of 5.7 sockets/U, a 36-port QDR IB switch with 18 external ports, a 1/10 GigE switch and three 1620 W power supplies with a peak efficiency in excess of 93 %. The Istanbul CPU supports three HT-3 channels at 9.6 GB/s each.

The prototype consists of 3 racks with 180 blades interconnected with a full bisection QDR IB network. The peak capacity of each rack with the HE CPUs is 12.1 TFlop/s at a 30 kW rating, or 18.8 TFlop/s/m^2.

Of particular interest in regards to energy efficiency is AMD's various power management features, such as Dual Dynamic Power management allowing separate management of cores and memory controller, the CoolCore technology for turning off unused parts, PowerNow enabling control of core frequency and voltage levels, and Smart Fetch for managing cores and caches from a power perspective, and the APML tools.

Figure 14: Rack of prototype

2.1.10 *RapidMind*

Key Objective: Evaluation of RapidMind programming model.

Lead Partner: LRZ.

The "RapidMind Multi-Core Development Platform" [16] promises easy and portable access not only to multi-core chips from Intel and AMD but also to hardware accelerators like GPUs and Cell. The basic concept of the RapidMind (RM) language is called "*data-stream processing*"; a technology that describes a powerful way to express data parallelism (see Figure 15). RM adds special types and functions to C++; the programmer defines operations (functions) on streams (special arrays). Data dependencies and data workflows can be easily described and will automatically include all information necessary for an efficient parallelization. The compiler and the runtime environment have maximum information to decide how to auto-parallelize the code.

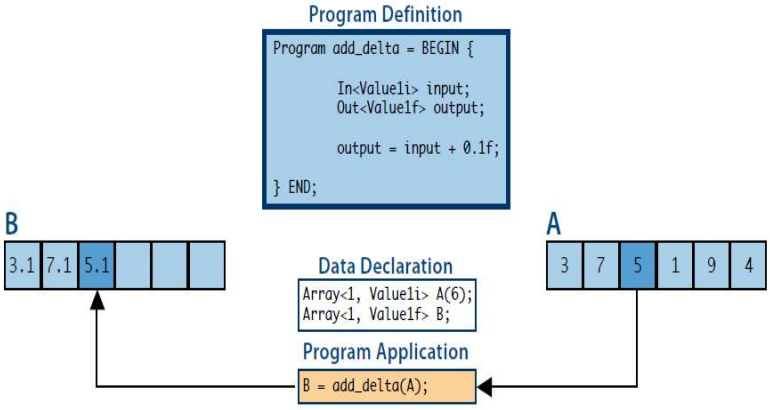

Figure 15: Data-stream processing in RapidMind

RM has been tested on nVIDIA Tesla boxes (RM "cuda" backend), nVIDIA Quadro FX cards (RM "glsl" backend), an IBM QS22-blades cluster (PowerXCell8i processors, RM "cell" backend) and several Nehalem-EPs (RM "x86" backend). A *comparative analysis of hardware accelerators* for HPC was one objective of this prototype. However, the results imply that the maturity of the available backends differs greatly and it is not advisable to draw conclusions on the suitability of different hardware accelerator for scientific codes from the performance results obtained with RM.

The second objective was to enable a *comparative analysis of new languages for HPC*. This could be well achieved; detailed findings are reported in Deliverable D6.6 [8]. We have ported all three mathematical kernels (mod2am, mod2as, mod2f) to RM, assessed productivity metrics like number of lines of code and development time and measured performance on various backends. Results show that RM is relatively easy to pick up and needs only few code lines to express the algorithm. Detailed findings of the experiences with the RM prototype have been reported in [41].

RapidMind Inc. has been acquired by Intel in autumn 2009 and will dissolve in Intel's new language Ct (C for throughput computing). Ct is currently only available for invited beta testers; a public beta should be available mid 2010. PRACE is in close contact with the Intel Ct[3] team.

The fact, that the RapidMind Multicore Development Platform is no longer available shows how fast developing the field of accelerators and accelerator languages is. When the EoIs for WP8 prototypes were due in mid 2008, many different accelerator architectures were available (Cell, ClearSpeed, GPUs, and already then rumours on Larrabee); two years later the market has been cleared out and seems to be dominated by nVIDIA. A phase of diversity is now followed by a period of standardization, with OpenCL being one of the precursors for transparent support of multiple architectures. Finding a standardized way to make the vast computational power of accelerators available to the HPC user community will facilitate the work for scientists, programmers and software companies at the same time.

In conclusion, the language concepts used in RapidMind and transferred to Ct are powerful ideas for expressing data-parallelism on many-core devices. Progress of data stream processing languages should be monitored closely. A performance and productivity assessment of Ct should be done once a first release will be available.

2.2 Research activities

2.2.1 *PGAS language compiler*

Key Objective: Evaluate the ease of use of PGAS programming model.

Lead Partner: ETHZ-CSCS.

Cray Compiler Environment (CCE) on the Cray parallel computing platforms offers integrated compilation capability for two Programmable Global Address Space (PGAS) languages, Co-Array Fortran (CAF) and Unified Parallel C (UPC). The PGAS model is expected to provide ease of programming as well as high performance for global-shared and distributed address space platforms. The CSCS team evaluated the CCE PGAS prototype compiler for functional correctness, conformance with language standards and usability using a subset of CAF and UPC benchmarks and applications. A performance release for these compilers is expected to be available on the next-generation Cray networks. According to Cray, this integrated compiler approach could result in significant performance improvements

[3] Publicly available information on Intel Ct can be found at http://software.intel.com/en-us/data-parallel/

over existing library and interpreter based approaches since the compiler and runtime infrastructure would have more information, freedom and opportunities to optimize data access and control patterns locally as well as remotely.

PGAS Language Compiler is installed on Cray XT systems at CSCS, Switzerland. The study aims at evaluating components of the software development and execution environments that are essential for developing and executing synthetic benchmarks and compact application kernels written using UPC and CAF language extensions. Although performance evaluation is not the focus of the current effort, the CSCS team investigates and highlights potential bottlenecks in the code translation and transformation processes that may not be simply addressed by a high performance communication interface, which is expected to be available on future Cray interconnects. In order to compare and contrast performance implications of different underlying hardware and system software, CSCS team evaluated CCE PGAS compiler on a vector system called X2. Unlike the Cray XT5 platform, which has a contemporary x86 processing node, Cray X2 system has proprietary vector processors with a uniform memory hierarchy within the node. Access to the X2 nodes are provided by the Edinburgh Parallel Computing Center (EPCC).

The PGAS programming model offers programming flexibility of a globally-shared memory programming model while introduces a concept of data locality that is similar to a distributed-memory model. In other words, globally shared data structures and functions (collectives) that access them, as well as data structures representing the locality and distribution of data across multiple processing elements are two important parts of the programming paradigm. Part of the global shared data structure is available locally and a compiler or interpreter could make use of this locality of reference to provide high performance. As a result, a user or code developer typically does not need to manually optimize and fine-tune the application to reduce accesses to remote data structures, while the UPC and CAF constructs, such as upc_forall loops, will aid in the automatic optimization. Unlike message-passing models, these remote accesses are performed without explicit messaging i.e. using one-sided communication primitives.

Co-Array Fortran (CAF), which is also a part of the FORTRAN 2008 standard proposal, extends the Fortran language syntax with a construct called *co-arrays*, which are essentially data structures that are shared between different *images* of a program. Accesses to these co-arrays then result in remote memory accesses that are similar to remote memory *put* and *get* operations. In addition to a simple syntax extension to parallel programming, on some networks, these operations could be performed far more efficiently than exchanging data using the MPI message passing operations. Moreover, as the CAF compilers offer interoperability with the MPI programs, an existing parallel MPI code need not to be re-written completely for exploiting benefits of the CAF approach on systems optimized for PGAS languages. For example, below is a representation of arrays using the CAF syntax illustrating a regular array (a)—a private copy for each CAF image—while a CAF array (a_caf) is distributed among the given number of images:

```
DOUBLE PRECISION a(ndim)
DOUBLE PRECISION a_caf(ndim)[*]
```

Similarly, UPC is an extension to the C language offering benefits of the PGAS model to programs written primarily in C. In UPC, program instances are called *threads* and data is divided up in shared and private spaces. There are language qualifiers to describe whether data is shared and how arrays could be distributed among available threads. The number of threads can be specified at both compile and runtime. Although the UPC language specifications do not address the issue of interoperability with other programming paradigms, for example MPI, some compiler instances have addressed this issue. For example, a simple array

(a) has a private copy for each thread, while array (a_upc) is spread across the number of available threads.

```
double a[ndim];
shared double a_upc[ndim];
```

Unlike globally shared programming model instances such as OpenMP, the PGAS model and its instances allow code developers to articulate locality of shared data types and offer mechanisms to control data mapping and distribution. Likewise, these restrictions permit the underlying compiler and runtime systems to distribute and schedule remote memory accesses in an optimal manner without maintaining a global, uniform access view of memory on a distributed memory system.

2.2.2 *Research on Power Efficiency*

Key Objective: Evaluate the power consumption of the internal components like CPU, HDD, RAM, etc.

Lead Partners: PSNC, EPSRC-STFC.

PSNC

The research activity mainly investigates power efficiency of different server implementations to identify:

- Power efficiency of different hardware solutions;
- Power consumption profile of HPC servers.

The measurement was performed using the following equipment:

- A high resolution power meter Lutron DW-6090;
- Digital-multimeters Picotest;
- Multi-channel data logger Yokogawa XL-100;
- Additional equipment such as current shunt resistors.

This equipment connected in the configuration presented in Figure 16 allowed for detailed measurement of power on both the AC and DC side of power supply not only at given point of time but also for creating a profile of power consumption while running tests stressing different components of servers.

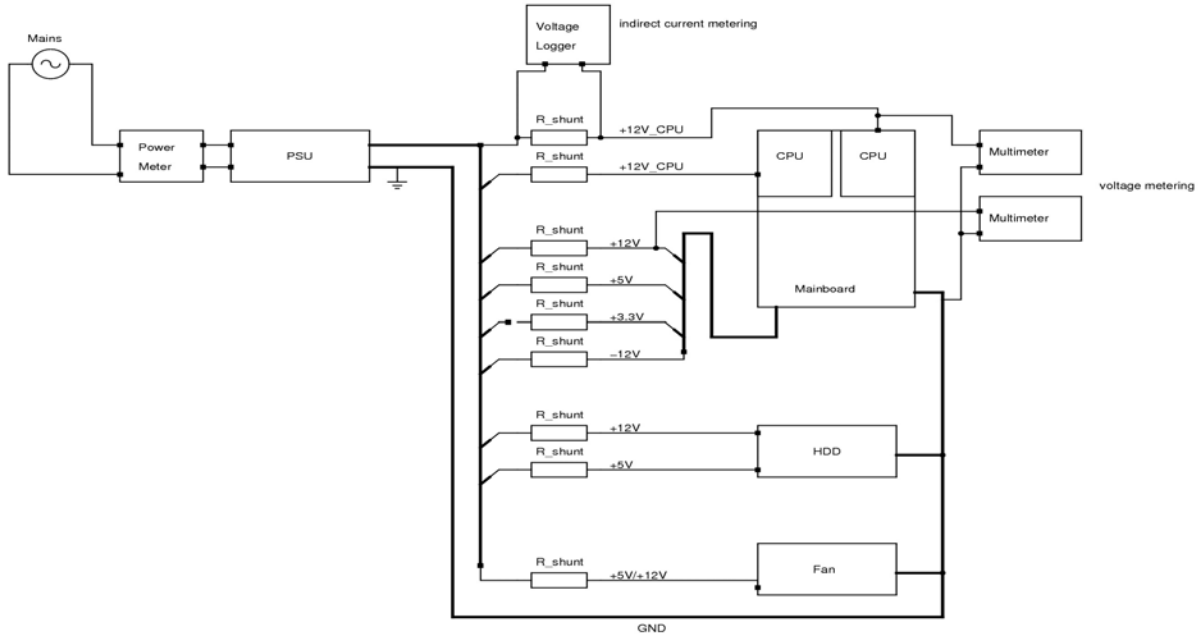

Figure 16: Connection diagram

The work focused mainly on the most popular x86 servers but also included other architectures such as Itanium 2 based server and SiCortex MIPS server.

STFC

GPU Prototype Cluster

The GPU cluster comprises eight Supermicro 825TQ-R700 LPB servers and eight nVIDIA Tesla S1070 servers thus providing 1:1 host to accelerator ratio. The Tesla servers feature four C1060 cards clocked at 1.3 GHz and 4 GB of RAM each. Further details are as follows

- Motherboard: SuperMicro X8Dai;
- Processors: 2× Intel Xeon E5540 @ 2.53 GHz (max TDP 80 W), SMT disabled;
- Memory: 6× 4GB DDR3 DIMMs;
- Network: Infiniband Voltaire HCA410-4EX, on-board dual gigabit Ethernet, IPMI;
- SATAII hard drive;
- Dual-power PSU;
- OS and software: Scientific Linux 5.3, gcc 4.1, icc 11.0, pgcc 9.0, CUDA 2.2.

2.2.3 *Parallel GPU*

Key Objective: Evaluate GPGPU computing in a cluster system.

Lead Partner: CSC.

The research activity by CSC investigates the performance of GPGPUs in cluster environments:

- Porting of WP8 EuroBen kernel subset (mod2am, mod2as, mod2f) to OpenCL and CUDA parallelized with MPI;
- Performance analysis of a real world GPGPU application (GPU-HMMER).

The initial testing of GPU-HMMER was performed on the single prototype node at CSC called "naru" which has 2 NVIDIA GTX 280 GPUs, which was already set up for use in other projects. Initial testing involved, comparing the performance of GPU-HMMER to the regular, multithreaded version of HMMER running on a HP DL785 server with 8 AMD Opteron QC 8360 CPUs (total of 32 cores).

The CUDA+MPI and OpenCL development was initially performed on a CSC in-house GPGPU cluster called "Flux", which has 13 nodes each with an nVIDIA Tesla C1060. Final benchmark runs were performed on the GENCI-CEA Hybrid technology demonstrator, described in Subsection 2.1.6.

The functionality and performance of the languages was assessed. In addition to basic evaluation, the goal is to answer some more specific questions:

- How simple is it to parallelize an existing CUDA program, and will a trivial parallelization of the code provide performance gains?
- How much work is it to port an existing CUDA program to OpenCL? This is an important question as many HPC sites are evaluating the risk in investing heavily in CUDA now (and potentially having to port the codes to OpenCL in the future);
- Can a GPGPU and an InfiniBand card coexist in the same system? Both devices can lock system memory for direct memory access which may cause unexpected interactions and should be studied.

2.2.4 *Performance Predictions*

Key Objective: Analysing and modelling the application and understanding its behaviour.

Key Partners: BSC.

During the years, BSC has been developing performance analysis tools and modelling techniques to measure and understand the behaviour of parallel programs. Three major components of this environment are: MPItrace, instrumentation package to instrument MPI applications and generate traces that have been ported during the PRACE project to some of the prototypes (in particular BG/P and ICE); Paraver [11], a trace visualization and manipulation tool; and Dimemas [12], a trace replay and interconnect simulator tool.

In Section 3.8, we present how these tools are used in prediction studies to describe the impact of different components of a hypothetical future machine on the performance of one of the benchmark applications used in WP5. The actual example we use is the GADGET benchmark. Starting from traces obtained on MareNostrum (BSC), we will predict the performance for one of the WP5 prototypes and one of the WP-8 prototypes. We will study the potential impact of using accelerators and will show a general analysis pointing to areas where some restructuring of the application would have potential impact on its scalability.

In the following paragraphs we briefly summarize two major components that will be used for such evaluation: the Dimemas simulator and a simple analysis model identifying the most relevant performance factors determining the parallelization efficiency of one application.

Tools

Paraver is a trace based visualization tool with additional capabilities to manipulate traces. The general methodology in our study is as follows. We first obtain a trace for the benchmark run using the MPItrace package. This is done both on MareNostrum and the ICE and BG/P prototypes. As the traces are typically large (a few GB in this case) we use Paraver to filter them, generating a trace with a subset of the original information (typically only the one cor-

responding to a sufficiently large computation burst) but representing the whole run. Visualising this trace it is possible to identify the periodic structure of the application and then cut a chop of the original trace, keeping all the detailed raw data for exactly one iteration of the program. This trace can be analysed in detail looking at a huge number of views and metrics, as well as converted to the Dimemas format for further predictive studies.

Dimemas is a simulator that based on the description of the computation and communication demands can rebuild the time behaviour the application would have shown on a hypothetical target machine. The computation is described in terms of its duration on the machine where the trace was obtained. The communication is described in terms of MPI call, partners and message sizes. The target architecture is modelled with parameters that describe the processor and interconnect. The target processor performance is compared to that of the tracing machine (CPUratio). Latency and bandwidth are the two major components of the communication model, but we also consider factors such as the number of network adapters in a node (which can of course be an SMP) or the contention in the network. These non linear effects as well as dependencies (production consumption order) or the end to end synchronization semantics of the communication (buffered/rendezvous, short/long message protocols, etc.) are taken into account by the simulator to estimate the execution time on the target machine. Dimemas also generates a Paraver trace of how the behaviour should be under such a machine model. An extremely useful practice is to compare with Paraver, the real and predicted traces. Even if sometimes differences show up, this helps identify issues that sometimes are due to the simplification of the model and sometimes due to problems in the implementation of the communication subsystem in the real machine.

Additionally, we have utilities that can cluster the different computation regions in the program according to their similarity. We have used this to identify the regions that one would first port to an accelerator and this drives Dimemas to apply to these regions a high acceleration factor while the rest of the code is simulated with the reference node performance.

Analysis

We have proposed a very general model to describe the scalability of an application in terms of general factors or concepts understandable without requiring understanding or looking at details of the application. The efficiency model is multiplicative with factors quantified with a value from 0 to 1. A high value of the factor means that it is not limiting the efficiency (or scalability) of the application. A low value implies that this factor is responsible for a big degradation in the performance of the application.

The very first model considers four factors: load balance, communication efficiency, raw core efficiency and computational complexity. The total efficiency of a single run could be described as the product of the first three factors

$$\eta = LB \times CommEff \times IPC$$

where Load balance (LB) and communication efficiency (CommEff) are computed from the Gant diagram in Figure 17. The blue area represents the total computation time of each process (one row per process, ranked from highest to lowest computational load), and the red plus yellow area represent the time in MPI by each process. The red area is the time all processes are in MPI out of the total elapsed time T.

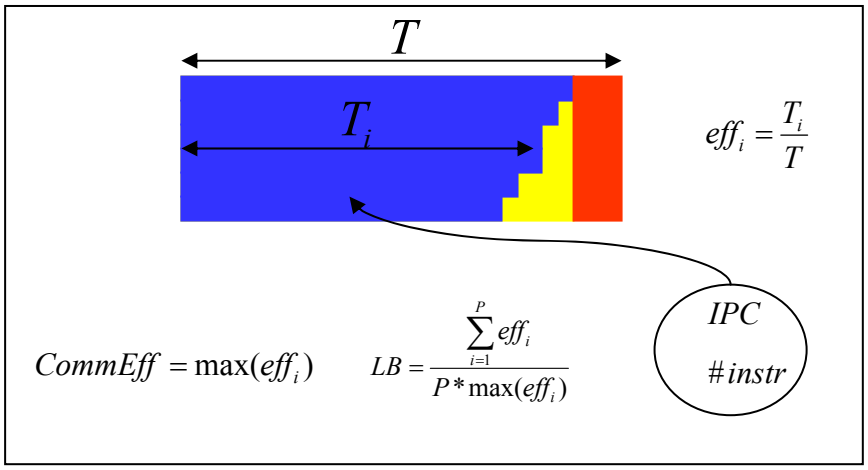

Figure 17: Gant diagram and efficiency model

The IPC (Instruction per cycle) factor would represent the raw core efficiency during the useful computation (blue in Figure 17) and should be normalized to the peak value of the processor. If we are interested only in the parallel efficiency, only the first two terms should be used. If we are interested in using the model to explain the scalability between different runs an additional term should be added to the product representing the computational complexity (in terms of total number of useful instructions) of the algorithm.

The model is nevertheless based on the aggregated time in each state (useful computation and MPI) and is not able to differentiate whether the possible degradation within the CommEff term is actually due to transfer delays or to synchronization. It is in this point where a Dimemas based simulation can help differentiate these two factors.

$$\eta = LB \times microLB \times Transfer \times IPC$$

The way to separate both components is presented in Figure 18. The idea is that a Dimemas simulation with an ideal interconnect (zero overhead in MPI calls and infinite bandwidth) would still have to preserve the synchronizations required by the application and might delay some processes (yellow in Figure 18) but still compute the ideal time (T_{ideal}) which is a lower limit for the application execution time. The rest up to the original application time is assigned to the actual data transfer component of MPI calls (Transfer term in the formula). The other term may actually represent serialisation (i.e. pipeline) or situations where individual intervals between synchronizing communication phases (typically collectives) are actually imbalanced but the process that arrives late is different every phase and when aggregating the total computation time ends up being similar across processes. Among the two indistinguishable causes this is the most frequent one in real applications and thus we call this term the micro load balance factor.

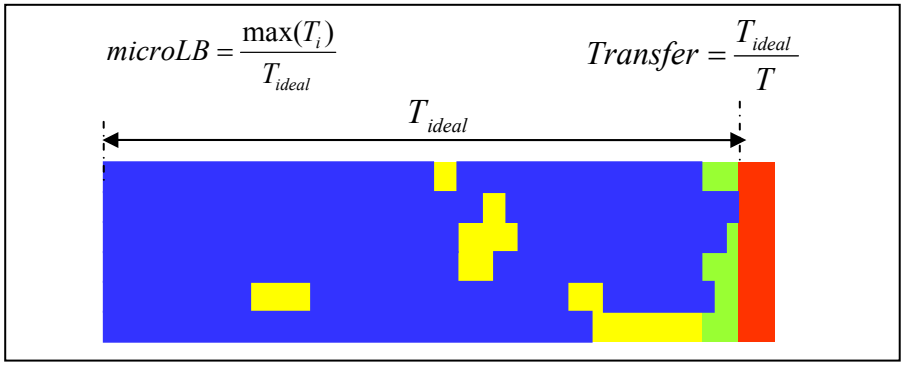

Figure 18: Model including microscopic load balance and serialization

3 WP8 evaluation results

This chapter describes the evaluation results obtained on the twelve future technology prototypes that were selected by PRACE and have been approved by the European Commission. The advanced nature of the WP8 prototypes makes a strongly structured approach for the assessment, as it was taken by WP5/7 impossible. However two components that are addressed by many of the prototypes are the evaluation of programming models and accelerator based nodes. Subsection 3.1.1 describes the rationale to use a subset of the EuroBen benchmark kernels to obtain comparable performance results. Porting of the selected EuroBen kernels to 12 different languages and/or programming models, namely MPI+OpenMP, UPC, CAF, Chapel, X10, CellSs, CUDA, CUDA+MPI, OpenCL, RapidMind, Cn and CAPS/HMPP, was a joint effort of WP8 and WP6. Network interconnect is another component evaluated in several prototypes. The *IMB benchmarks* are used in some of them although specific implementations are developed for some cases. A general overview about the benchmarks and applications used by WP8 is given in the Annex Chapter 6.

In the following, the assessment reports are arranged under several sections such as “GPGPU Programming Languages and Accelerated Compilers”, “Hybrid Programming Models”, “Intra-node Bandwidth”, “Inter-node Communication Network”, “PGAS Languages”, “Novel I/O” and “Energy Efficiency” as far as their evaluation objective matches with these categories. Exceptions from this rule are Section 3.8 which describes the experiment to identify issues that may be relevant when scaling or porting applications to future architectures and Section 3.9 which summarizes the evaluation results and gives some concluding remarks.

3.1 Performance experiments on prototype hardware

3.1.1 *Reference performance*

Figure 19 shows the processor family share of the November 2009 Top500 list. 79 % of the most powerful systems in the world are based on Intel EM64T processor technology. In order to be able to compare EuroBen results on WP8 prototype hardware and language platforms on equal footing, the performance values obtained on a standard dual-socket Intel Nehalem-EP 2.53 GHz processor platform (E5540) was used as reference performance baseline. This processor has theoretical peak performance of 10.12 GFlop/s per core.

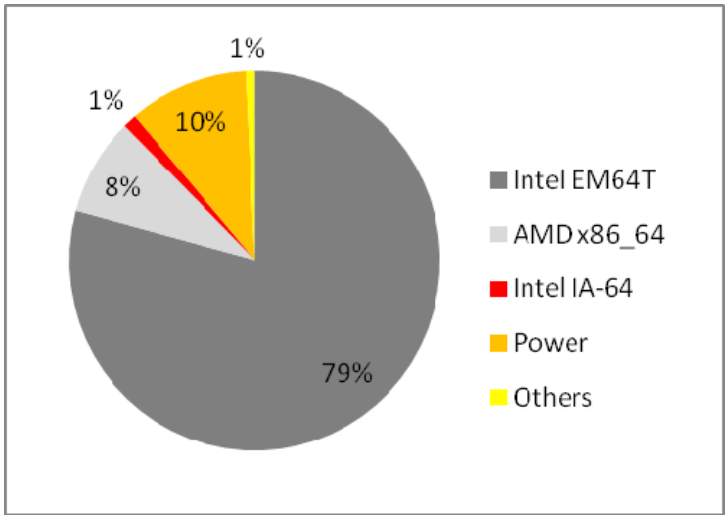

Figure 19: Processor family share in the November 2009 Top500 list

Since Green-IT and the use of GPGPUs as compute accelerators were the most notable facts during both supercomputing conferences in the year 2009, this chapter will also compare the measured performance values of different accelerators in terms of performance per Watt with the Nehalem-EP reference platform.

In the text below the rationale for the choice of the kernels from EuroBen [13] together with the performance found for each of them is given.

Nehalem-EP single core performance results

Dense matrix-matrix multiplication (mod2am): This kernel was chosen because of the very favourable ratio between floating-point operations and memory accesses. One may expect that it reaches near peak performance on any processor, standard or of accelerator type. Along with the C + MKL code, also a standard Fortran implementation was run because Fortran is also heavily used in HPC program development.

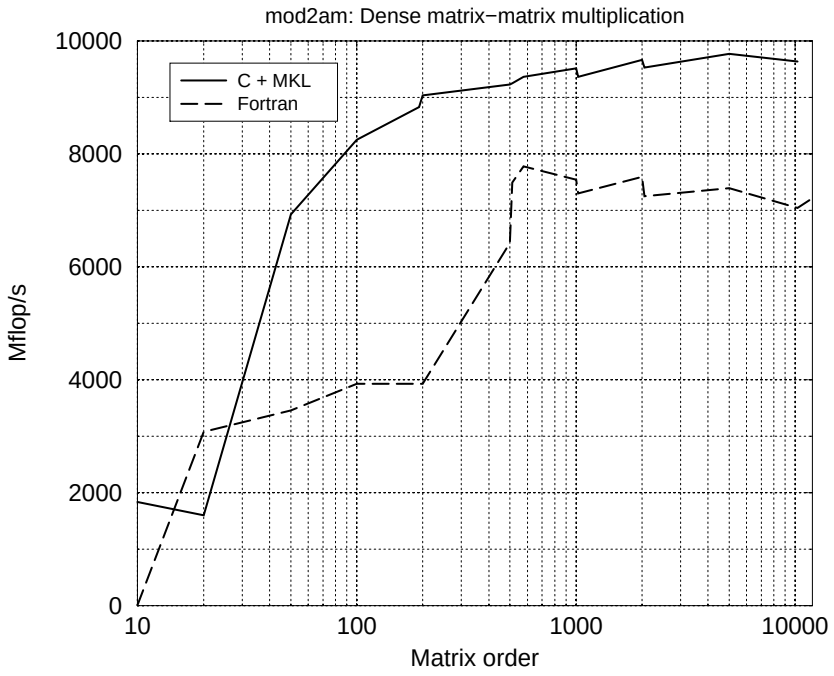

Figure 20: Reference performance of dense matrix-matrix multiplication

The kernel can attain more than 9 GFlop/s for matrix orders ≥ 2000.

Sparse matrix-vector multiplication (mod2as): Sparse matrix-vector multiplication is an essential ingredient of all iterative sparse linear solvers and as such of a large fraction of scientific modelling applications. The ratio of floating-point operations and memory accesses is very poor. In addition the vector elements are accessed indirectly which makes it impossible to cache them. On may expect this kernel to perform poorly on all processors, although probably better on general CPUs than on most accelerator processing elements.

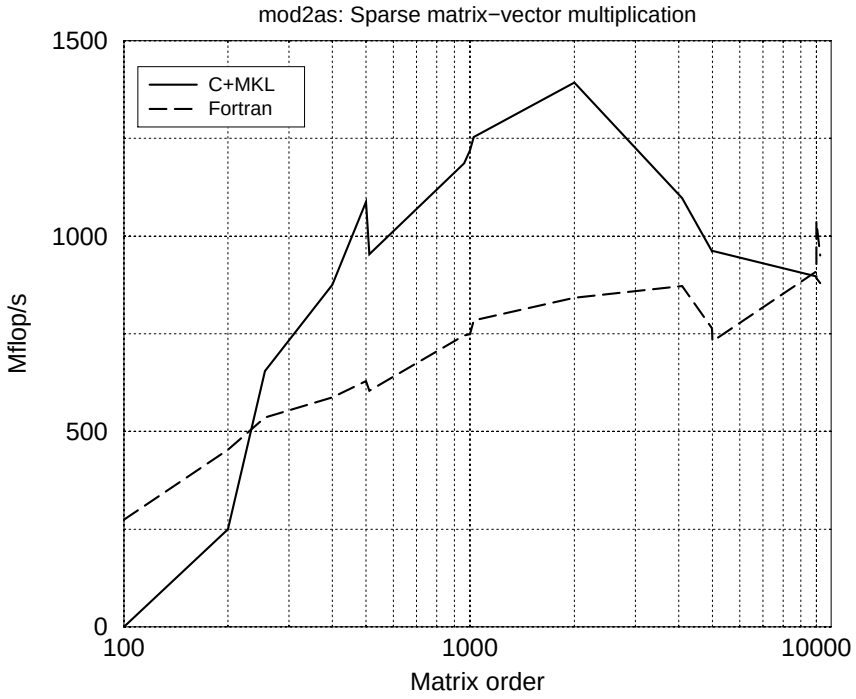

Figure 21: Reference performance of sparse matrix-vector multiplication

As can be seen the performance for both the C + MKL version and the Fortran version is only a small percentage of the theoretical peak performance. For the smallest and largest matrix orders the Fortran implementation is somewhat faster than the library-based implementation.

1-D complex-to-complex FFT (mod2f): FFTs are a part of many different types of applications, ranging from particle codes to weather simulations. The ratio of floating-point operations to memory accesses it reasonable, but the memory access patterns are intricate and therefore one may not expect a large fraction of the peak performance. For accelerator processing elements a similar behaviour may be expected.

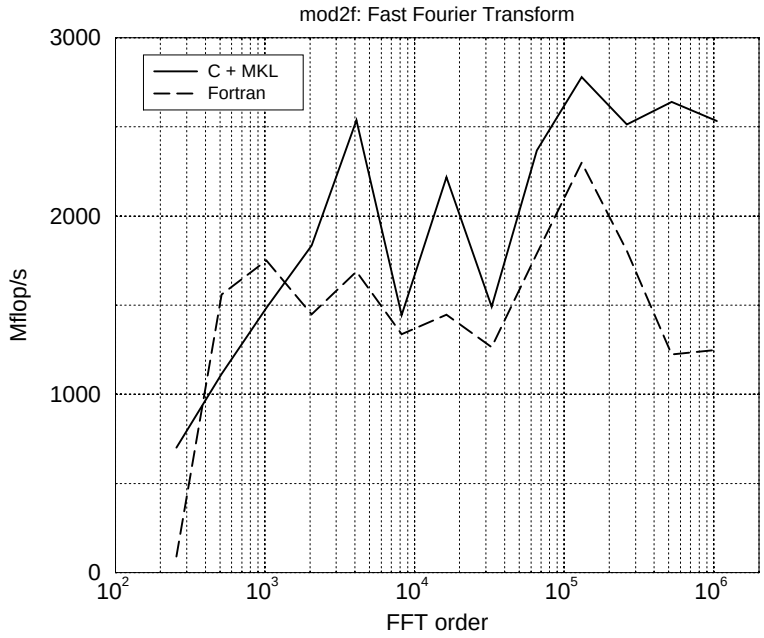

Figure 22: Reference performance of 1-D complex-to-complex FFT

Shift-register random number generator (mod2h): This kernel was chosen because of its frequent occurrence in Monte Carlo-type simulations. In contrast to the other kernels only a few

floating-point operations are involved. The majority of operations are of integer and logic type. The speed is therefore expressed in MOp/s instead of MFlop/s.

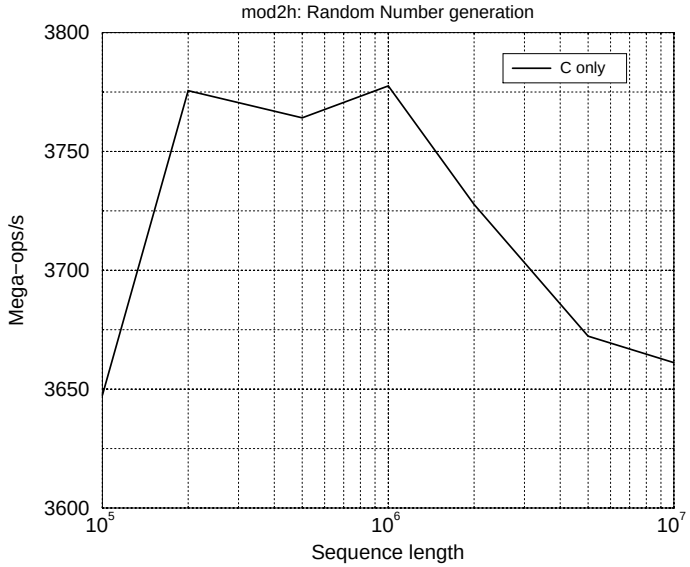

Figure 23: Reference performance of the random number generator

The MKL library does not include random number generators so only the performance of the C code is shown in Figure 23.

Dual socket Nehalem-EP performance results

Figure 24 and Figure 25 shows the reference performance of EuroBen kernel measured on a dual socket 2.53 GHz Intel Nehalem-EP node when running 1, 2, 4 and 8 thread or MPI program instances. In the following the dual-socket Nehalem-EP EuroBen kernel reference performance baselines are defined as the highest single node performance obtainable e.g.:

- 8 thread MKL results for mod2am;
- 4 thread MKL results for mod2as;
- 1 thread MKL results for mod2f;
- 8 instances results for mod2h.

The performance of mod2am on 8 cores is indeed about 8 times that of a single core. This comes as no surprise as the kernel is entirely computation-limited. The performance of mod2as is fairly close to 3 times higher than the single-core ones. This is because the computational intensity of this kernel is mostly memory bound. Note the steep rise in performance from matrix order 1000 to 2000. This is due to a better fit in both caches and functional units.

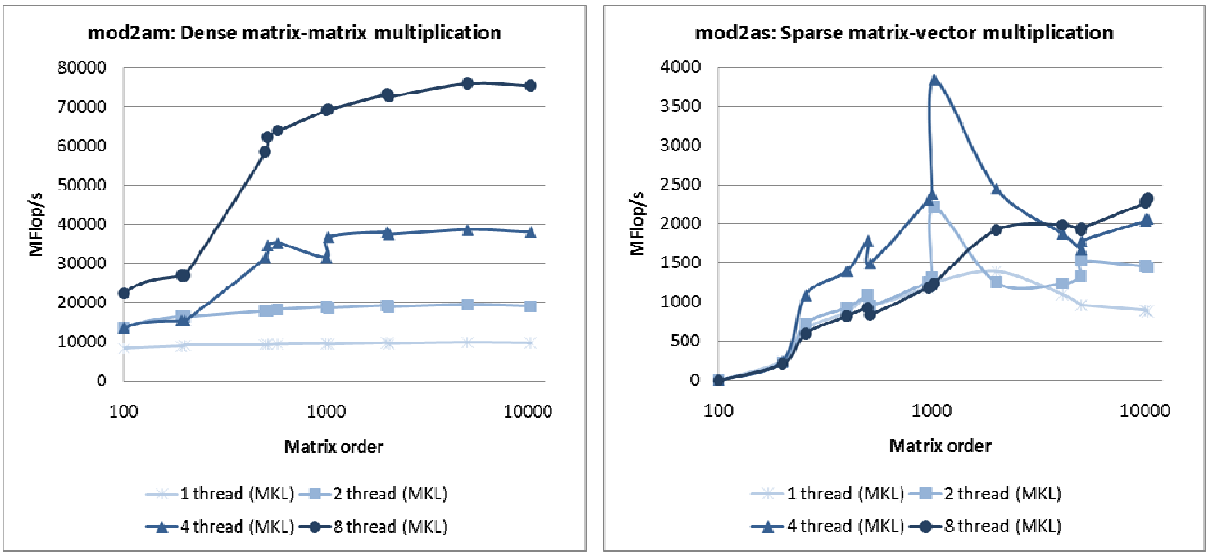

Figure 24: Reference performance of dense matrix-matrix and sparse matrix-vector multiplication

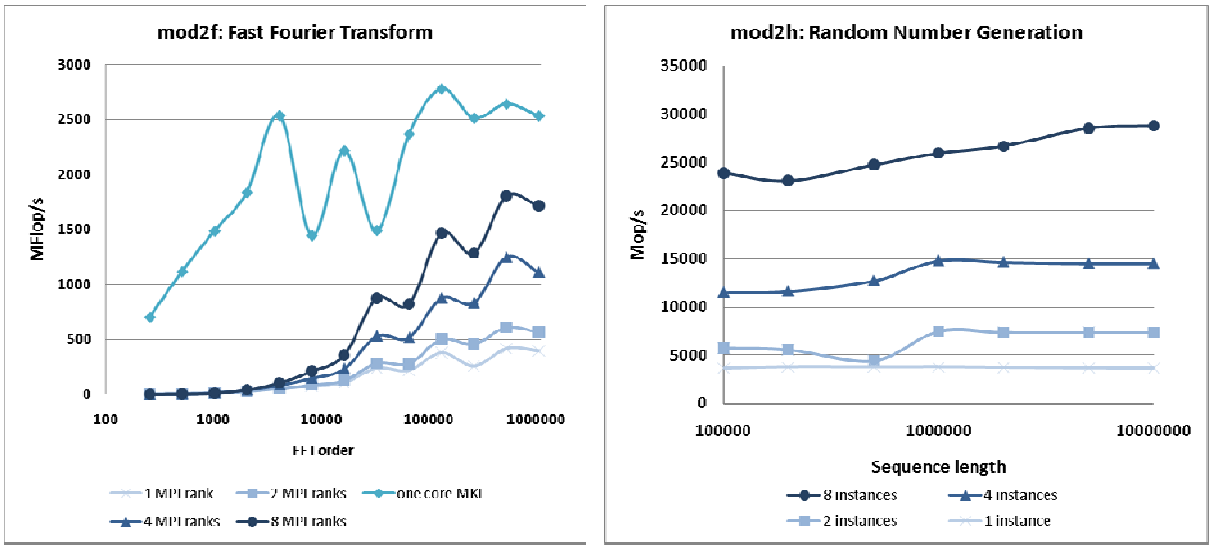

Figure 25: Reference performance of FFT and random number generation

For the FFT-kernel, mod2f, no readily equivalent full-node performance can be given as it cannot be run on multiple cores without serious performance implications (see FFT results in Figure 25). Hence the single core Nehalem-EP MKL performance will be used as reference. In case of mod2h no MPI nor thread or OpenMP parallel version does exist. Therefore the accumulated performance of 8 concurrent serial mod2h instances will be used as reference performance.

The total power consumption of one dual socket Nehalem-EP system running 8 threads of mod2am is 303 W. Hence the power efficiency of a 2.53 GHz dual socket Nehalem-EP node is 251 MFlop/s per Watt.

3.1.2 *Numerical issues*

Introduction

In this subsection, we study the numerical behaviour of heterogeneous systems such as CPU with GPU for some orthogonalisation processes. We focus on the influence of the different floating arithmetic handling methods of these accelerators with Gram-Schmidt orthogonalisation using single (SP) and double precision (DP). To do so, we first have a look at the resulting orthogonality of the computed basis for dense matrices, then for sparse matrices.

Orthogonalisation Process

Several variants of the orthogonalisation process exist: Householder reflections, Givens rotations and the Gram-Schmidt process are the most common ones. The Gram-Schmidt process is available in different versions: classical (CGS), modified (MGS) and classical with re-orthogonalisation (CGSr) for instance. The classical version is the simplest and most easily parallelisable: it is faster than the others, as it can take advantage of BLAS 2 operations instead of BLAS 1 in the modified algorithm. But the main drawback of CGS is the possibly high loss of orthogonality within the computed basis due to round-off errors. The MGS algorithm tries to correct this, and in fact manages to provide a more accurate version of the Gram-Schmidt process. Both are mathematically equivalent, but the MGS exhibits less parallelism. The CGSr process provides an even more accurate version, by re-orthogonalising each computed vector. In practice it is not necessary to re-orthogonalise at each iteration, and so the cost of this algorithm using selective re-orthogonalisation is close to CGS while being more accurate.

Tested hardware

The hardware IEEE floating-point norm is handled slightly differently on GPGPUs than on fully IEEE-754 architectures like most CPUs. Here are some examples of these differences for SP only, taken from the programming manual:

- Addition and multiplication are often combined into a single multiply-add instruction (FMAD), which truncates the intermediate result of the multiplication;
- Division is implemented via the reciprocal in a non-standard-compliant way:
- Square root is implemented via the reciprocal square root in a non- standard-compliant way;
- For addition and multiplication, only round-to-nearest-even and round-towards-zero are supported via static rounding modes; directed rounding towards +/- infinity is not supported.

There are more details about IEEE support of nVIDIA GPUs in the programming manual. The GPU hardware used for the experiments in this paper is an nVIDIA Tesla S1070 scientific GPU embedding four C1060 GPU cards. Each of these cards holds 4 GB of 512 bits GDDR3 memory @ 800 MHz, providing a maximum memory bandwidth of 102 GB/s per card.

Implementation

For each different version of the Gram-Schmidt process, the same basic BLAS operations are applied. Consequently, we may use optimized BLAS operations to take advantage of the different hardware in an optimized and portable manner.

The implementation of the dense orthogonalisation process on classical CPUs follows the standard algorithm and uses ATLAS subroutines.

Concerning the sparse case, we follow the implementation provided in the sparse matrix vector multiply utility from nVIDIA, mixing this product with ATLAS routines to comply with the described CGS algorithms.

GPU Implementation for the dense case is also close to the original algorithms and uses mainly CUBLAS, except that we have to take into account the memory operations between host and device memory: allocation and transfers.

For sparse matrices, we use the optimized sparse matrix vector multiply from nVIDIA, which can handle different matrix formats. Provided this optimized matrix-vector product, we use CUBLAS routines to exploit the GPU power in a portable way.

Experimentation methodology

The precision of the GS process for one subspace size is calculated using the greatest absolute dot product of every dot product of the orthogonal basis:

$$\max | (v_i, v_j) |, \text{ with } i \neq j \text{ and } v_i, v_j \in V \text{ basis.}$$

Precisely, this would be the "worst" orthogonality between two vectors of the basis. A value of 0 corresponds to no errors, and a value of 1 indicates a completely wrong result. Also, the accumulator of the dot products uses the same precision as the rest of the calculations: if the precision is SP, then the accumulator is SP too. Same applies for DP. This way we do not add another parameter to the calculations.

Dense matrices

Figure 26 shows the behaviour of each system on Hilbert matrix taken from Matrix Market [40], during the dense CGS Arnoldi projection. CGS is known to be very sensitive to machine precision, which is usually IEEE-754 compliant.

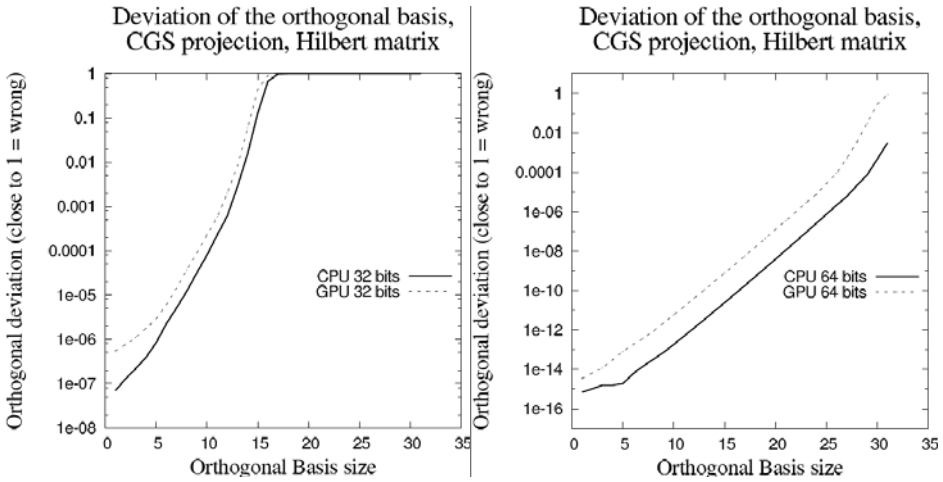

Figure 26: Precision of the CGS Arnoldi Projection of a 10240 square Hilbert matrix

For the dense case, the GPU's different IEEE handling introduces a loss of one digit compared with the CPU result for the orthogonality of the computed basis in SP and DP.

Sparse matrices

The orthogonalisation process is used in the QR method as well as in the Arnoldi projection, and so we chose to take as test matrices two matrices from the University of Florida sparse matrix collection, whose original problem is the computation of the eigenvalues. Figure 27 shows how the GPU behaves compared with a CPU on the Andrews sparse matrices using CGSr.

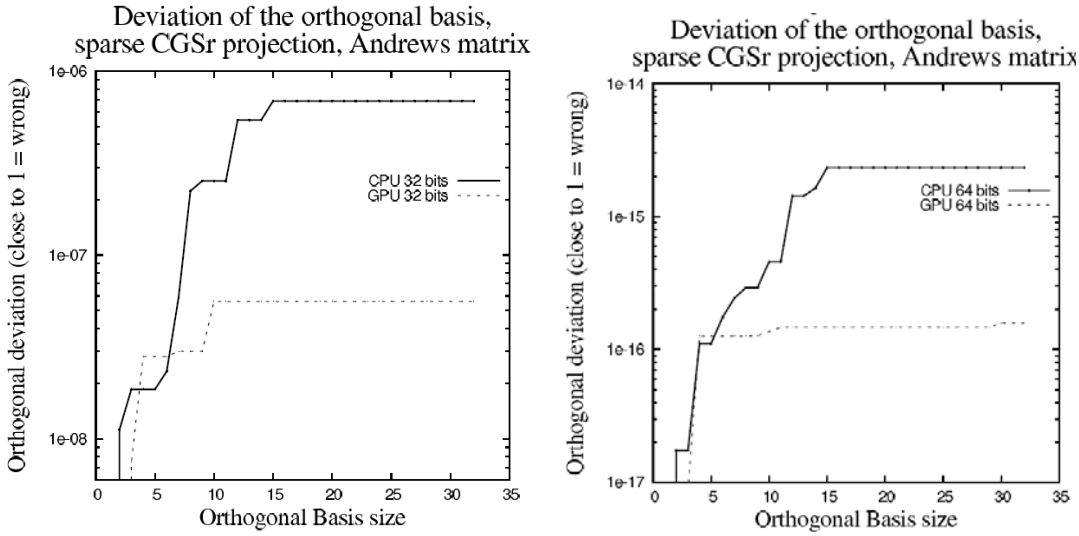

Figure 27: Precision of the sparse CGSr process on sparse Andrews matrix

For the sparse matrices, the GPU results are less regular, providing either a better or worse orthogonal basis, which is in fact close to the one of the CPU case. This is due to a different execution order of the floating-point operations as well as very small numbers in the V basis that tend to be truncated during computations on the GPU.

3.1.3 *Accelerated Programming Languages and Compilers*

Novel programming models such as HMPP, OpenCL, RapidMind or StarSs are able to generate code for general purpose CPUs as well as GPGPUs. WP8 used the two EuroBen kernels mod2am and mod2as to compare the performance of the code generated from these different languages for nVIDIA C1060 GPGPUs.

The following subsections describe the performance results WP8 was able to obtain for the two EuroBen kernels using these novel programming languages. Remarks concerning their ease of use and their maturity are also given when appropriate.

nVIDIA CUDA

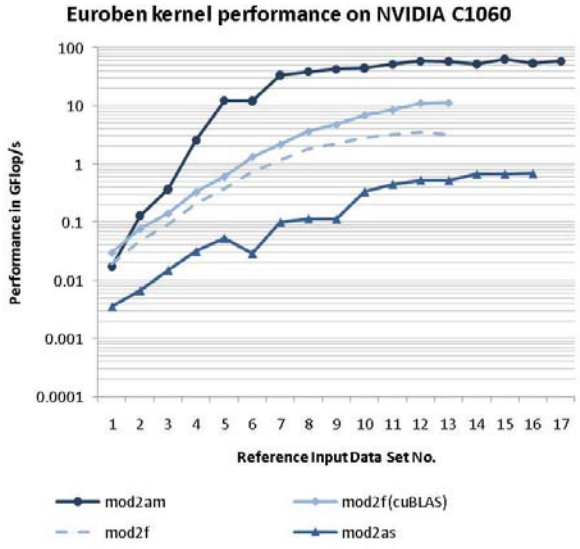

Figure 28: Comparison of the CUDA kernels on nVIDIA c1060 GPUs

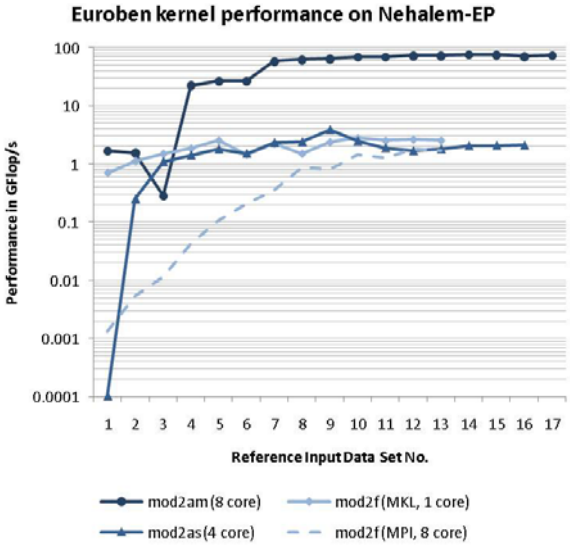

Figure 29: Comparison of the MKL kernels on Nehalem-EP

The three EuroBen kernels mod2am, mod2as and mod2f have been ported to CUDA [14][15] and, for the sake of comparison, to the Intel Math Kernel Library (MKL). The experiences gathered during the ports and the results have been reported in detail in D8.3.1 [2] and D6.6 [8]. Performance measurement has been done on the WP8 prototype Hybrid Technology Demonstrator. Porting to CUDA was not trivial, especially in the case of mod2f: the cuFFT library supported only SP at that time, and a full hand-coded port was necessary to achieve reasonable performance (~ 3000 lines of code compared to 75 lines for the version with the cuFFT call).

nVIDIA recently released version 2.3 of CUDA. Among various changes, the cuFFT-library does now support DP. Figure 28 shows a comparison between all kernels ported to CUDA, including the hand-coded CUDA port and the cuFFT DP version. This performance is compared against the best performing MKL ports of the three kernels (Figure 29). Note that MKL does not support multi-threading for FFT, we have additionally benchmarked the mod2f C+MPI version. The DP peak performance of 8 Nehalem-EP cores (running at 2.53 GHz) and one C1060 GPUs is comparable (80 GFlop/s vs. 78 GFlop/s).

As described in D8.3.1, the hand coded port of the one-dimensional Radix-4 FFT offered many challenges; deep knowledge of the hard- and software was required to achieve significant performance. The results of the new version using cuFFT indicate the true potential of the nVIDIA Tesla cards. However, this leads to the question how an average experienced programmer can exploit the potential of GPUs if his code does not allow using library calls.

PGI Fortran and C Accelerator Compiler

CUDA certainly played a very important role in the success of nVIDIA GPUs in HPC. It provided a relatively easy way to program the hardware without the need of going to a very low level thus remaining portable across different GPU models. However despite this success there is certainly a demand for higher level abstraction and portability not only across nVIDIA GPUs but across GPUs supplied by other vendors. Such a possibility is offered by HMPP (CAPS entreprise) and more recently by PGI accelerator compilers (Portland Group Inc). There are many similarities between the two approaches: both support C and Fortran, both work through #pragma constructs similar to OpenMP, both currently support only nVIDIA hardware (both generate CUDA internally although HMPP generator for OpenCL is also available now). In fact, it is conceivable that eventually they will provide a basis for a common standard. However at this stage it is premature to forecast anything since it is not clear how efficient and successful these approaches are going to be. There used to be some semantic differences in their approach though they are converging towards the same models (yet not the same syntax). For example, HMPP required an accelerated kernel to be defined in a function called codelet whereas in the PGI model the accelerator pragma simply precedes for/do loops. The last version of HMPP (release 2.2.0) offers regions too whereas PGI added more pragmas to provide more accurate descriptions of data transfers. Nevertheless, the underlying hardware dictates many similarities, for example, in the respect to the data management (e.g., data copying to and from the GPU) and both approaches rapidly evolve in the same direction as users require new features.

PGI accelerator programming model was first introduced in early 2009 as a beta program and then formally released in version 9.0 of PGI compiler suite. Its advantages are:

- Minimal changes to the language;
- Possibility of incremental program migration;
- PGI unified binary model so that a single binary can be executed on the hardware with or without GPU;
- Performance feedback and standard x86 tool chain.

PGI accelerator directives can be as easy as adding a single line:

```
#pragma acc region for parallel
for( i = 0 ; i < m; i++ ){
  for( k = 0; k < n; k++ ) {
    for( j = 0; j < l; j++ ){
      c[i][k] = c[i][k] + a[i][j]*b[j][k];
    }
  }
}
```

The accelerator programming model can be mixed with MPI and OpenMP. In this scenario, runtime functions need to be used which allow discovery and allocation of accelerator devices. Not all data structures are currently supported and the compiler may require some help in figuring out data dependencies (e.g., loop not parallelizable due to data dependency).

We used the PGI accelerator model with the EuroBen benchmarks and successfully compiled and ran mod2am (matrix-matrix multiplication) and mod2as (sparse matrix-vector multiplication). The results are summarized in the two figures below. The speedup of the PGI GPU accelerated version over the PGI Nehalem version on mod2am was up to 6 whereas no acceleration was observed on mod2as. The best percentage of theoretical peak on mod2am was 17 % (single core) on Nehalem and 11 % on C1060.

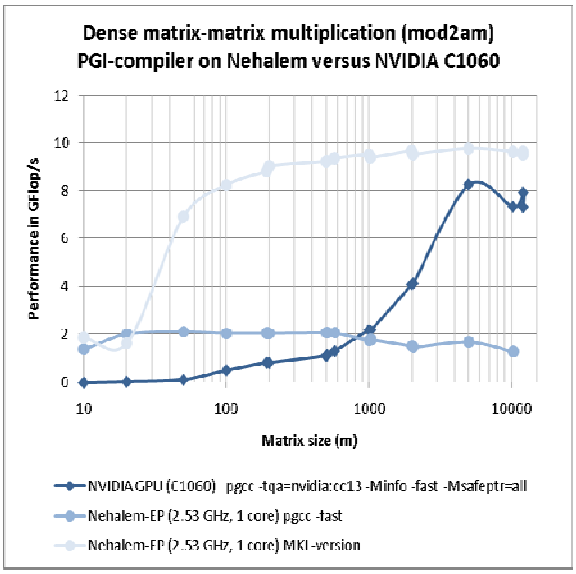

Figure 30: mod2am on a single core Nehalem vs. nVIDIA C1060

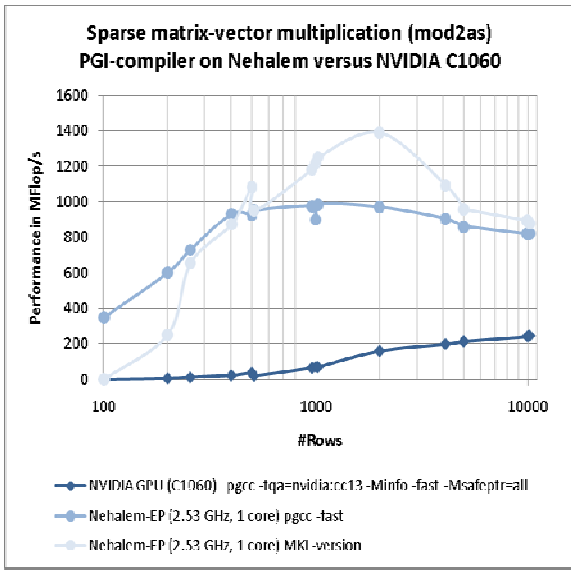

Figure 31: mod2as on a single core Nehalem vs. nVIDIA C1060

HMPP Workbench for Fortran and C

The CAPS HMPP workbench was first released in 2007, opening the way for a new approach to GPGPU programming. Rather than being a general purpose compiler, HMPP takes annotated programs (in the form of comments for Fortran or #pragma in C) and produces another source code according to the desired target. HMPP was the first tool to introduce fallback mechanisms to the CPU version should the GPU code fail. The initial approach taken by HMPP designers was to focus on pure functions instead of sections of codes (as PGI did first): it was the concept of codelets (closely related to CUDA kernels). This provides a flexible mechanism for the programmer to describe the data layout and memory movements to and

from the GPU. HMPP also provides ways to specialize the produced code for the underlying architecture.

Here is an example of HMPP usage in C code. One should keep in mind that, in real code, situations are much more complex than in benchmarks. Therefore, programmers need to have access to rich tools to express their algorithms.

```
// simple codelet declaration

#pragma hmpp Hmxm codelet, args[a;b].io=in,
args[c].io=out, args[a].size={m,l},
args[b].size={l,n}, args[c].size={m,n}, TAR-
GET=CUDA

void mxm(int m, int l, int n, const double
a[m][l], const double b[l][n],
double c[m][n])

{    int i, j, k;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            c[i][j] = 0.0;}}
    for (i = 0; i < m; i++) {
        for (k = 0; k < n; k++) {
            for (j = 0; j < l; j++) {
                c[i][k] = c[i][k] + a[i][j] *
b[j][k];}}}
```

```
// usage of the codelet

#pragma hmpp Hmxm advancedload, args[a;b],
args[a].size={m,l}, args[b].size={l,n}

for (i = 0; i < nrep; i++) {

#pragma hmpp Hmxm callsite,
args[a;b].advancedload=true

#pragma hmpp Hmxm callsite

          mxm(m, l, n, (double (*)[m]) a,
(double (*)[n]) b, (double (*)[n]) c);

}

#pragma hmpp Hmxm delegatedstore, args[c]
```

Figure 32 : Example illustrating the richness of expression offered by HMPP's set of directives

The two following figures illustrate the main results obtained so far using HMPP on the CEA nVIDIA Tesla prototype: while it was quite easy to produce an efficient version of mod2am, disappointing performances were measured for mod2as. This is not surprising and agrees with the results found with the PGI compiler since GPUs are designed for matrix operations (hence leading to good performances for mod2am). The reduction operation contained in mod2as on the other hand is a very difficult algorithm to parallelize efficiently on GPGPUs.

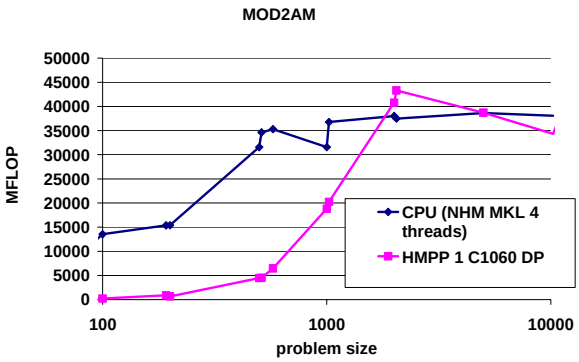

Figure 33: Nehalem-EP single socket mod2am performance versus nVIDIA C1060 using HMPP

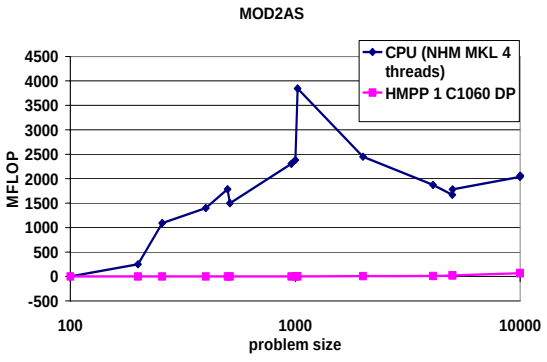

Figure 34: Nehalem-EP single socket mod2as performance versus nVIDIA C1060 using HMPP

StarSs

StarSs is a programming model developed at BSC for Cell, SMPs and GPUs. The main advantage of StarSs is that it allows easy portability of application across multiple platforms, with minimum changes in the code. Only the runtime for each of the languages under the StarSs programming model differs. The changes required in the code are mainly related to the

architecture of the target machine. Below we present the results of the EuroBen kernels for different languages under the StarSs programming model.

CellSs

The CellSs was installed on Maricel at BSC. Both mod2am and mod2as kernels of EuroBen were ported to Maricel using CellSs. Figure 35 shows the performance results for mod2am. It is implemented by using the partition matrix multiplication. It can be noticed from this figure that there are some variations in the performance as the problem size increases. This is because each partition block has size of 64×64 and its multiplied using the SPU blas library dgemm_64×64. As a result, the application has maximum performance when the problem size is a multiple of the block size i.e. 128×128, 192×192 etc. The lack of peak improvement, especially after the problem size is just over 192×192, 512×512 and so on, is because of the usage of more number of blocks to accommodate the few cells (matrix elements) that exceed the multiple of 64×64. These excess blocks will have zeroes appended to fill up the remaining cells that don't have any valid data, and the computation is carried out on those cells as well resulting in uniform time taken to execute the problem size. With the time being almost similar, and the problem size varying, we get dips in the graph when the problem size is not a multiple of the block size. This is explained by the equation:

$$\text{Performance (MFlop/s)} = (2 \times M \times L \times N) / X$$

where, M × L is the dimension of matrix A and L × N is the dimension of matrix B, and X is the time taken to do the multiplication.

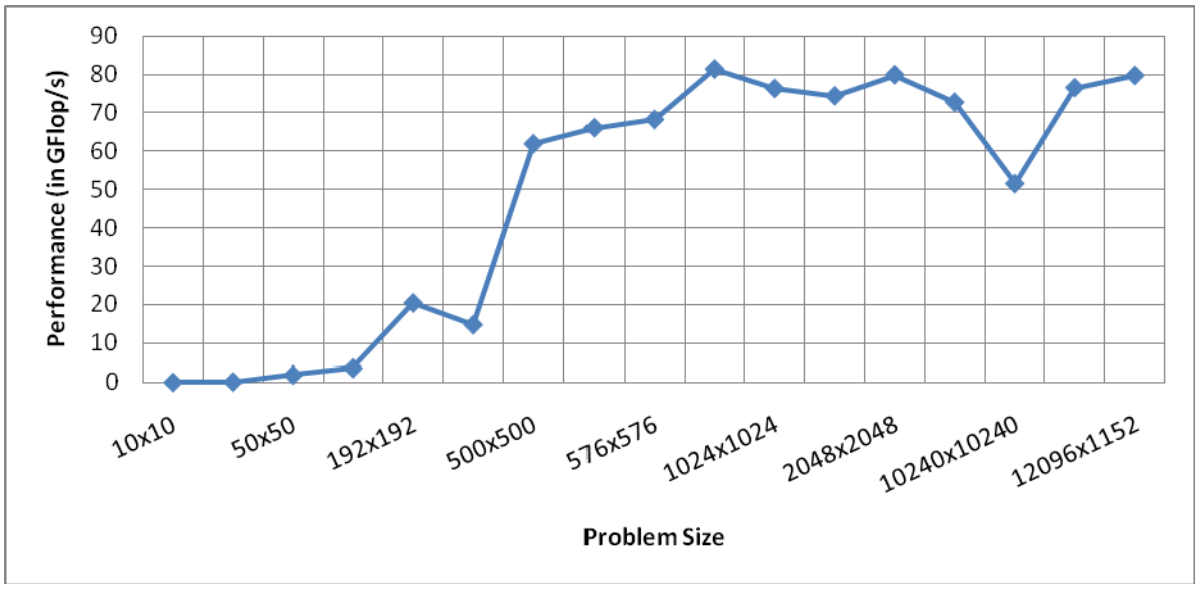

Figure 35: Performance of mod2am using CellSs on Maricel

Figure 36 shows the performance of mod2as using CellSs on Maricel. Like any accelerator, the performance of mod2as is very poor. Detailed analysis of the trace using Paraver [11] shows that the DMA transfer between the PPU and SPU takes lot of time. A better option in these types of application would be to run them on PPU instead of SPU.

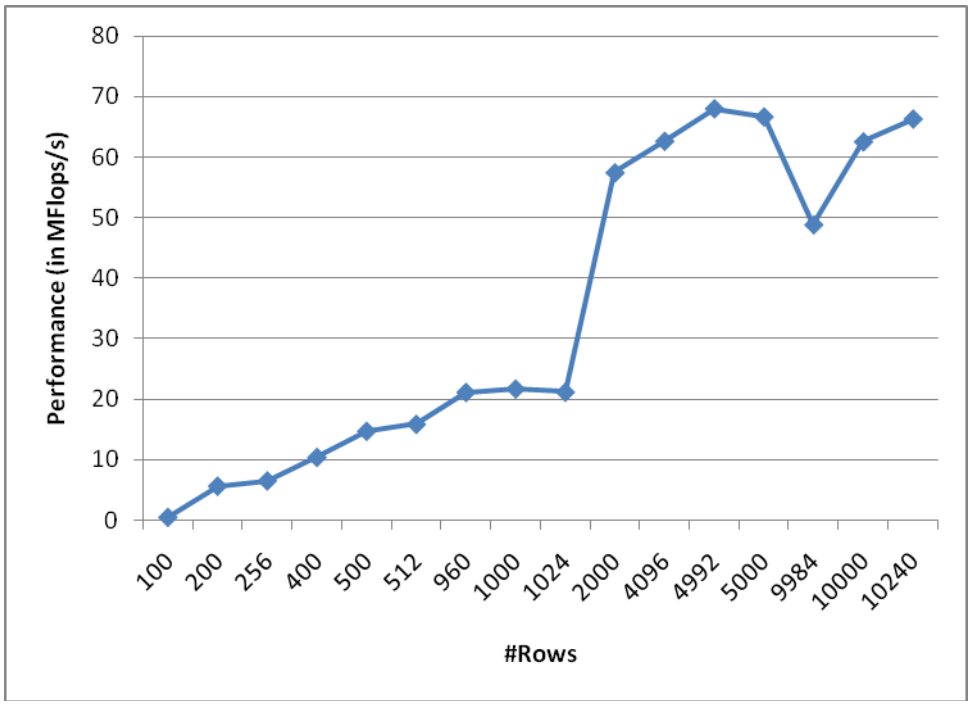

Figure 36: Performance of mod2as using CellSs on Maricel

GPUSs

The mod2am kernel of EuroBen kernel was ported to the nVIDIA Tesla using GPUSs. The porting took minimal effort from CellSs to GPUSs, and the only difference was that due to larger memory, multiplications were done in blocks of larger size. Figure 37 shows the performance of running on 1-4 GPUs with block size of 512×512 and 1024×1024. It is obvious that the behaviour of the graph follows the same pattern as that of the CellSs version of mod2am. However, the decrease in the performance of matrix multiplication of 12096 × 512 matrix with 512 × 12096 matrix is not just for the reason mentioned above. The CUBLAS library gives better performance with multiplication of square blocks (especially with the power of 2), but it does not give a good performance for rectangular blocks, even worse, when the size of the blocks are too small.

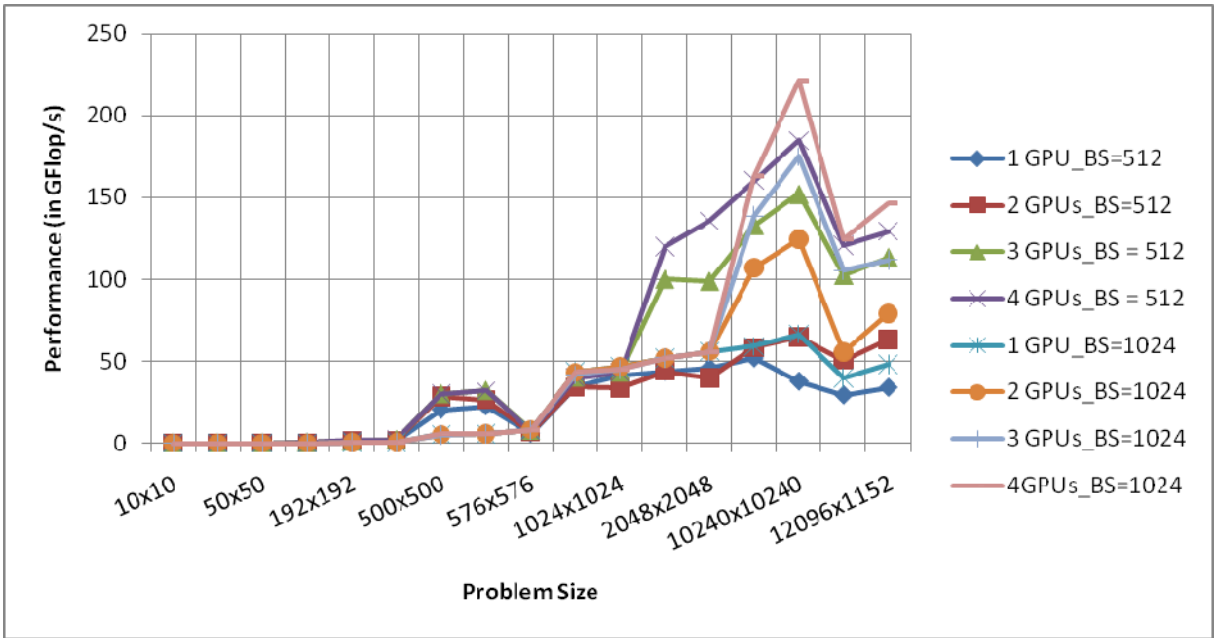

Figure 37: Performance of mod2am using GPUSs on nVIDIA Tesla

SMPSs

A SMPSs version of mod2am and mod2as were implemented on Nehalem and the performance is shown in Figure 38 and Figure 39 respectively. Here, SIZE refers to the number of input elements for each task.

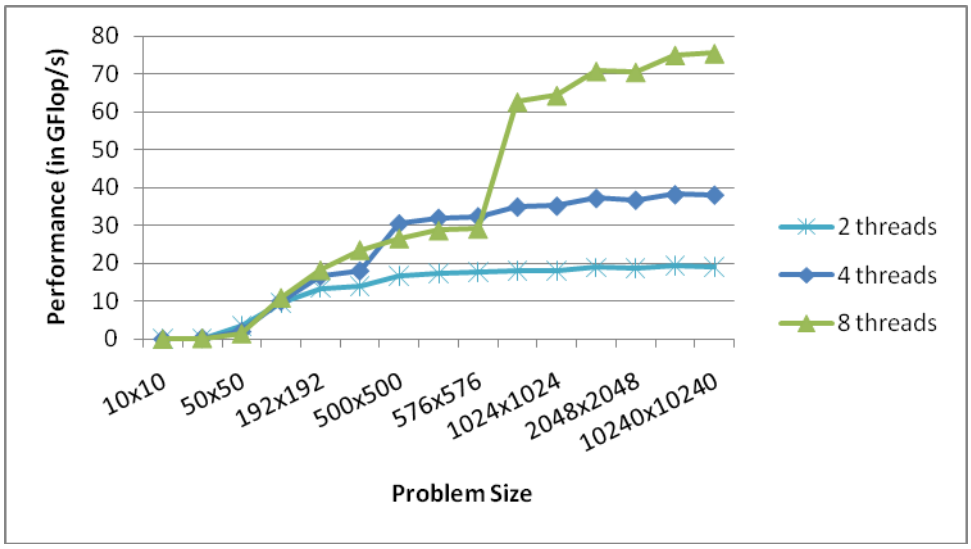

Figure 38: Performance of mod2am on Nehalem EP using SMPSs

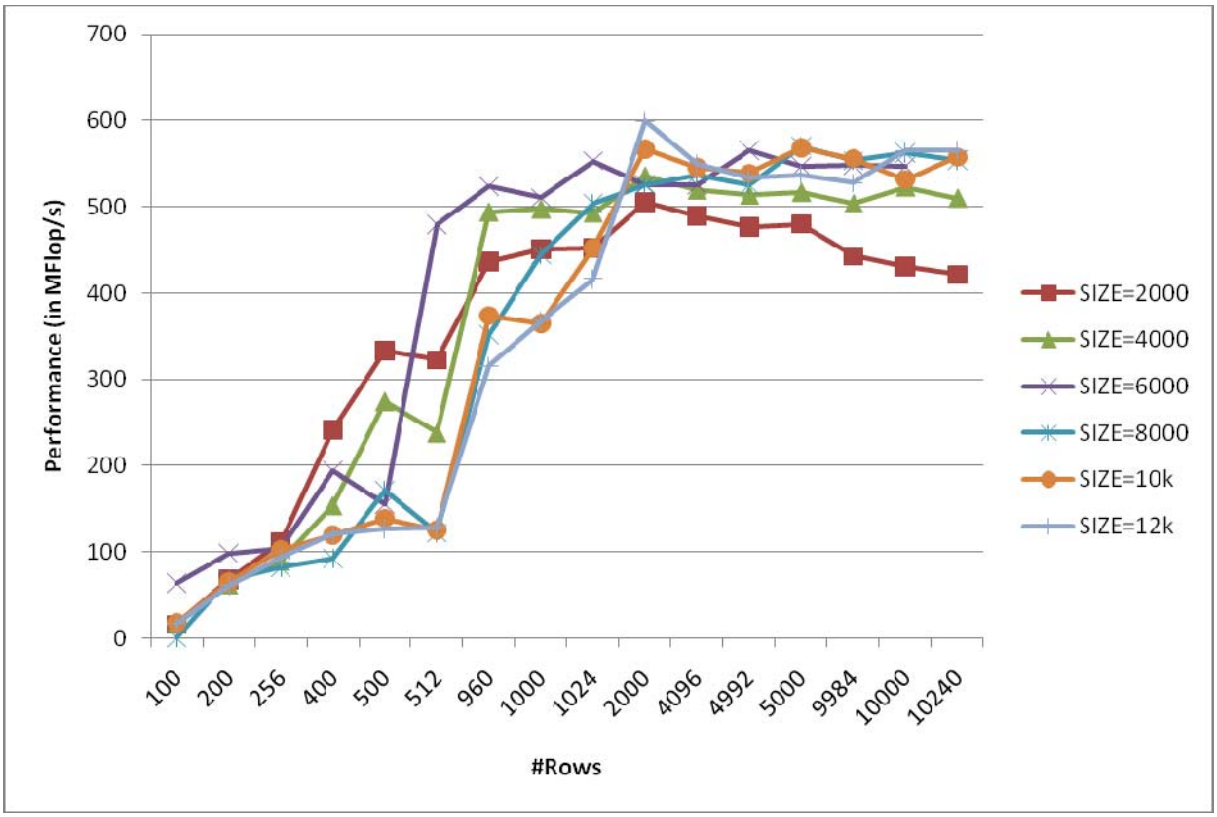

Figure 39: Performance of mod2as on Nehalem using SMPSs

ClearSpeedSs

The ClearSpeedSs version of the StarSs model has been developed and installed on the ClearSpeed-Petapath prototype. The porting from the GPUSs to the ClearSpeedSs version of mod2am took a couple of days. The reason was the different association (row/column wise) of the data being expected by the available kernels (CUBLAS → Csdgem). Still the two versions do look very similar. Had we had a compute kernel accepting the same format, only a change

of the actual kernel invocation would have been needed. Figure 40 shows the achieved performance for different problem sizes on 1 to 8 accelerator boards.

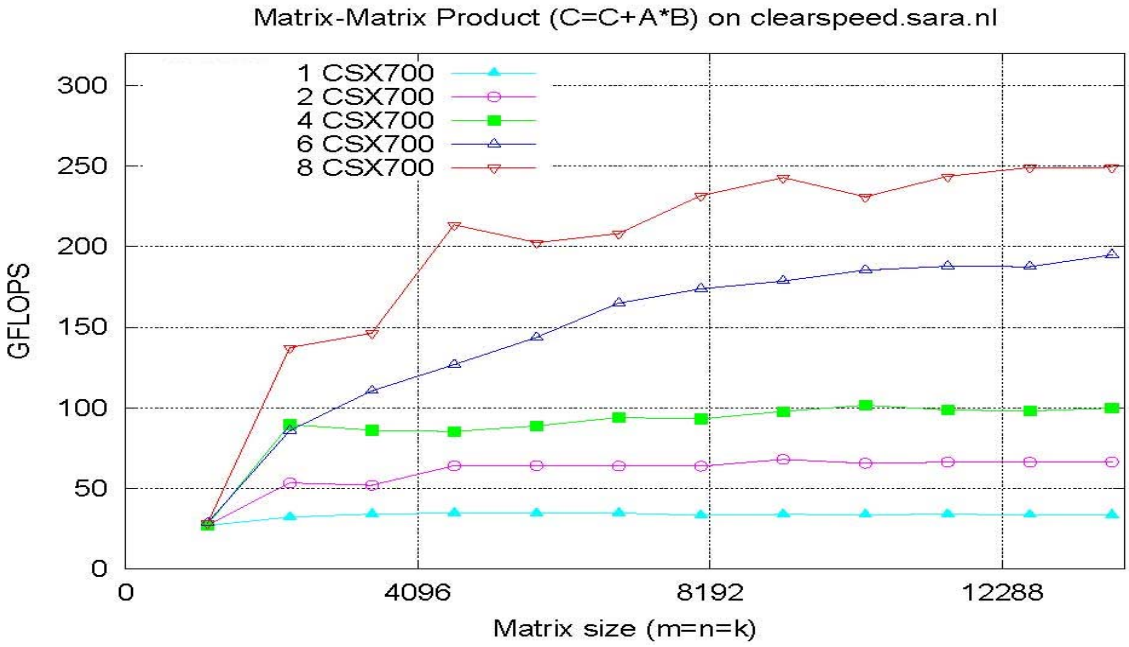

Figure 40: Performance of mod2am using ClearSpeedSs on the ClearSpeed-Petapath prototype

RapidMind

We have ported three EuroBen kernels (mod2am, mod2as and mod2f) to RapidMind (RM) and done various performance measurements. Table 1 gives an overview of the peak performance of all hardware.

Hardware	SP Peak Performance	DP Peak Performance
Nehalem-EP (2.53 GHz, 1 core)	20.24 GFlop/s	10.12 GFlop/s
Nehalem-EP (2.53 GHz, 8 cores)	161.92 GFlop/s	80.96 GFlop/s
1 C1060 GPU	933 GFlop/s	78 GFlop/s
1 PowerXCell8i (8 SPUs)	204.8 GFlop/s	102.4 GFlop/s
2 PowerXCell8i (16 SPUs)	409.6 GFlop/s	204.8 GFlop/s

Table 1: Hardware used in the RapidMind experiments.

For mod2am three different implementations of the dense matrix-matrix multiplication have been evaluated. First, a *"simple"* straightforward version was implemented (which consisted of only 20 lines of RM code). Second, a *"gpu-optimized"* version which was based on code available at the RM developer site was adapted. The "gpu-optimized" version used the RM datatype Value4f (4-tuples of floats) to store the matrices and operated on 4x4 sub-matrices which perfectly fitted the GPU's SIMD units. Third, a *"cell-optimized"* version was used. This version was again based on code available from RM, and worked on a block partitioning of 64x64 blocks. All matrices are in a "block swizzled" format to ensure contiguous memory access. Furthermore, computations and memory transfers are overlapped using double buffering. This elaborated version consists of more than 200 lines of RM code. A detailed discussion of the results is given in [41].

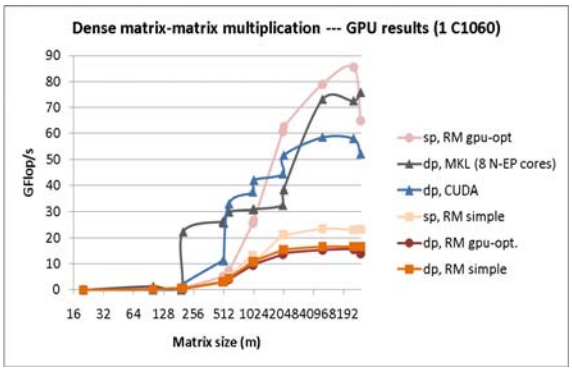

Figure 41: RapidMind results for mod2am (GPU)

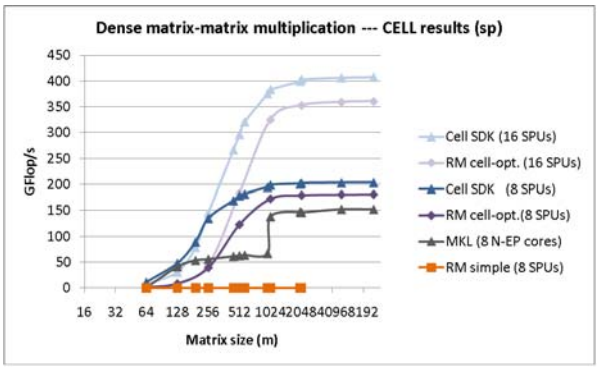

Figure 42: RapidMind results for mod2am (CELL)

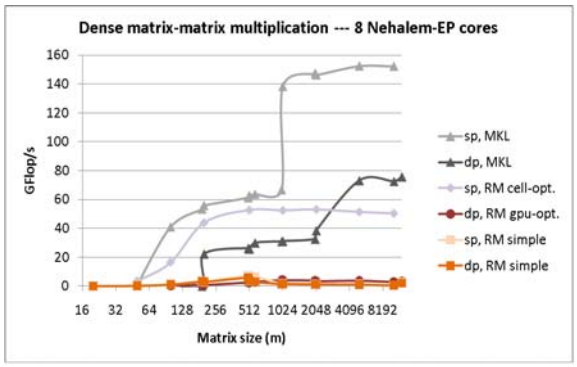

Figure 43: RapidMind results for mod2am (x86)

Figure 41, Figure 42 and Figure 43 show the results for all three mod2am versions on various backends. Results range from very bad (e.g., for the simple version on the RM "x86" and "cell" backend) to quite good performance (e.g., reaching 88% of a highly optimized Cell SDK sgemm code, which itself almost runs at peak performance).

The sparse matrix-vector multiplication (mod2as) is not well suited for accelerators (see Subsection 3.1.1). Figure 44 gives an overview of the results for mod2as on two backends and compares them with other languages. Figure 45 shows results for mod2f on various backends. Aside from the exceptional good performance of the cell-optimized mod2am version on the (single-precision only) RM "cell" backend, RM is at least a factor of 2.5 slower than the hardware specific implementations.

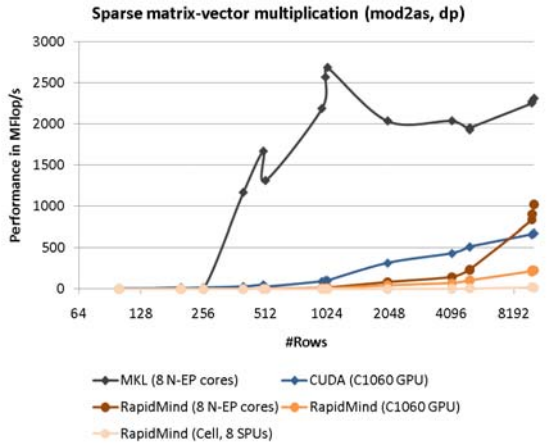

Figure 44: RapidMind results for mod2as

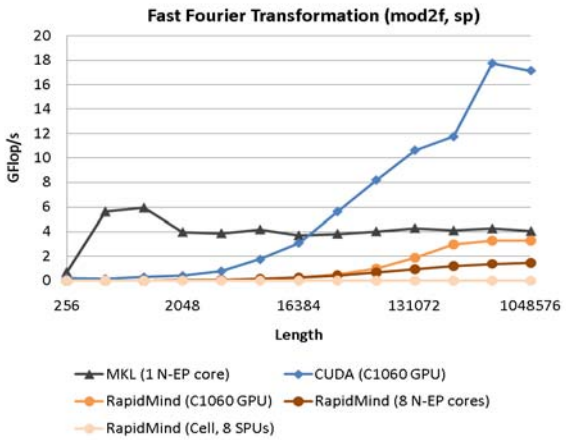

Figure 45: RapidMind results for mod2f

While pure code portability (taking existing code and running it on a different backend) is usually given (except some minor bugs found in the RM "cell" backend), performance is not at all portable across backends and performance prediction is very hard. Part of the performance behaviour for the different mod2am versions can be explained in retrospective (and after further discussion with RM staff members). E.g., Value4f was absolutely necessary to obtain decent performance on the "glsl" and "cell" backend, and blocking was not done by RM automatically but had to be taken care of by the programmer. It is important to note that these deficiencies are a sign of the missing maturity of the RM backends and the complexity that comes with supporting so many different backends which are often immature themselves

(e.g., remember the huge penalty of misaligned date accesses for older CUDA hardware). The development of High-level languages (HLL) would benefit greatly if OpenCL could become a standardized and high-performance abstraction layer for accelerators. The data stream programming model of RM enables much more aggressive auto-parallelization than any sequential languages can offer.

OpenCL

The OpenCL framework promises to create a unified language for executing codes on heterogeneous platforms consisting of CPUs, GPUs and other processors. It is generally expected to become the de facto language of accelerated computing and there has been a great amount of interest in it. Several vendors have been developing their OpenCL implementations at a rapid pace and the first limited beta versions were available already in Summer 2009, just 6 months after the ratification of the standard.

At CSC, testing was performed on beta versions of both AMD/ATI and nVIDIA OpenCL implementations. As the beta versions are under NDA, no details can be disclosed publically. Since the initial runs, both vendors have released new versions of their OpenCL implementations which should improve performance and fix a number of critical bugs. However, due to time constraints these could not be tested.

3.1.4 *FPGA experiments*

The timings from the original codes in this subsection were obtained on a Intel Xeon Paxville, running the code as compiled by gcc with full optimisation. For completeness also the reference performance values obtained on a dual socket Nehalem-EP system using the Intel compiler with full optimization as well as the Intel Math Kernel Library (MKL) (see Subsection 3.1.1) are listed in the table below.

mod2am

Algorithm version	Speed (MFlop/s)	Implementation Effort (day)	Lines of Code	Device Utilization	Clock Speed (MHz)
Original (SP)	587	0.5	38	N/A	N/A
Reference platform MKL (1 thread, 1024×1024) (DP)	9387	~ 0.25	277	N/A	2530
Reference platform MKL (8 threads, 1024×1024) (DP)	693000	~ 0.25	277	N/A	2530
HCE (on-board)	2859	10	543	93 %	67
HCE (off-board)	2463	10	543	93 %	67
VHDL (on-board)	1846	16	1252	39 %	115
VHDL (off-board)	1081	16	1252	39 %	115

Table 2: mod2am performance results

The VHDL version timings are taken from multiplying two 1024×1024 matrices together, as it is easier to work with powers of 2 in hardware implementations (a version able to work with non-power of 2 sizes would be more complex but probably just as fast). It performs 16 floating-point operations per clock cycle and runs approximately 3 times faster than the software

version (on-board), dropping to about twice as fast when the time taken to transfer data across the PCI bus is taken into account (off-board).

The Harwest port uses a similar strategy of caching subsections of the matrices in block RAM and using a pipelined tree of floating-point operations; however the Harwest port pipelines 63 floating-point operations rather than the 16 in the VHDL version and achieves better performance. The code is very much altered from the original C version, and contains unrolled loops, pipelined functions and HCE-specific code and directives.

mod2as

Algorithm Version	Speed (MFlop/s)	Implementation Effort (day)	Lines of Code	Device Utilization	Clock Speed (MHz)
Original (SP)	489	0.5	23	N/A	N/A
Reference platform MKL (1 thread, 1024×1024) (DP)	1254	~ 1.0	300	N/A	2530
Reference platform MKL (4 threads, 1024×1024) (DP)	3845	~ 1.0	300	N/A	2530
HCE (on-board)	34.6	4	132	55 %	67
HCE (off-board)	4.9	4	132	55 %	67
VHDL (on-board)	622	10	2490	43 %	110
VHDL (off-board)	6.2	10	2490	43 %	110
Original (DP)	489	0.5	23	N/A	N/A

Table 3: mod2am performance results

The sparse matrix by vector multiplication kernel (spmxv) is in some ways similar to the matrix by matrix multiplication (mxm). However, while the mxm performs a large number of operations on each item of input data and can therefore make good use of cache memory, the spmxv operation uses each element of the matrix only once. This means that it is memory bound and an FPGA is unlikely to have much advantage over a conventional processor. A modest speed up over the gcc-generated code on the host CPU was achieved with the VHDL version, probably due to the advantage gained by fetching from multiple external memory banks simultaneously. Additionally, the time taken to transfer the input data over the PCI bus to the FPGA device is a major bottleneck. When this is taken into account the performance of the VHDL version is impaired by a factor of 100.

The HCE version of the kernel uses a similar structure to the VHDL version but scope for optimisation is more limited as many of HCE's parallelisation directives only work effectively on relatively large data blocks of predetermined size, which does not apply well to this kernel. Although it was possible to speed up the original port twelvefold by pipelining 8 mul-

tiply and add operations, the HCE version is still much slower than the host or VHDL. It is likely that this could be improved further given more time.

mod2f

Algorithm Version	Max Speed (MFlop/s)	Implementation Effort (day)	Lines of Code	Device Utilization	Clock Speed (MHz)
Original (DP)	481	N/A	420	N/A	N/A
Original (SP)	465	0.5	420	N/A	N/A
Reference platform MKL (1 thread) (DP)	2778	~0.5	398	N/A	2530
VHDL (on-board)	2284	11	2119	50 %	115
VHDL (off-board)	1177	11	2119	50 %	115

Table 4: mod2f performance results

When running on data in its internal memory, the FPGA reaches speeds approximately 6 times faster than the original gcc-generated code running on the host CPU. But comparing the FPGA-based results to the reference platform performance one clearly can see that the latest generation of general purpose processors do provide FFT performance values which are quite comparable to the values obtained on FPGAs. The original kernel is a radix-4 FFT. The VHDL version is a simple radix-2 implementation but gives identical results. When its pipeline is filled, it can complete 2 FFT butterflies (each one comprising 10 individual floating-point additions, subtractions and multiplications) per clock cycle. At 115 MHz this gives a sustained speed of almost 2.3 GFlop/s.

Table 4 above shows the maximum performance achieved by each implementation. As the FFT performance is more sensitive to the size of the data set than that of the other kernels, performance for each data set size is given in Figure 46:

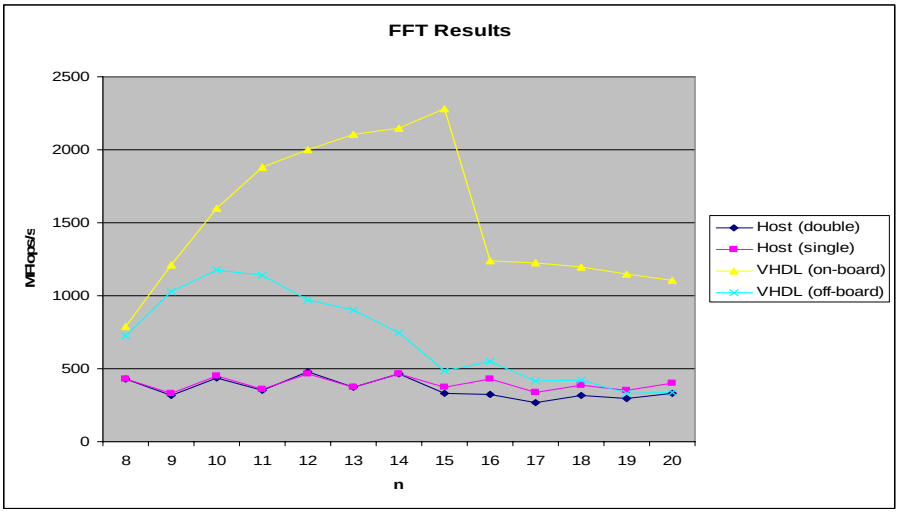

Figure 46: Performance of mod2f for different problem sizes

The performance of the VHDL version varies much more than that of the host; with faster computation, factors such as memory bandwidth, PCI bus bandwidth and start-up overhead become more significant. Looking at the on-board performance, the VHDL version is fairly

slow for a 256 point transform – this data size is small enough that start-up overheads for the algorithm and for each pass become noticeable. As the size increases, performance improves and reaches its peak for the 32768 point transform, which the FPGA can perform around 6 times faster than the gcc-code run on the host CPU. This is the largest transform which can be performed entirely within the FPGA's internal block RAM and does not require any transfers to or from DRAM during the algorithm. For larger sizes, performance tails off as the rate at which data can be read from or written to the DRAM banks becomes more and more significant. The FPGA is still faster than the host by a factor of 3-4, probably because reading and writing 4 separate external memory banks simultaneously gives it an advantage. When the time taken to transfer data to and from the FPGA is taken into account (off-board row in the table), much of the VHDL version's speed advantage is lost. Unfortunately there was insufficient time to complete a Harwest port of this kernel.

mod2h

Algorithm version	Speed (millions of values per second)	Implementation Effort (day)	Lines of Code	Device Utilization	Clock Speed (MHz)
Original (SP)	14.4	1	37	N/A	N/A
Reference platform single core (DP)	3700	~ 0.25	37	N/A	2530
HCE (on-board, int)	217.6	6	171	26 %	80
HCE (off-board, int)	4.0	6	171	26 %	80
VHDL (on-board)	900.0	6	690	54 %	150
VHDL (off-board)	8.1	6	690	54 %	150

Table 5: mod2h performance results

The VHDL version of the algorithm is capable of generating 900 million single precision values per second (6 per clock cycle with the design clocked at 150 MHz). This algorithm, consisting primarily of bit shifts and exclusive or operations, was particularly amenable to FPGA acceleration. The main transform of the random number generator can be expressed elegantly in VHDL and it was also possible to eliminate the final floating-point scaling from the VHDL implementation as it was just a bit shift, replacing it with a custom integer-to-float conversion which completes in one clock cycle instead. However, when the time taken to transfer the results back to the host is taken into account, the VHDL version's performance is dramatically reduced by a factor of more than 100, making it slower than the software kernel. The bandwidth over the PCI bus appears to be limited to approximately 32 MB/s.

Porting to Harwest was slightly complicated as the original kernel makes extensive use of 64-bit integer operations, which Harwest does not support. The algorithm had to be rewritten and tested using multiple 32-bit operations to achieve the same result. Because Harwest does not support DP floating-point, this port passes back the unconverted 64-bit integer results so that the host can perform the final conversion to floating-point in full precision. By employing similar optimization strategies to those used in the VHDL version (using HCE-specific directives to turn the main transform into a combinatorial operation, and running 3 separate instances of it in parallel), numbers can be generated at a rate of 217 million per second, much

faster than the host, though slower than the hand-coded VHDL. However this is subject to the same PCI bus bandwidth problem, so when the time taken to transfer the results back to the host is included, the speed again falls to 4 million values per second.

Notes on Power Consumption

Low power consumption is a major advantage of FPGAs so it is worth considering the PRACE kernel implementations in this context. Unfortunately the Maxwell system does not provide a facility for measuring the actual power consumed by the accelerator boards, so only a very rough estimate of power usage based on data available online is possible.

Virtex-4 devices are much more power efficient than those of previous FPGA generations. Information released by Xilinx suggests that a large Virtex-4 FPGA (in this case an LX device, but the FX family used on the AlphaData boards in Maxwell is likely to be similar) running a demanding bitstream with most of the slices occupied at 200 MHz is likely to consume around 2-3 W. Given that our implementations of the EuroBen kernels all run at 150 MHz or less and mostly occupy a lower percentage of the logic slices, it seems reasonable to assume that they also will not consume more than 3 W. Assuming 3 W power consumption, the Flop/s-per-Watt value for the EuroBen kernels reaches almost 1000 MFlop/s-per-Watt (for the matrix multiply kernel which is the fastest). This compares very favourably to conventional CPUs, which may achieve an order of magnitude less.

3.1.5 *LRZ + CINES (Phase 1)*

This section describes the LINPACK HPL porting on the LRZ-CINES phase 1 prototype AMBRE.

This work has been done with the helpful collaboration of ClearSpeed-Petapath, using a similar approach to a work done on TSUBAME (at the Tokyo Institute of Technology), taking advantage of both accelerator and processors of LINPACK HPL run on the whole configuration. The approach of energy efficiency remains the main focus of this evaluation.

The LINPACK HPL application measures the performance of a cluster when solving linear equations, which heavily uses DGEMM and DTRSM. The CSXL BLAS library can be used to distribute the DGEMM computation between the host processor cores and the ClearSpeed processors.

DGEMM Load Balancing Between Processors

The LINPACK application is linked with the CSXL BLAS library. The CSXL library will intercept all of the BLAS function calls, such as DGEMM and DTRSM, and then distribute the compute between the CSX710 processor and the CPU cores as appropriate.

CSXL provides an environment variable (`CS_BLAS_HOST_ASSIST_PERCENTAGE`) for the user to help the library distribute the compute between the CPU and CSX710. If this variable is set to zero, all of the computation will be sent to the CSX710; when the variable is set to 100, all of the compute is sent to the CPU cores.

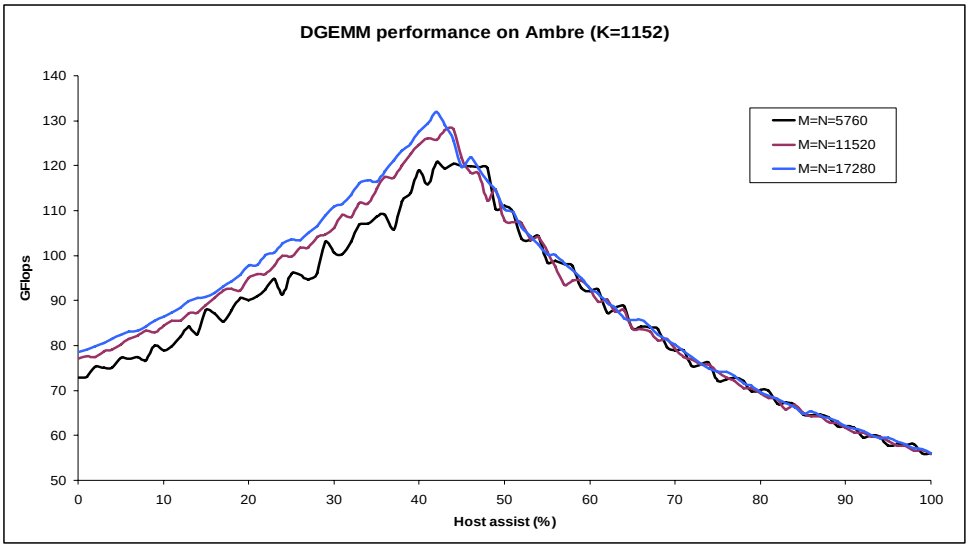

Figure 47: CPU/CSX710 (K = 1152)

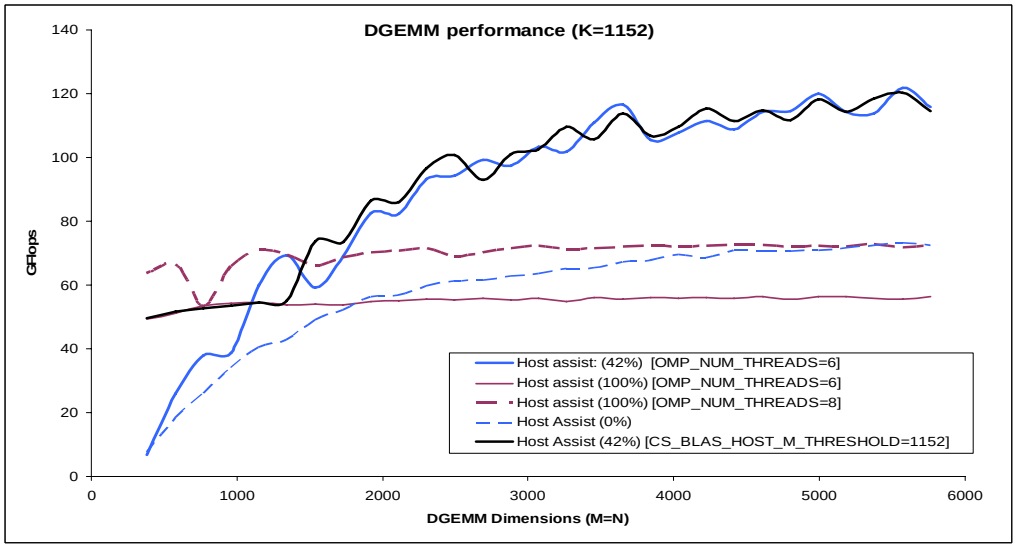

Figure 48: CPU / CX710 across matrix sizes

Figure 47 shows the DGEMM compute performance over the full range of host assist percentage for a number of DGEMM dimensions. With a zero host assist percentage, the CSX710 provides between 70 and 80 GFlop/s. With a 100 % host assist, the CPU provides between 50 and 60 GFlop/s, where each CPU core is contributing between 8 and 10 GFlop/s. When the hosts assist percentage is between 40 % and 50 %, both the CPU and CSX710 combine to provide 120 and 130 GFlop/s; which is not equal to the combined peak of the two processors, but definitely an improved performance over the processors in isolation.

According to the CSXL user guide, to offload the DGEMM computation to the CSX710, the matrix dimensions must be a multiple of defined values. M and N must be a multiple of 192 and K must be a multiple of 288. For dimensions that are not a multiple of these values, the DGEMM is split into parts where the largest part is performed on the CSX710 and the DGEMM "edges" are performed on the host. For the purposes of the current LINPACK benchmarking, the dimensions have been chosen so those are a multiple of the defined values.

A host assist percentage of 42 % was chosen from the results presented in Figure 47 since this is where the best peak performance is obtained. Figure 48 compares the DGEMM performances of the host BLAS with OMP_NUM_THREADS=8, OMP_NUM_THREADS=6, with the CSX710 and the combined CPU and CSX710 across difference matrix dimensions.

The CPU performance is fairly consistent across the different matrix dimensions, and the eight core performance is approximately 33 % better than that of six cores.

HPL on the 32 Node Cluster

Figure 49 shows the performances while tuning the CSXL BLAS library host assist parameter. From this plot it can be seen that a host assist of 44 % provides the best performance. This means 44 % of the DGEMM compute is performed by six of the eight cores using the Intel MKL Blas library and 56 % of the DGEMM compute is performed on the one CSX710 accelerator processor. Only six of the eight cores are used by the MKL library (configured by the OMP_NUM_THREADS environment variable) because one core is required to manage the CSX710 processor and MKL operated more efficiently with an even number of cores.

Figure 50 shows the performance of the AMBRE cluster with and without the ClearSpeed-Petapath accelerator cards. This is run using the CSXL BLAS library with the host assist percentage set to 42 % and 100 %, where at 100 % all of the DGEMM compute is passed to the Intel MKL BLAS library. There is a small overhead in the CSXL forwarding the DGEMM to MKL but is negligible in this context.

Over the 32 nodes of the AMBRE cluster the CSX accelerators add 1.4 TFlop/s to the LINPACK score for the given HPL parameters when N=361,152. This is 43.75 GFlop/s added per CSX710 card. The final LINPACK score is 3.5 TFlop/s.

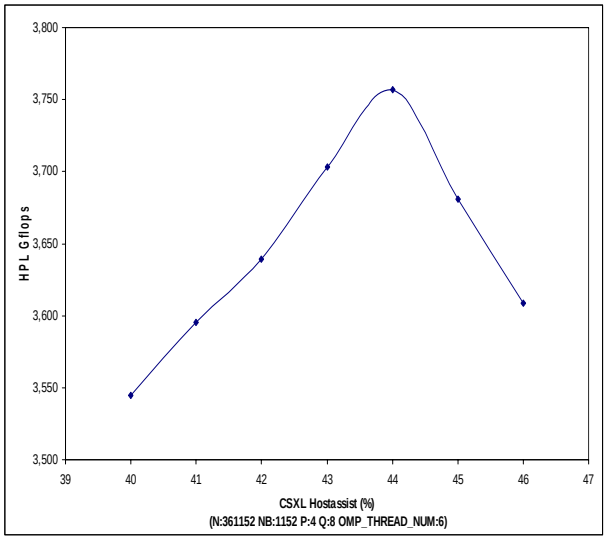

Figure 49: HPL / 32 node + CSXL with varying hostassist

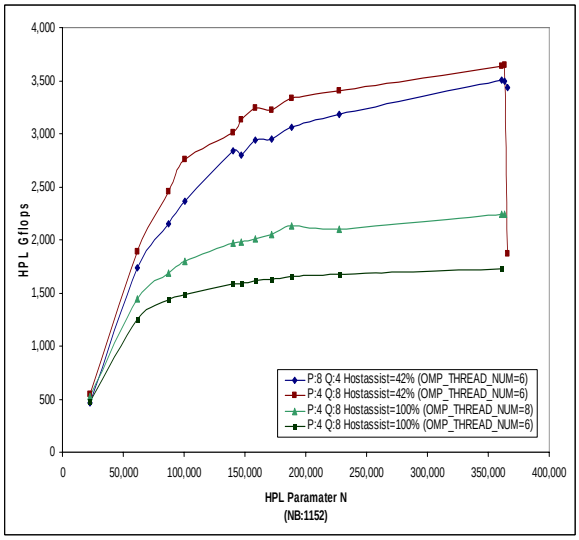

Figure 50: HPL / 32 node + CSXL with varying memory size

Summary

The prototype AMBRE, using the 32 nodes each with one CSX710 accelerator card, achieves a LINPACK score of 3.5 TFlop/s. The same cluster without using the ClearSpeed- Petapath cards achieves 2.24 TFlop/s. For these LINPACK parameters 43.75 GFlop/s is added per CSX710 card to the host core compute.

Host Only LINPACK		Accelerated LINPACK	
Score	From the peak	Score	From the peak
2.24 TFlop/s	86 %	3.50 TFlop/s	63 %

The LINPACK parameters, using the ClearSpeed-Petapath accelerator cards, have been tuned using the N, P, Q and host assist. These parameters affect the LINPACK score the most and

the 3.5 TFlop/s score is very close to optimum LINPACK score that is expected using the AMBRE hardware, reaching 86 % from peak performance without using CSX710. The NB parameter has been set to 1152 throughout this report, where this is a known optimal value for LINPACK using the ClearSpeed-Petapath accelerators, leading to 63 % of peak performance of both Nehalem-EP processors and CSX710 accelerators.

To add additional GFlop/s to the current score the easiest and most effective approach would be to add further accelerators to the hardware. Adding a second CSX710 processor to each of the cluster nodes would add additional computational power and could add a further 40 GFlop/s per node or 1.2 TFlop/s to the existing LINPACK score.

*From the energy point of view, the CXS710 represents **10 %** of a node with two Nehalem-EP 2.53 GHz, while their contribution in LINPACK brings a speedup of **56 %.***

3.1.6 *LRZ + CINES (Phase 2)*

This subsection compares the application performance of RAxML on the SGI ICE prototype with the corresponding RAxML performance on three other system architectures (BlueGene/L at San Diego Supercomputing Center (SDSC), Infiniband Opteron cluster at Technical University Munich (TUM) and SGI Altix4700 at LRZ). The second paragraph describes the influence of the topology aware scheduling (optimal placements of the user threads concerning intra-node and inter-node communication topology) on the performance of the real world application Gadget.

Application tests

RAxML

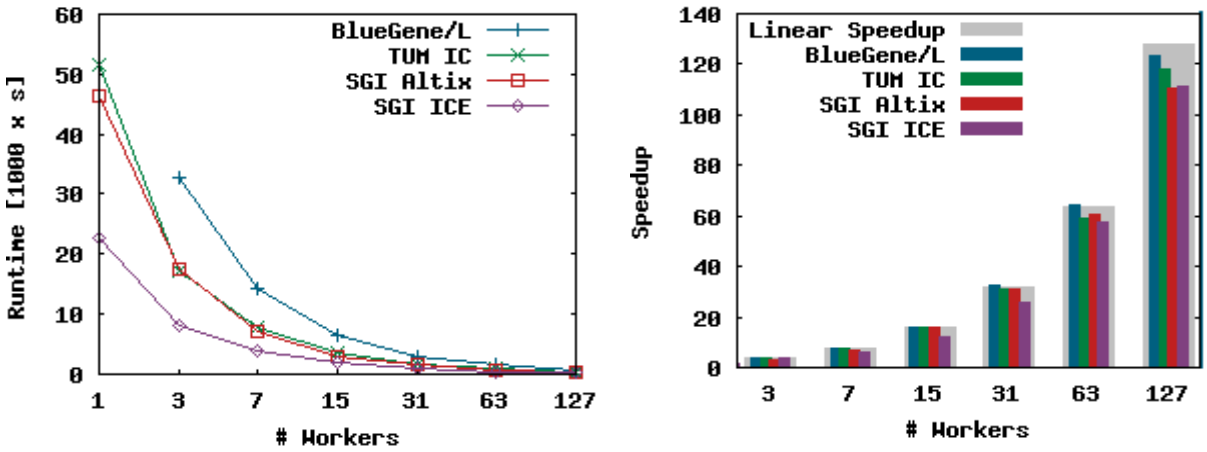

Figure 51: Runtimes and Speedups for synthetic DNA dataset with 50 sequences

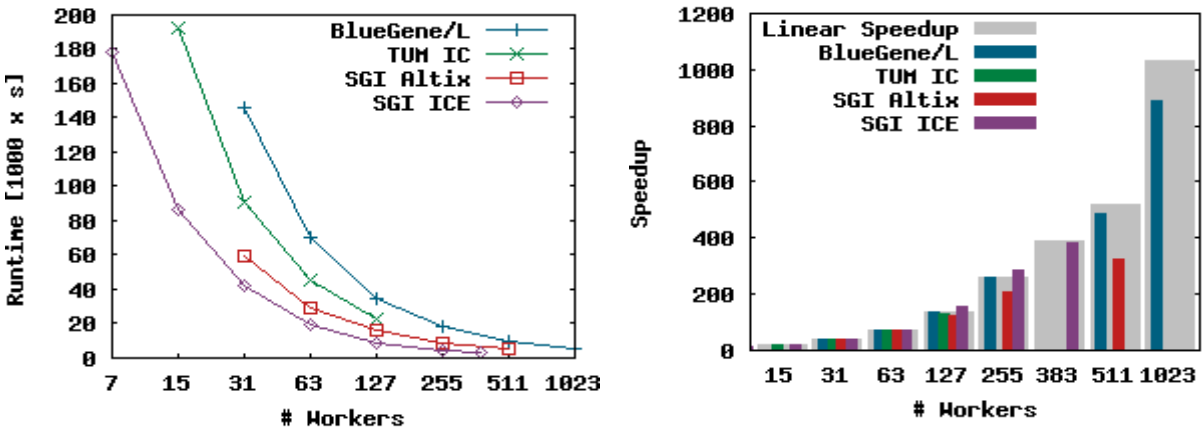

Figure 52: Runtimes and Speedups for synthetic DNA dataset with 250 sequences

The experiment with the smaller 50 sequences and 500,000 basepairs datasets (Figure 51) shows runtimes on the ICE system that are approximately 75 % lower than on the BlueGene/L and more than 50 % lower than on the SGI Altix4700 and the Opteron Cluster. Since RAxML is memory-bandwidth bound, these results were expected given the high aggregated memory bandwidth of the Intel Nehalem-EP processor. The scalability looks similar for all 4 systems, though the ICE prototype performs slightly worse than the other 3 systems.

Due to the runtime and/or memory limitations, the experiments with the 250 sequences and 500,000 basepairs dataset could not be conducted with a small number of workers. Therefore a few numbers are missing in Figure 52. The scalability evaluations for this dataset are relative to the run with the least amount of workers that was possible. Though the numbers are not easily comparable, the ICE system seems to outperform the BlueGene/L for small core counts.

Topology aware scheduling

Future Exaflop/s systems will very likely consist of up to 100 million processor cores (and even more logical cores (SMT)) in several 10^5 compute nodes. In order to keep the cost of the network interconnect within 20 % of the total hardware investment budget, the topology of the interconnect of these systems will most likely be realized as 3D-network torus or n-dimensional hypercube, as dragonfly network, as pruned fat tree or some other type of hierarchical network interconnect with position dependent communication latencies of the communication partners and even position dependent per processor core network bandwidth.

As showcase to pin down the influence of the optimal placements of the user threads on the network topology, LRZ had used the German 9728-core SGI Altix4700 system HLRB II and the PRACE core application GADGET [31] [32]. Please note that these kind of evaluations cannot be accomplished on small prototype systems since the influence of topology-aware scheduling on application performance really becomes relevant (measurable) on large systems with hierarchical network interconnect like the actual HLRB II.

Figure 53 shows the network topology of the 9728-core SGI Altix4700 system HLRB II at LRZ. The system consists of 19 partitions each containing 128 or 256 Intel Montecito blades which are connected through a NUMAlink4 (NL4) 2D-torus network. Within each partition the NL4 network is organized as a fully non-blocking fat tree topology. Hence the measured bisection bandwidth values per blade pair vary from 1.6 GB/s for intra-partition communication to 0.2 GB/s for the total system.

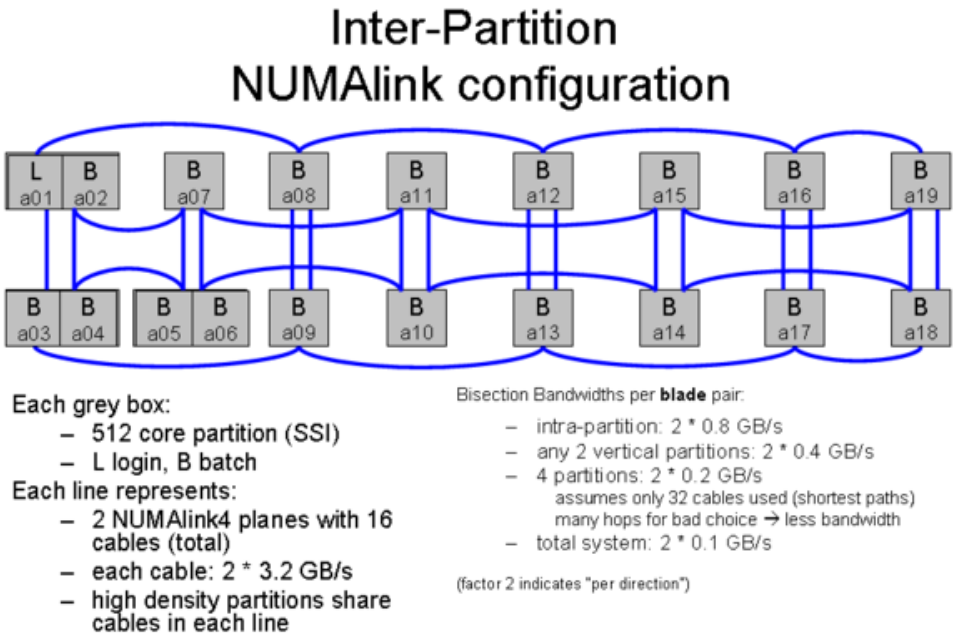

Figure 53: Network topology of 9728-core SGI Altix4700 system HLRB II at BAdW-LRZ

Table 6 lists the measured execution times for 256-core GADGET runs on the German Altix4700 supercomputer HLRB II using different task placement strategies, under the following placement conditions:

- Optimal placement of the MPI tasks using the topology-awareness feature of the scheduling software Altair PBSPro (256 MPI tasks on one 512-core partition and optimal placement of the MPI tasks concerning the number of NL4 router hops within the partition);
- Distribution of 8 32-core MPI chunks on 8 different 512-core partitions.

In average the run times of GADGET using the topology-aware scheduling feature of PBSPro are reduced by 10 %. The observable deviations in application performances will presumably even increase for large-scale applications due to other effects such as congestion of network links due to suboptimal network routing algorithms or I/O traffic. Hence, for future multi-Petascale systems topology-aware scheduling algorithms will be a very important tool to optimise application performance as well as to ensure that measured application performance values are reproducible.

Run	1	2	3	Average
Optimal placement (256 MPI threads on one 512-core partition)	28.95 s	28.89 s	28.92 s	**28.92 s**
Suboptimal placement (Placement of 8 32-core chunks on 8 512-core partition)	32.54 s	31.82 s	32.38 s	**32.25 s**

Table 6: Measured execution time of 256-core GADGET runs[4]

[4] Gadget test case1: 32,768,000 particles and 256 MPI tasks

SGI Management Center

The SGI Management Center (SMC) is the core component in a group of applications, which are targeted to provide an integrated solution for management of heterogeneous cluster environments.

The SMC acts as the central component and provides visualization of the cluster environment, monitoring the state of the nodes and tools to install ("provision") new or maintain currently installed software. Currently only Altix XE, CloudRack X2 and Octane III systems are directly supported, integrated support for the SGI UltraViolet systems is planned for Q2 2010.

According to current plans, the SMC will provide a GUI driven interface to the Tempo cluster management software, which is currently CLI-only management tool for SGI ICE systems. However, due to the early stage of development, no further details are known and integration and support may change.

As SMC is supposed to be used in heterogeneous environments, support of non-SGI platforms is provided by a couple of commonly used management tools like ROAMER, IPMI, DRAC or ILO. Due to the limitations within these tools, only part of the SMC functionality may be available on other platforms.

Another interesting SMC feature is the Application portal. The Application portal is an integrated, web-based user interface to create, submit jobs and monitor them during runtime. It currently has integrated support for FLUENT, Abaqus and ANSYS and an integrated interface for Altair's PBSPro. An interface to add custom applications is planned for the near future. This feature could not be tested in detail, as neither the software packages nor PBSPro where installed and licensed in the test environment.

The main SMC window is built out of frames, which can be freely arranged around the central frame. The central or system frame provides three tabs named Configuration, Instrumentation and Provisioning to perform the corresponding tasks (configure nodes, monitor the system and manage new and installed software). Additional frames, for e.g., list the available host (~ groups), event logs or management of software images can be added to the central frame and different customized layouts, thereby providing different views of the system that can be saved as so called "dashboards".

The host frame displays via predefined icons a quick overview over the state of single hosts or a complete host group. Hosts, which exceed a user definable warning threshold, thus can be easily identified. Via this tab it is also possible to open system console windows to a single host or a group of hosts.

The default installation provides a subset of predefined metrics under the instrumentation tab, which can be easily extended by selecting additional metrics from the predefined list. Each of these options can be used for single hosts, a selectable subgroup of host or a complete host group.

SMC also includes a monitoring and event system to track system values and display this information via the SMC GUI. These monitors run periodically on the cluster and provide metrics which are then displayed via the SMC. The standard system can be extended by the custom monitors, which may be user-defined scripts or programs to measure specific events and return the data to be displayed by the SMC.

The provisioning tab enables the administrator to manage multiple different payloads and kernels. A payload is a compressed file system, which is used during the installation, and contains installed software and system packages. Different payloads and kernels can be combined into different images, which allow possible harmful upgrades to be easily undone. The provisioning of nodes, which is done in parallel via multicast, provides life upgrade of systems if

only minor changes are done as well as complete system upgrades. It further allows scheduling an immediate upgrade including a reboot or allows setting a flag to perform the upgrade during the next reboot.

In the current version of SMC, two major drawbacks where found. For one, it is possible to reset or power down nodes with one click and without any further confirmation. This might be fatal, if by accident a node group instead of a single node is selected. The other major disadvantage is the lack of history information in the instrumentation tabs. In the current implementation, monitoring of a host starts when the node is selected and ends, when the node is deselected or another node is chosen. This approach severely limits the usefulness of the SMC for monitoring and analysis of recent problems.

In summary it can be said, that the SGI Cluster Management provides an intuitive GUI based interface for the management of clusters and combines different tools, which either have been provided in different packages or could only be used via a command line tool. Due to the pre-production state of the software, not all functions could be tested and it is expected that some options will be changed and improved in future releases.

3.1.7 *Intel MIC Architecture*

Gadget

In cooperation with Intel, LRZ has started to evaluate the ease-of-use and potential performance gains of the Intel MIC architecture for a compute-intensive, widely used astrophysics code - GADGET-2 for cosmological N-body/SPH simulations. GADGET is also included in the PRACE benchmark suite.

First, the most time consuming routines have been identified using some common benchmark data sets and state-of-the-art analyzing tools. A closer look at the code and especially the routine `force_treeevaluate_shortrange()` showed that the "tree-structure" employed in Gadget needs some adaptation to allow an easy vectorization of the code. The CUDA-enabled Gadget version from Carsten Frigaard[5] provides a modified implementation of this function -tailored to massively parallel devices- which was used as a basis for the porting to the Intel MIC architecture. The porting and optimization of this routine to the Intel MIC architecture was straightforward and took only a couple of days. Scaling tests of this routine showed that the performance on the Intel MIC architecture is faster than the CUDA version on a recent GPU.

Orlando Rivera (LRZ) is now working on the integration of this kernel into the Gadget application. This requires a porting of Gadget and its libraries to different operating systems and an incorporation of the data transfer via the PCIe bus. The porting of Gadget and the numerical libraries has been done; the optimization of the data transfer is work-in-progress.

TifaMMy

General description of the code

The TifaMMy [43] library uses a block-recursive approach based on space filling curves to avoid cache misses in cases of executing level three BLAS and LAPACK algorithms [44][45].

The parallelization of the approach is implemented by using the OpenMP 3.0 [46] task concept which allows parallelizing nearly all kinds of code by defining the number of available processors, the tasks which can be executed in parallel and possibly requires barriers. Code

[5] G2X: GADGET2 optimization Using the CUDA Architecture (http://frigaard.homelinux.org/g2x/)

examples showing how this has successfully been applied to TifaMMy's recursive algorithms can be found in [45].

For porting the algorithms to the Intel MIC architecture based prototype processor software development platform, it has to be taken into account that such a processor could be plugged onto a PCIe board. Hence, in this set-up it can be regarded as a system in a system with its own and separate memory and address space. This means that all data required for processing must be explicitly exchanged through the PCIe bus.

Transferring the data structures to the separate memory address space on the Intel MIC architecture processor based environment requires special care regarding any pointers involved, such as converting into symbolic addresses and relocating afterwards.

As described in Subsection 2.1.4, the underlying version of the Intel MIC architecture provides in-order CPU cores. Thus, the recursive data structure should be created with the system's "normal" processor. However, as our data structures are implemented in C++, there is no easy access to the underlying matrix elements. Due to the technical reasons, it is only possible to copy flat data specified by the starting address and a certain number of bytes to the PCIe board's memory. The task is to assemble a flat memory structure from different composed C++ classes. Hence, we are able to derive several descriptive patterns from our class structures with a specified size and one data stream that contains all matrix elements. After copying these flat models via PCIe, we used new constructors in our application to rebuild the data structure on the accelerator board without doing the recursive construction. When the computations on the board are finished, we only need to copy the data stream back to the computer's main memory and replace the elements in the matrices on the host.

By using this mechanism, the same code can be used on standard CPUs and on the Intel MIC architecture as is. However, to be able to fully exploit latter's performance potential, it is highly recommended to use the vector/SIMD capabilities via e.g., intrinsics which come with the Intel compilers as part of the software development vehicle. As these had already been used intensively in previous versions of TifaMMy being written for SSE3 based architectures, we simply had to convert these vector-intrinsics into the new 512-bit format, applying the corresponding vector intrinsic. The utilized version of Intel MIC architecture offers a variety of new vector instructions, which are described in detail in [47].

In addition, some routines had to be added to implement the data exchange mechanism described above. Due to OpenMP 3.0 support on the Intel MIC architecture software development vehicle, it is not necessary to re-implement the core algorithms of our linear algebra modules. Generally speaking, this can be regarded a key advantage of the Intel MIC architecture compared to other parallel accelerator architectures such as e.g., CELL, GPU, DSP or FPGA based solutions. A vast number of existing parallel codes have been implemented through common shared memory libraries like Pthreads, OpenMP, or Threading Building Blocks, the porting effort to the Intel MIC architecture is straightforward and very low compared to rewriting the code for e.g., CUDA or OpenCL. The main programming effort has to be put into the data exchange (however, this is necessary for any accelerator card based architecture) and into applying vector intrinsics. As any code being optimized for recent x86 CPUs should use SSE intrinsic for the sake of performance and efficiency, porting these vector instructions to the Intel MIC architecture's vector instructions is rather easy and straightforward.

Hence, as a conclusion we think porting an x86 based optimized parallel code to the Intel MIC architecture does not impose too much of a burden to the programmer. Especially the general purpose programmability and flexibility of the MIC architecture based processors make it very suitable for complex highly parallel workloads.

Performance Results

This section provides a short performance analysis of TifaMMy on the Intel MIC architecture. Two aspects are very important: scalability and absolute performance in comparison to a traditional multi-core processor based system (using "large" cores). Here, a very recent x86-based processor high performance system is compared to the Intel MIC architecture. Figure 54 shows the archived speed-ups (and scalability), while Figure 55 shows the absolute performance of TifaMMy on the Intel MIC architecture software development vehicle. More details cannot be provided here due to NDA restrictions.

As one can see in Figure 54 the scalability of the Intel MIC architecture is slightly beneath the tested x86 system but the Intel MIC architecture scales well up to a (much) larger number of cores. In addition, it should be considered that the graph shows the scalability for best absolute performance on Intel MIC architecture. For TifaMMy this means that 3 hardware threads per core are used in order to exploit the full computational power of the Intel MIC architecture in-order cores. If one chooses 1 thread per core, one would get a super linear speed-up (scalability above ideal scaling) due to the shared L2-cache, but not the best absolute performance. Hence, because matrix multiplications require a lot of data movement the scalability marginally decreases for more than one thread per core due to the increased amount of moved data.

In terms of absolute performance the Intel MIC architecture is able to affirm the positive impressions from the scalability analysis. By evaluating Figure 55 one can see that TifaMMy's performance on the Intel MIC architecture with 3 threads per core is above the one-core-performance of the tested x86 system. Furthermore the Intel MIC architecture performance is higher than the one for the x86 system for this specific highly data/thread-parallel workload and highly optimized TifaMMY implementation.

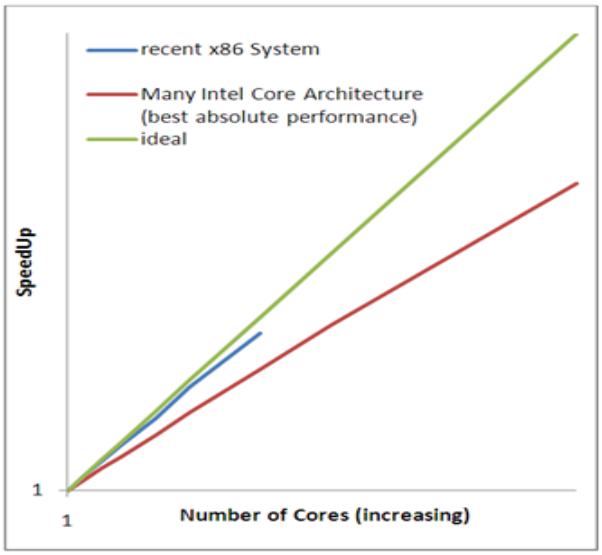

Figure 54: Scalability of TifaMMy on Many Intel Core Architecture and a recent x86 system.

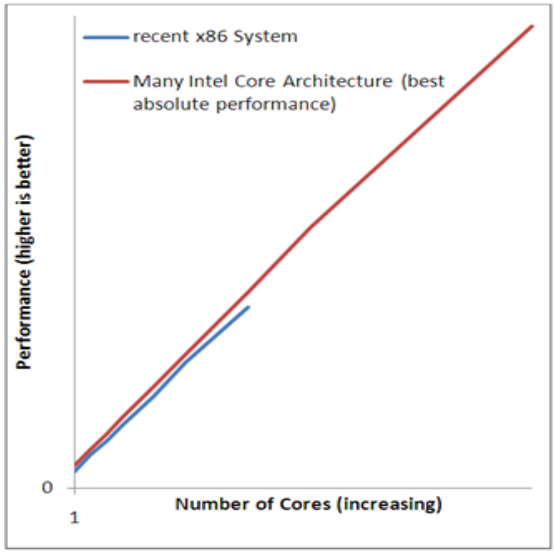

Figure 55: Absolute performance of TifaMMy on Many Intel Core Architecture and a recent x86 system.

3.1.8 *Petapath experiments*

This section describes the tests performed on the Petapath/ClearSpeed accelerated prototype of NCF. The Petapath system was proposed for the evaluation of some large codes on the Petapath Feynman e740 and e780 prototype hardware. Two of these codes were expected to run well on a Petapath system: a many-body astronomy code and a medical imaging code,

respectively and two for which we expected the porting to be difficult: a very large sparse linear solver framework and a geophysical code. In addition we ran, as required, the standard EuroBen kernels that were to be performed on all prototype platforms.

We first discuss the findings with the kernels in order to compare them with the reference performance on the Nehalem-EP platform as discussed in Subsection 3.1.1.

On the Petapath system, we ran the dense matrix-matrix multiplication kernel mod2am on a single processor, called a Multi-Threaded Array Processor (MTAP) of a CSX700 card as that can be regarded as the fundamental computational unit for the Petapath system. Each card harbours two MTAPs and it is easy to engage as many of them as are available in a Petapath device which would be 16 in a Petapath Feynman e780. Multi-card results have already been reported in an earlier deliverable D8.3.1 [2] with a maximum performance of 520 GFlop/s on 16 MTAPs (a full e780). In Figure 56, we show the performance graph for 1 MTAP (theoretical peak performance 48 GFlop/s) over the full range of problem sizes required for this kernel:

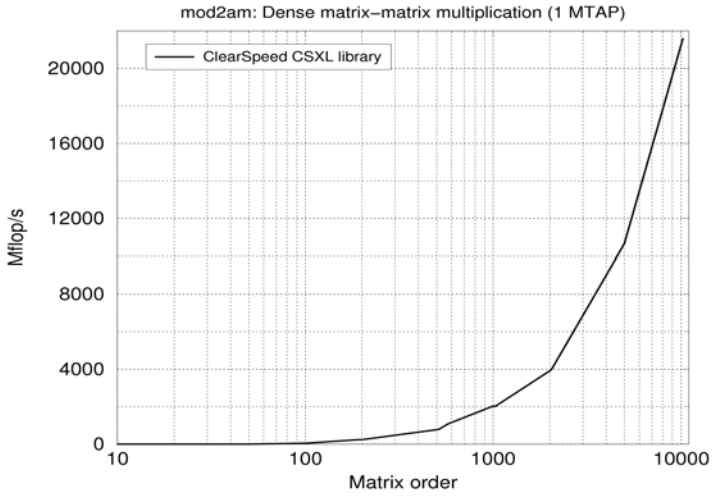

Figure 56: Performance of mod2am on 1 MTAP

As can be seen the speed starts to exceed that of a single Nehalem-EP core for matrix orders > 5000 and the performance curve does not level off for orders as large as 10240. So, one might expect that a significant fraction of the peak speed can be realized for large matrix orders. For small matrix orders one should not use the accelerator as the transport from data to the accelerator and back takes the majority of the run time.

Note that no optimization for the host-assist percentage is done (see Subsection 3.1.6) as this is tied in with a specific matrix order. In real applications where matrices usually will be of a fixed size such optimization could be done and boost the performance significantly.

The processors in the Petapath system are of the streaming type and therefore may be expected to perform poorly on sparse matrix-vector multiplication as implemented in kernels mod2as. This assumption was indeed confirmed as is evident from Figure 57.

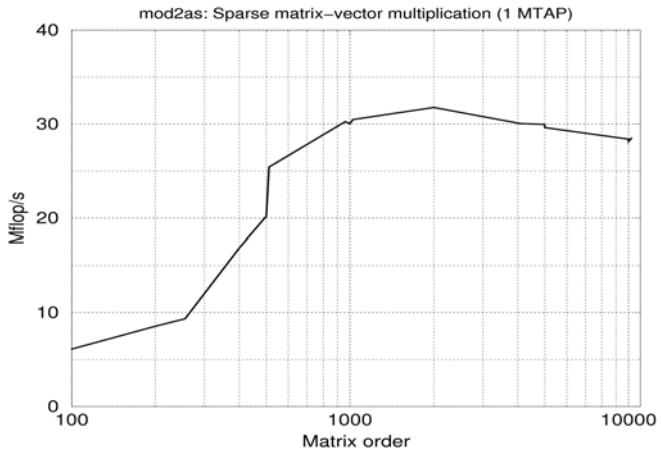

Figure 57: Performance of mod2as on 1MTAP

The performance is poor for two reasons: first there is no support for reduction operations in the CSX700 processor, a property it has in common with GPU accelerators. Second, the memory access is extremely unfavourable with respect to the memory structure of the processor and hence is severely memory bound. The same is true for general CPUs but in these processors the memory structure is much less sensitive to bad memory access patterns. When one compares this result with that of the reference performance of mod2as in Subsection 3.1.1, it is clear that one should avoid using the Petapath hardware for this type of algorithms unless the algorithm itself would be totally restructured.

The FFT kernel mod2f has been run using ClearSpeed's CSDFT library. The 1-D FFT order that can be performed ranges from 256 to 8192, so the range is smaller than the problem set provided for the kernel. Nevertheless, the performance for the orders that are available is quite good as is shown in Figure 58.

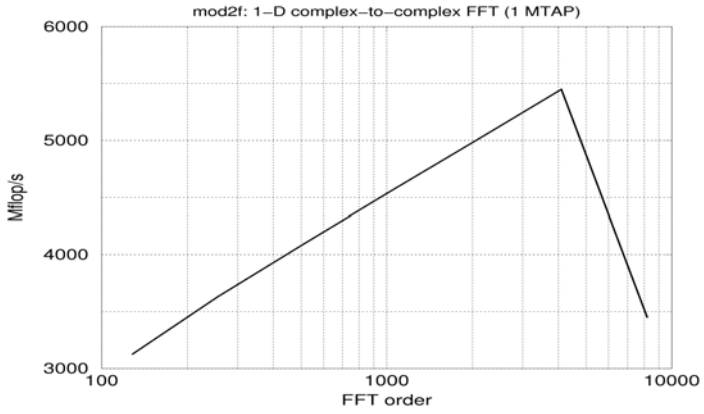

Figure 58: Performance for mod2f on 1MTAP

The performance is better than the reference outcomes everywhere, especially for the smaller orders where it can be faster by a factor of 3-5. For N=8192, the speed degrades to 3.4 *GFlop*/s, still *about* 30 % faster than the reference performance.

For the Random Number generation kernel, mod2h again a library routine from the library CSRNG could be used. For this kernel the results compare well with the single core reference performance reported in Subsection 3.1.1 as is evident from Figure 59.

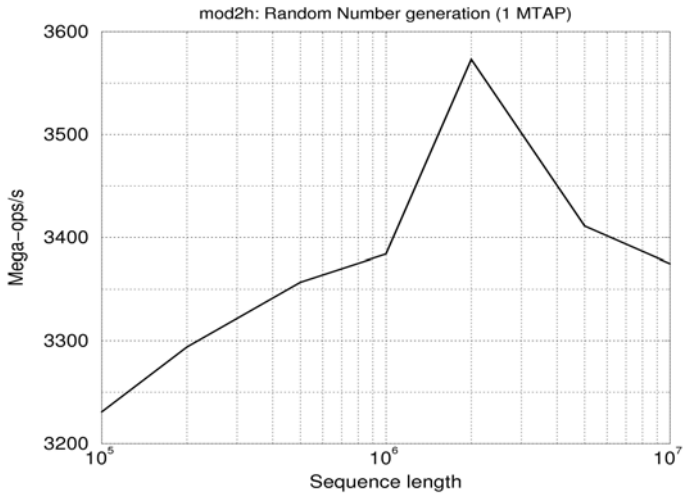

Figure 59: Performance for mod2h on 1 MTAP

One MTAP and one Nehalem-EP processor core deliver more or less the same performance for sequence lengths up to 3,000,000. The reference platform shows roughly a 10 % performance advantage in comparison to 1 MTAP over the whole range of sequences.

As for the applications to be ported, this is still in progress. Due to problems in delivering the hardware both by HP and ClearSpeed, the Petapath system was only installed by mid October 2009. There are preliminary results for the medical imaging application. First experiments have shown that for a reduced-size model the time to solution could be brought from 2.8 to 0.4 sec per convolution (the main computational kernel). It is expected that for the full-sized problem the speed up can be increased by another factor of 4, leading to a speedup of a factor 25 to 30.

3.2 Hybrid Programming Models

3.2.1 *MPI+OpenMP*

Hybrid programming using a mixture of MPI and OpenMP has been studied by WP6 using the EuroBen kernels. Results indicate that the defined reference input data sets (RIDS) are not large enough to show the real potential of hybrid programming. The main focus of the RIDS was to compare single-node CPU performance with performance achieved on today's accelerator hardware, which puts strong limitations on the data set size. In everyday HPC codes a data parallelization is often necessary to tackle the scientific problem, but these data sets are not represented in the RIDS. Figure 60 gives an overview of the achieved performance for mod2am on Nehalem-EP. The mixed MPI+OpenMP code essentially combines an MPI parallelization with calls to the multi-threaded MKL. The pure MKL code was benchmarked on one node with 8 cores. Figure 61 gives performance figures for mod2as. The comparison of both figures reveals that the overhead of the additional MPI parallelization is more noticeable in the compute-bound mod2am. For the memory-bound mod2as a more distributed placement (4×2) is able to outperform the pure MKL version (1×8) since it results in higher bandwidth per thread.

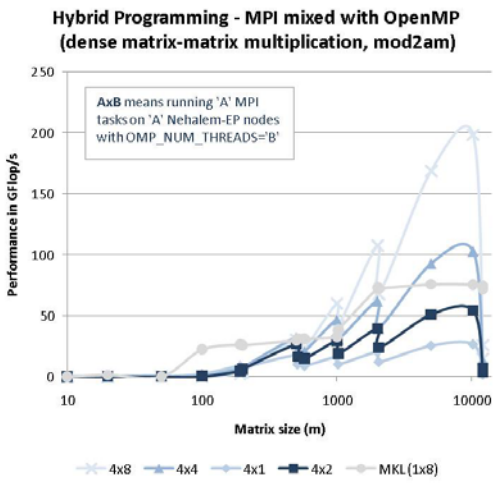

Figure 60: Hybrid MPI+OpenMP results for mod2am

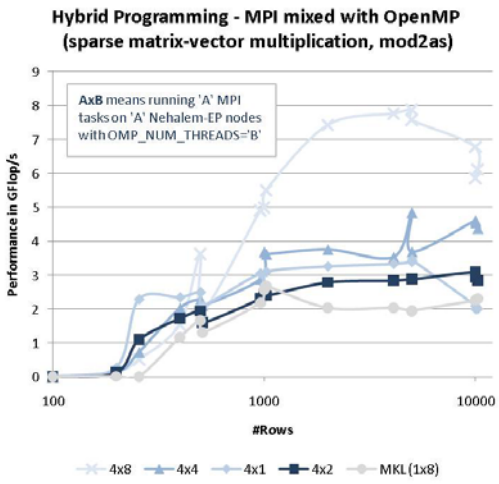

Figure 61: Hybrid MPI+OpenMP results for mod2as

Apart from the results of these small test cases, with current multi-core architectures, the hybrid model has begun to attract attention and seems effective to improve performance of heavy parallel applications. To give an example, Figure 62 shows the scalability of a Linear Algebra subtask of a Car-Parrinello simulation of 256 water molecules (performed with Quantum-Espresso simulation code) on the cluster BCX (CINECA Linux cluster with AMD x86_64 Opteron Dual Core 2.6 GHz). The matrix size is 1024×1024, while the pure MPI version saturates at 128 cores the hybrid version scales up to 512 cores.

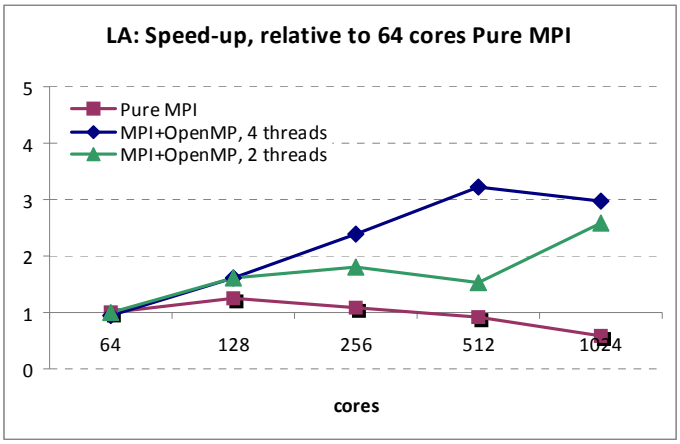

Figure 62: Speedup of the Linear Algebra subtask of a Car-Parrinello simulation

A second example from a study at BAdW-LRZ illustrates the scaling potential of the MPI+OpenMP programming approach on modern multi-core system architectures: Figure 63 and Figure 64 show the strong scaling behaviour of the Berlin Quantum ChromoDynamics program (BQCD) [37] using either a pure MPI or a hybrid MPI+OpenMP parallelization approach. All results are based on $48^3 \times 96$ lattice sizes. Up to 4096 cores, the pure MPI version of the code is faster on both systems. For larger core counts combining OpenMP with MPI results in a substantially better performance. Hence the hybrid programming approach seems to be well suited for future multi-Petascale systems based on multi- or many-core shared memory compute nodes. This example especially shows that the performance advantage of hybrid programming is often revealed only on very high core counts, higher than those of current WP8 prototypes.

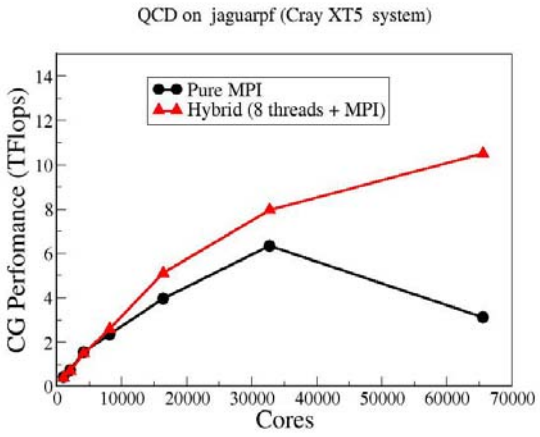

Figure 63: BQCD scaling results on Cray XT5 system

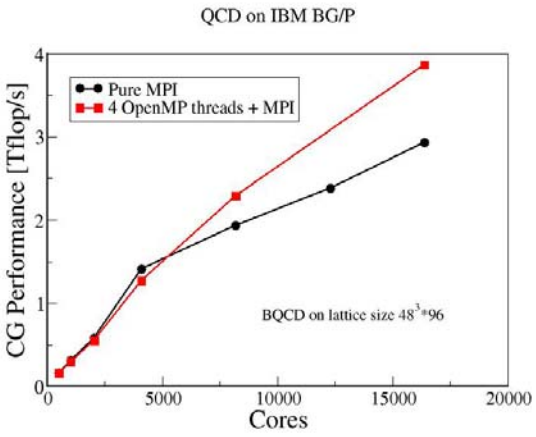

Figure 64: BQCD scaling results on IBM BlueGene/P

3.2.2 *MPI+CUDA*

While single-node performance of GPGPUs has been studied rather extensively, the performance of parallel GPGPU programs has been given relatively little attention so far. However, the parallel performance considerations are quickly becoming more important as GPUs are being introduced to large clusters. CUDA is the prevalent GPGPU programming model and MPI by far the most popular parallelization scheme in HPC today. Thus, it is likely that many HPC GPGPU programs in the near future will use CUDA combined with MPI parallelization and it is very important to investigate its capabilities.

Mod2am:

The mod2am was parallelized and run using different amounts of MPI tasks on the GENCI-CEA GPU cluster. The MPI processes are striped evenly over the GPUs (assigned GPU = process_number modulo GPU_count). The following graphs illustrate the results. N is the node count and P is the process count per node. For example 2N×2P means "two nodes with two processes each". The reference input data set sizes, described in D6.6, were used. In Figure 65, an explicitly coded matrix-multiply routine is used and in Figure 66, the CUBLAS DGEMM routine is called for the serial matrix multiplication within an MPI task.

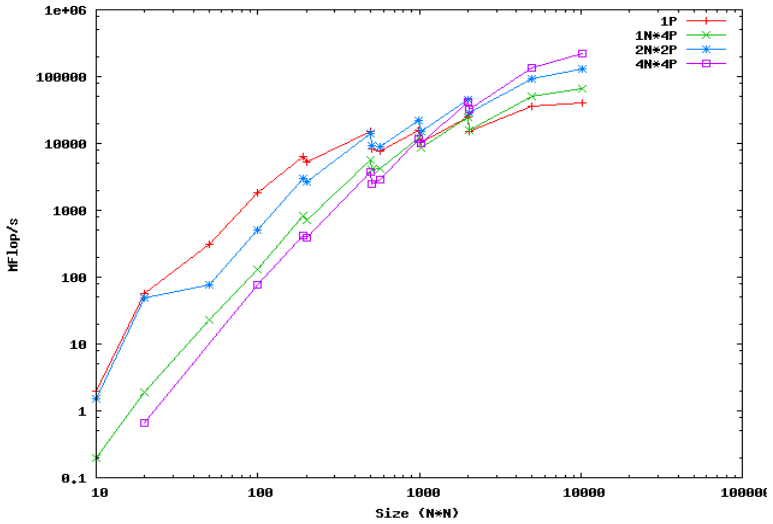

Figure 65: Results of CUDA+MPI mod2am with a reference mxm routine

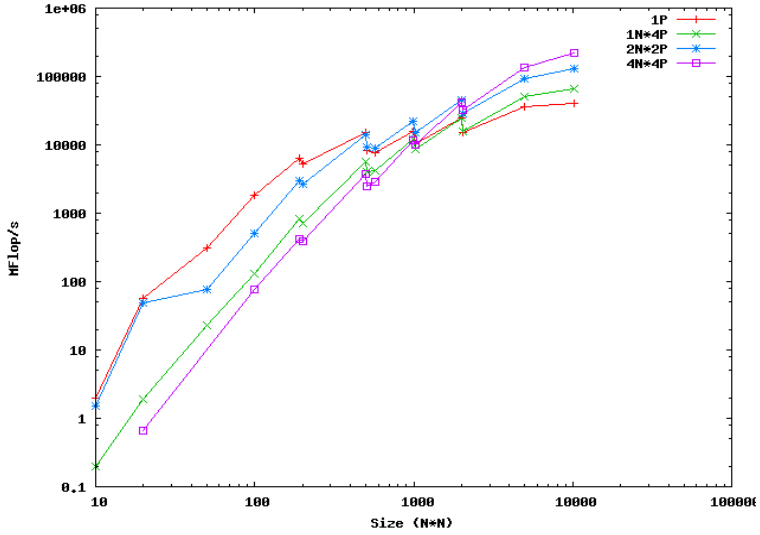

Figure 66: Results of CUDA+MPI mod2am with a CUBLAS mxm routine

It can be observed that for small matrix sizes using a single MPI process and the hand-coded reference matrix-multiplication routine provides the best results. For larger matrices, the best performance is achieved when using CUBLAS and a full set of 4 processes, thus oversubscribing the GPUs with 2 tasks each.

Mod2as:

Figure 67 illustrates the results of the sparse matrix EuroBen kernel (mod2as).

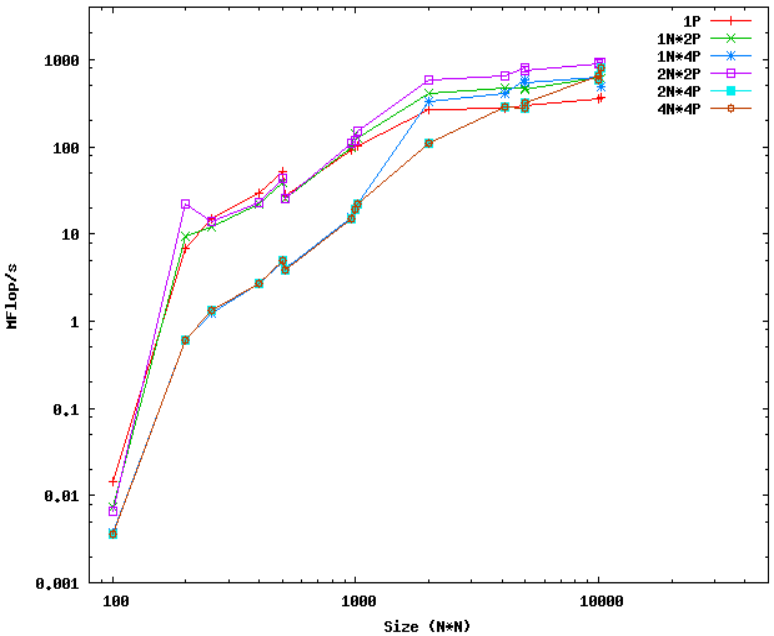

Figure 67: Results of CUDA+MPI mod2as

It can be observed that using a full system with 4 processes and 4 nodes is surprisingly slow. For reasonably sized matrices, using two nodes with two processes seems to yield the best performance.

The parallelization for the tests was done naively by moving all data between GPU and CPU memory every time an MPI communication was performed. This could be done more efficiently transferring only the subset of data needed or using memory pinning. However, using the latter, causes the program to hang, most likely because the DMA access from the GPU is conflicting with InfiniBand RDMA communication. nVIDIA is reportedly working with Mel-

lanox to enable direct communication between GPU memories on remote nodes. This will potentially improve the performance and simplify programming of MPI-parallel CUDA code significantly.

GPU HMMER:

GPU-HMMER is one of the first production-quality GPGPU programs available. GPU-HMMER is a CUDA port of the HMMER open sequence analysis application, used widely in bioinformatics. The original implementation is available from [17] and the GPU-optimized version from [18].

The reference CPU runs were performed on hippu.csc.fi, a HP ProLiant DL785 G5 with 8 Opteron 8360 SE CPUs and an optimized and threaded version of the HMMER code. While there were 32 cores available, the test was run only up to 25 threads as after this, adding new threads degraded the performance.

Figure 68 illustrates the performance difference of CPU vs. GPU implementations of the HMMER code.

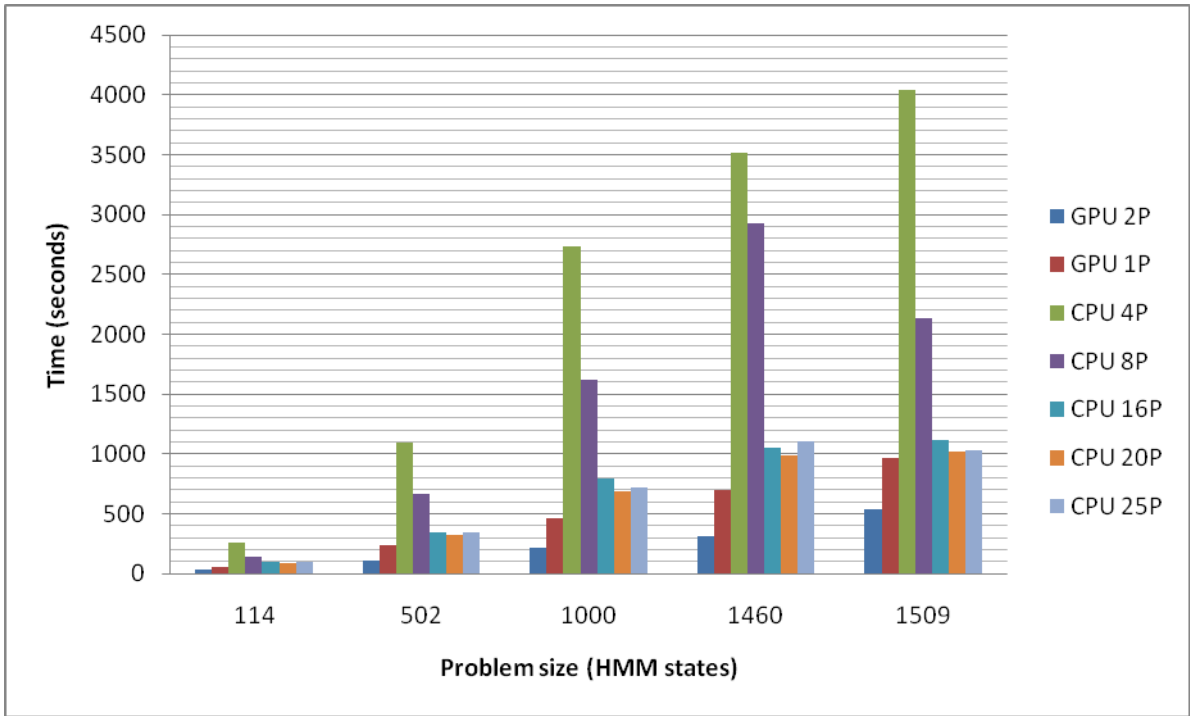

Figure 68: Runtime of CPU vs. GPU HMMER

It can be clearly observed that even when using a single GPU consistently outperforms the CPU implementation running on a powerful SMP system. The problem scales to two GPUs and is faster, by a factor of 1.9-3.2, than the best CPU results, depending on the problem size. The runs were performed without issues. For larger problem sizes, the memory on a single GPU would not be sufficient but using dual GPUs, the maximum problem size can be doubled. Most of the relevant production runs are currently in the domain of 100-2000 HMMs. It can be concluded that GPU-HMMER is a very promising application.

3.2.3 *MPI + CellSs*

Figure 69 shows the performance of mod2am with MPI+CellSs on Maricel. The kernel was run on 8 nodes, with one task per node.

A detailed analysis of the traces does show that the current version does not overlap computation and communication. A 30 % improvement could be obtained without modifying the source code if this overlap was implemented on the run time.

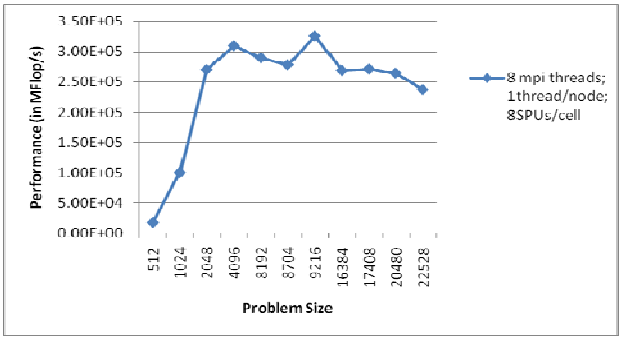

Figure 69: Performance of mod2am with MPI+CellSs on Maricel

3.3 Intra-Node Bandwidth

3.3.1 *Triads (RINF1) benchmark results (BAdW-LRZ)*

Bandwidth and spatial degradation[6] numbers were measured for execution of the vector triad (A=B+C×D) with OpenMP as well as MPI, based on vector lengths between 100 and 16 million, and strides up to 32. Table 7 indicates the results obtained for various platforms:

Platform	Altix ICE Intel Nehalem-EP	Altix4700 Intel Montecito	IBM POWER6	Ultraviolet Intel Nehalem-EX
Sockets per node	2	2	8 or 16	2
Cores per node	8	4	8 (of 32)	16
Peak performance				
Single OpenMP thread (GFlop/s)	2.99	2.31	-	2.59
Scaling efficiency to full node (%)	48	46	-	44
Single MPI task (GFlop/s)	3.26	2.49	2.31	2.91
MPI scaling efficiency to full node (%)	100	100	96	100
Memory Bandwidth				
Single OpenMP thread (GB/s)	9.40	4.81	-	5.97
Scaling efficiency to full node (%)	32	26	-	38
Single MPI task (GB/s)	9.83	4.90	4.74	5.98

[6] The loss of spatial locality due to striding leads to a corresponding loss of compute performance, since only parts of a loaded cache line are used. Within the triads benchmark, this effect is described by the degradation factor, which is defined as the ratio of contiguous to strided performance on an array of the same size. High values denote a correspondingly high loss of performance.

MPI scaling efficiency to full node (%)	30	27	56	38
Spatial degradation factor				
Serial execution from memory	11.5	21.5	20.3	11.8

Table 7: Memory bandwidth and spatial degradation factors of different processor platforms

Note that a dual socket blade was used in the Altix4700 case. For the Altix4700 single socket blade scaling efficiencies for memory bandwidth are by a factor of 2 higher; on the POWER6 node only up to 8 threads or tasks were run and scalability was calculated with respect to that number. Generally, scaled performance is obtained by forming the product of base performance, node size and efficiency.

To also obtain an impression of the cache hierarchies' influence, the following picture gives the impression of the single task performance for various architectures:

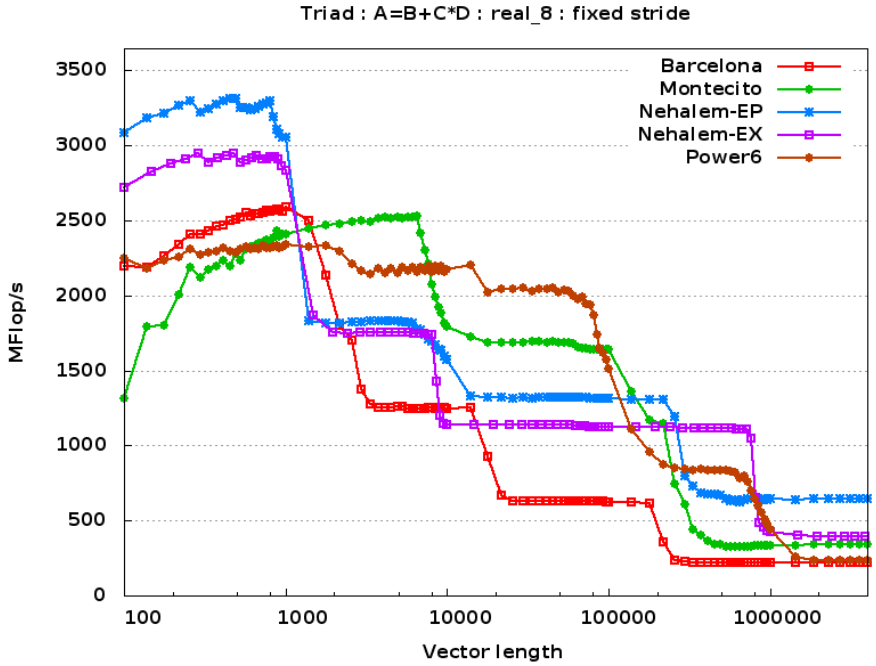

Figure 70: Single task performance of the vector triad measured on different processor architectures

3.3.2 *Random Access Results*

On the SGI systems, Intel compilers and SGI MPT were used to run the program gups_opt. On the IBM system, the parallel environment based on IBM's compiler and MPI were used; on this platform, the "inline" keyword should be removed from the **`update_table()`** subroutine since apparently it is unsupported by the compiler. In all cases, 64-bit compilation and the –DLONG64 macro were used. The size of the problem was chosen such that 2 GB per MPI task were used. The benchmark results are collected in Table 8. Performance numbers are in units of MUPS (Mega-Updates per second).

Tasks	Size parameter	Nehalem-EP (SGI ICE) in MUPS	Montecito (Altix4700) in MUPS	IBM POWER6 in MUPS	Nehalem-EX (Ultraviolet / Standard Server) in MUPS

1	**28**	24.1	6.4	2.1	22.9 / 23.1
2	**29**	42.1	10.2	4.1	33.8 / 36.3
4	**30**	61.3	18.1	7.7	45.0 / 61.6
8	**31**	93.3	32.7	14.4	65.9 / 91.1
16	**32**	165	-	-	118 / 132
32	**33**	251	-	-	202 / 191
64	**34**	416	-	-	318 / -
128	**35**	-	-	-	407 / -
256	**36**	-	-	-	540 / -
Efficiency at 8 Tasks		**48 %**	**63 %**	**85 %**	**36% / 49%**

Table 8: Measured random access memory latencies of different processor architectures

The Nehalem-EX measurements on the Ultraviolet system were performed with various settings (using the Global Register Unit (GRU[7]) for communication between blades, or via shared memory only, and using SGI MPT), but the results only showed minor variations; the results in the table are the overall best ones, using the GRU. According to SGI, a shared memory implementation of the benchmark has achieved up to 800 MUPS on the Ultraviolet. On the Intel standard server, Intel MPI 3.2 was used to perform the measurements, which except for complete machine filling (32 cores) gave better results than the Ultraviolet system, partially comparable to Nehalem-EP on a per-core basis.

3.3.3 *Host to accelerator bandwidth (NCF)*

One of the most serious bottlenecks limiting the applicability of accelerator technologies today is the host to accelerator bandwidth available in present PCIe Gen 2.0 or PCIe Gen 1.1 based device implementations. It is therefore very important to test the effective bandwidth from the host node to the accelerator and *vice versa*. The measurements were performed at NCF using the ClearSpeed/Petapath prototype using one Multi-Threaded Array Processor (MTAP) of a CSX700 card. In hindsight the message lengths used were somewhat small to do full justice to the data transfer library CSPX. Ongoing experiments indicate that the non-optimised results shown here can much be improved on. The non-optimised, 1 MTAP results are shown below:

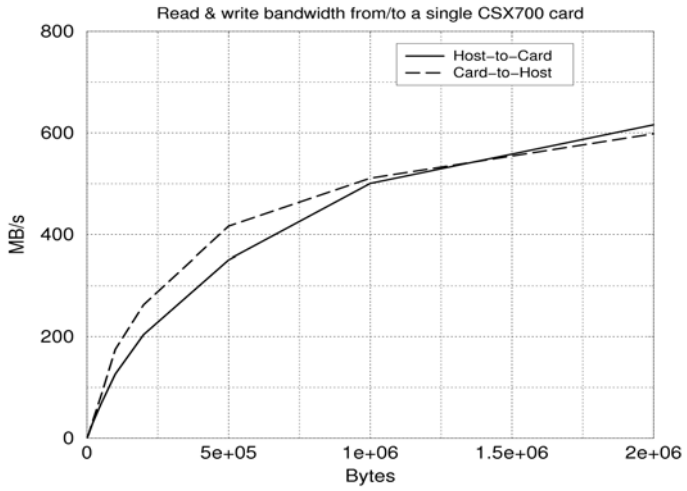

Figure 71: Measured host-card bandwidth 1 MTAP

[7] The Global Register Unit (GRU) that extends cache-coherency from the blade to the entire NUMAflex environment and provides other memory related functionality.

The peak bandwidth quoted to a CSX700 card is 2 GB/s per direction (i.e. PCIe x8 Gen. 1.1). About 35 % of this bandwidth can be attained for on- and off-card traffic. For small messages the transfer speed is lower because of the influence of the latency that occurs because of the software initialisation: 667 µs for host-card traffic and 640 µs for card-host traffic. For the Feynman e740 and e780 that feature a PCIe Gen.2.0 ×16 connection of 8 GB/s per direction, the measured bandwidth shown in Figure 71. The 8 GB per second per direction (full duplex) are shared by the 4 (resp. 8) e710 cards, each one bounded by the 2 GB per second per direction full duplex PCIe ×8 Gen 1.1.

3.4 Inter Node Communication Network

3.4.1 *eQPACE*

One challenge of porting application code to QPACE is related to the PowerXCell 8i architecture, which is quite different from commonly used x86 and PowerPC processors. The difficult part is the implementation of the application on the SPE. The programmer has to take care of data management, i.e., allocation of Local Store (LS) space and explicit DMA get and put operations to load data from or store data to main memory.

Optimizing QCD application code on QPACE requires [38]:

- With essentially all lattice QCD kernels being memory bandwidth limited, it is mandatory to reduce the number of memory accesses to an absolute minimum and to optimize re-use of data in the on-board LS;
- When parallelizing the application communication has to be organized in such a way that the network latencies of $O(10^4)$ clock cycles are hidden;
- Efficient use of the SPEs floating-point pipeline, i.e., SIMDisation of the code.

To measure the performance of the torus network a ping-pong test was used to estimate the latency. One packet of 128 Bytes was sent from the LS of node A to the LS of an adjacent node B and back to A. The latency was estimated as the round-trip time divided by 2, which was found to be 3 µs.

To measure the bandwidth, node A sends a message to B and vice versa; multiple messages can be sent concurrently. The bandwidth depends on the message size and the number of concurrent communications. With a sufficient number of packets on the fly, it is possible to get close to the theoretical maximum bandwidth of 900 MB/s.

The simulation of the motion of a string (solving a one-dimensional wave equation) was the first (tutorial) application that was ported from a Cell cluster version to the QPACE SPE-centric programming model.

Another non-QCD application, which was implemented on QPACE, was the multigrid algorithm [42]. In this implementation the SPE-centric programming model had been chosen. The problem to solve was the Poisson problem on a regular three dimensional grid. For a better mapping of the grid to the computing node the torus network was extended to a bigger virtual torus network with the SPEs of one Cell processor forming a small Cartesian network themselves.

The discretization of the Laplace operator was done with a simple seven point stencil. This leads to a communication scheme, which involves only neighbour to neighbour communication on fine grids. So this fits very well to the torus network. But on coarser grids with less grid points than computing nodes, messages have to be routed through the nodes, which are not computing ones. This was implemented in software. That increases the network latencies significantly. One could decrease this impact by using hardware routing. On QPACE this

could be done by another FPGA image. Some considerations of this implementation are presented later in this document.

A simple stencil code, like the one used in this implementation, is memory bandwidth bounded on the Cell processor, when choosing big enough domains. But the performance of this implementation shouldn't be bounded by memory bandwidth, because the properties of the network should be investigated. So the domain size was limited to fitting into the local stores of the SPEs. That leads to a domain of 16×16×16 per SPE. With this size the performance is bounded by the network throughput. It was discovered that the most limiting factor is the number of messages one can send in a specific time span independent of the message size (750 messages per ms). This is the high penalty on the coarse grids, where only 2×2×2 messages are used.

The examination of the timings in the FPGA showed that most of the latency time is caused by the interface of the FPGA to the Cell processor. This time is the five sixth of the whole latency. The bandwidth of this link also limits the overall performance of the network. When a node just has to forward a message, the message has to cross this interface twice. To avoid this, the forwarding could be done on the FPGA. That would also take load from the Cell processor.

To be able to perform this forwarding the address must be added to the message header. A relative addressing according to the torus structure is reasonable. A node sends the message to a neighbouring node, which is closer to the destination and corrects the relative address. When a node receives a message with relative address zero, it transfers the message to its cell processor. Additional buffers between the links are needed to make possible the serialization of the messages of the different links and the Cell processor possible. These considerations were not implemented, because there was no more space left on the FPGA.

The prerequisite for the High Performance LINPACK (HPL) [21] benchmark is an efficient MPI implementation for message passing between compute nodes. This also requires a low-level communication interface to access the QPACE torus network. Due to the design of the HPL for the Cell processor (QS22 patch), which uses the 8 SPEs as accelerators, it was decided to use a PPE-centric communication model where the PPE is responsible for sending and receiving messages.

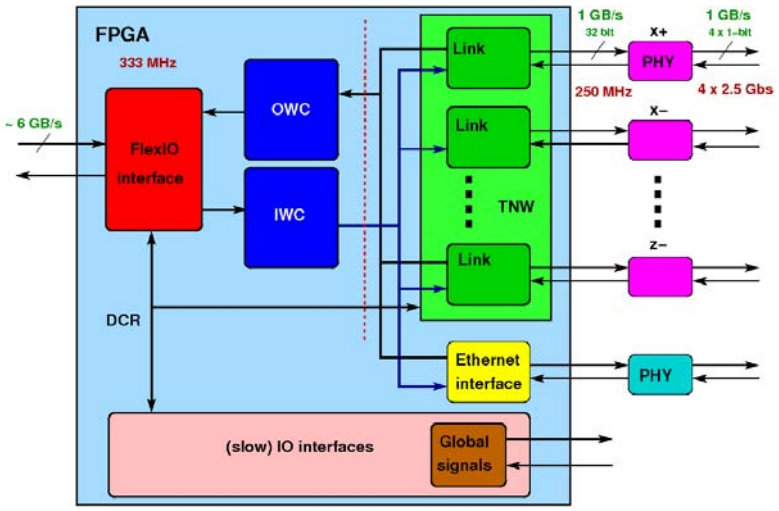

Figure 72: QPACE network processor

HPL has communication requirements that differ from QCD, like efficient transfer of large messages and collective operations, which demand extensions to the network processor design as well as to the software stack. For that reason, the FPGA based network processor was

modified in order to support direct data transfers between main memory and torus network. This modification (inserting a DMA engine in place of the Inbound Write Controller IWC) is the basis for a low-level torus communication API that (only) supports main memory to main memory data transfers over the torus network. The alternative approach of using the Memory Flow Controller (MFC) of an SPE instead of modifying the FPGA image has not (yet) been implemented.

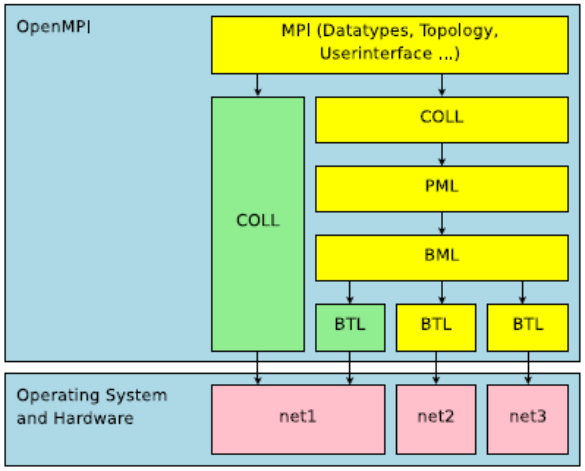

Figure 73: Open MPI Modular Component Architecture

A new Open MPI Byte Transfer Level (BTL) component, called tnw, was developed for MPI message transfers over the torus network which uses the low-level torus API. Some issues caused by the implementation of the network processor had to be taken into consideration when using the DMA engine regarding alignment of send and receive buffers and the message size which has to be a multiple of 128 Bytes. For the latter case a workaround was implemented by using the pipeline protocol which allows for splitting a message transfer in two parts and sending the first part via copy in/out while the remaining part meets the size constraint.

Another limitation is that the network processor needs the physical address of the send/receive buffer which must be contiguous and locked in main memory. Here, the workaround is a separate contiguous region of physical memory which is reserved by the kernel and access to this area is provided by an additional kernel module. Then, memory buffers that are to be used for MPI communication operations must be allocated by a specific memory allocator.

Using the tnw component it is possible to send messages via MPI between neighbouring nodes over the torus network. If a process cannot be reached over the torus the slower Gigabit Ethernet is used which does not require any modifications. In order to support torus communication between all processes, software or the network processor could implement a routing mechanism. Here, the latter approach seems to be more efficient due to less memory copy operations. However, a limitation might be the remaining resources available in the FPGA based network processor. So far, routing facilities for QPACE have not been investigated.

Besides the send/recv operations in the tnw BTL component also MPI collectives used in HPL were optimized. A new Open MPI COLL (Collective) component, called tnw_tuned, was developed. This component implements the collectives MPI_Allgatherv(), MPI_Scatterv(), MPI_Bcast() and MPI_Allreduce(). These operations were adapted with the focus on torus communication and improved the performance of HPC significantly.

The HPL result for two racks of the QPACE cluster on 512 nodes: 43.01 TFlop/s (55.71 TFlop/s peak) and 723 MFlop/s/W. This is currently the best performance per Watt ratio in the Green500 list [39].

Measuring the performance of the PPE-centric communication over the torus network (communicate messages from main memory to main memory) the ping-pong benchmark produced the following results:

- Latency of 4.7 µs for 128 Bytes message;
- Bandwidth of 845 MB/s for 64 KB message.

The development of the Open MPI subset for QPACE is still going on and it can be expected that the performance values for HPL can be augmented. To what extent experiences gained so far can be used with other applications is currently investigated by the implementation of a multigrid method on QPACE.

Summary

- A 4-rack QPACE system was installed but the system is still under development. Especially the network processor requires special attention. Due to the limited facilities for debugging errors in the FPGA, several low-level test cases have to be designed to reveal or reproduce an error which makes it difficult to figure out the source of a malfunction.
- Low-level benchmarks and tutorial applications were run.
- Different versions of FPGA loads have been used.
- A subset of MPI functions was implemented for the torus network.
- An efficient QPACE version of the High Performance LINPACK benchmark was developed (basis for Top500/Green500 ranking).

3.4.2 *BAdW-LRZ*

Intel MPI benchmark results

Point-to-point measurements had been used to obtain (bi-directional) link bandwidth and latencies (for a single task pair between two nodes), as well as the bisection bandwidth and latency (aggregate over all nodes). The MPI_Sendrecv() call is typically the best-optimised call. This call is therefore used for the measurements reported in Table 9. The bandwidth measured in Table 9 is for Send and Receive together. The MPI implementation used for all three systems is the message-passing toolkit (MPT) from SGI.

Platform	Altix ICE	Altix4700	UltraViolet
Number of nodes used (divide by two for node pairs)	32	16	16
Link Bandwidth (GB/s)	3.66	1.90	7.20
Bisection bandwidth (GB/s)	57.9	13.4	34.3
Efficiency (%)	98.7	88.1	78.8
Link Latency (µs)	1.7	2.6	1.60
Bisection Latency (µs)	1.7	2.7	3.0

Table 9: MPI send/receive benchmark results

Here interconnect efficiency is considered relative to the number of **node pairs** used. The aggregate bisection bandwidth achieved on the UltraViolet system is at present only approximately a third of the hardware bandwidth; the values measured with the GRU (displayed here)

are between 5 and 20 % better than without it. Note that the UltraViolet system was run in a routerless configuration with a node paired torus topology; larger systems must be equipped with routers.

The measured bisection bandwidth and latency curves for the various platforms are displayed in Figure 74. The per-node-pair bandwidth is plotted with an indication how many node pairs were active provided for each curve.

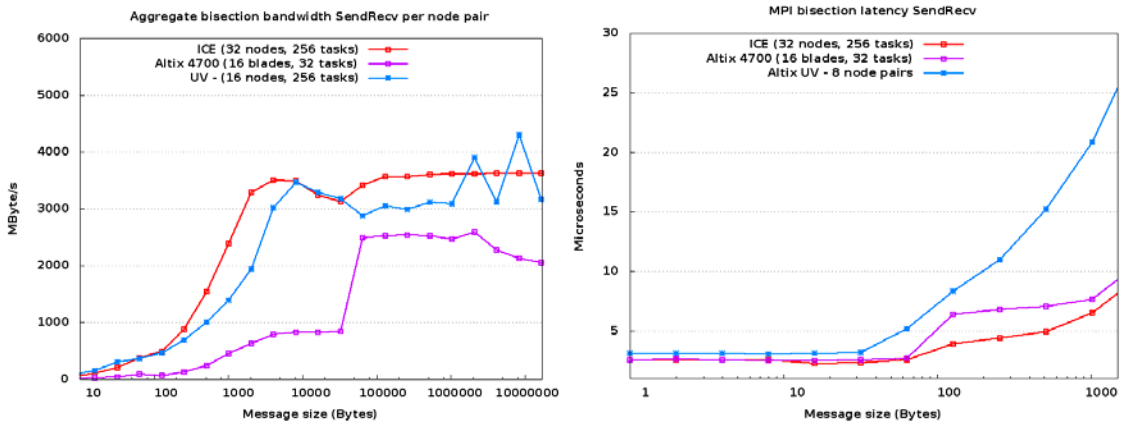

Figure 74: MPI bisection bandwidths (left figure) and MPI send receive latencies (right figure)

For collective communication, the execution time for the MPI_Allreduce() function as a function of message buffer size is illustrated for the various platforms in Figure 75 remapping illustrates the performance losses to be expected if the message pattern invokes blocking or non-optimal routes in the interconnect (see Subection 3.1.6). Note that on all platforms, 256 MPI tasks were started for the reduction. The UV values, due to a still present hardware or MPI implementation problem, are here measured using shared memory, without the GRU offload; for small message size still an improvement of a factor of 3 is seen compared with the other systems, although further tuning of the MPI implementation still appears to be pending, as can be seen from the message size 128 Byte outlier.

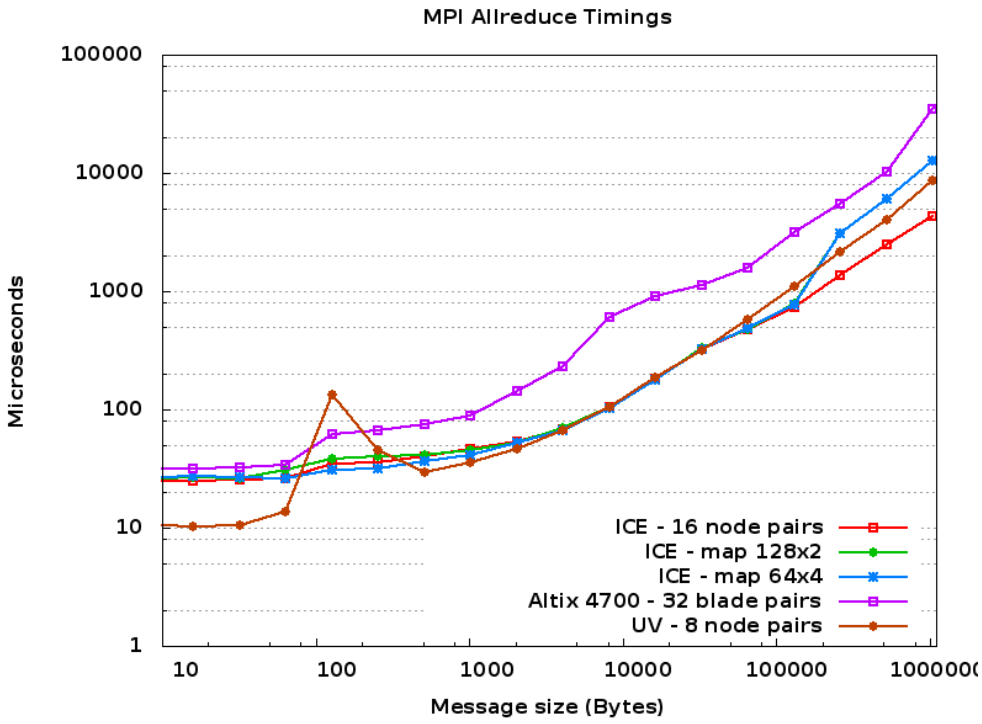

Figure 75: MPI Allreduce timings for SGI Altix4700, ICE and UltraViolet

Alternative MPI implementations and interconnect pruning on ICE

In order to investigate implementation-dependent effects as well as the effect of providing only partial bandwidth in the interconnect hardware, the following measurements were performed:

- Bisection and collective performance using Intel MPI 3.2, comparing with SGI's MPT (version 1.24);
- Bisection and collective performance with half of the IB-uplink switch ports shown in Figure 4 disabled (only 8×3 instead of 16×3 active IB uplinks to the central Infiniband switch as shown in Figure 4).

Figure 76 illustrates the bisection bandwidth using different MPI implementations with a pruned (blocking factor 2) and unpruned fat tree network topology (using 256 MPI tasks with a 128×2 mapping):

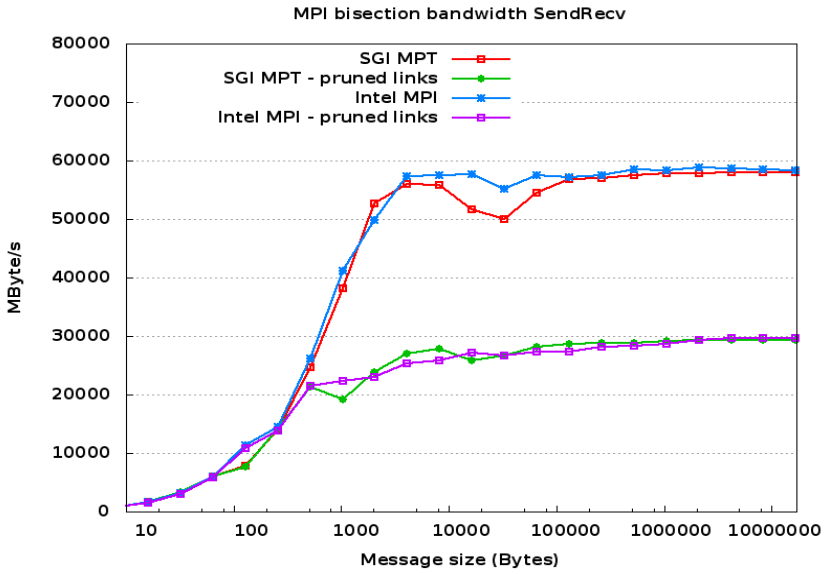

Figure 76: Measured Altix ICE MPI bisection bandwidths

Generally, bisection bandwidth halves as expected; Intel MPI seems to have a slight performance edge over SGI MPT here. Note that for SGI, a setting of the environment variables MPI_DSM_DISTRIBUTE=1 and MPI_BUFFER_MAX=32768 was used. The latencies are more instructive for short message lengths, again showing slight advantages in favour of Intel's implementation, but no increase in latency for the pruned case (see Figure 77):

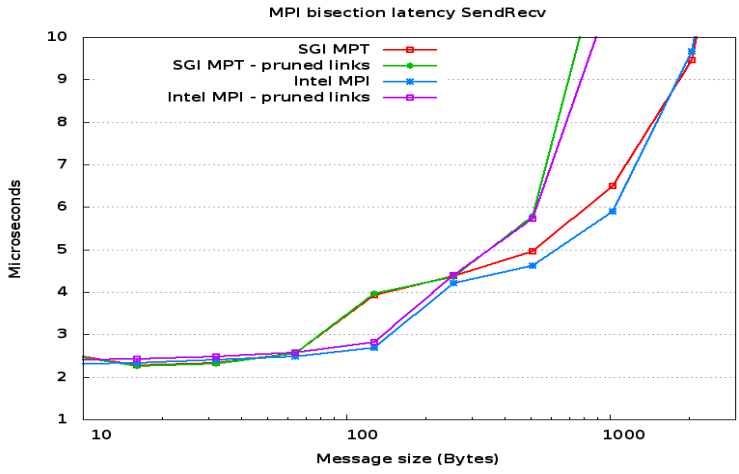

Figure 77: Measured Altix ICE MPI bisection latencies

The following collective MPI calls were investigated: MPI_Allreduce(), MPI_Alltoall() and MPI_Barrier(); the second of these is probably the most resource-intensive call in the MPI API. The following SGI MPT settings were used to pin MPI tasks as well to minimize re-transmissions of messages: MPI_DSM_DISTRIBUTE=1, MPI_BUFFER_MAX=2000, MPI_BUFS_PER_HOST=256. For Intel MPI, only the setting I_MPI_PIN=yes was used. 256 tasks were executed on 32 nodes. For the Barrier call, the following table displays the measured latencies:

Implementation / mode	Barrier Latency (µs)
SGI MPT / full	23
SGI MPT / pruned	23
Intel MPI / full	14
Intel MPI / pruned	14

Table 10: Measured barrier latencies

The run-to-run error for the above numbers is approximately 1 µs; no difference is observed between pruned and full switch configuration. Note that the UV barrier latency measured for 256 MPI tasks, using SGI MPT, is 4.9 µs.

Figure 78 illustrates the effects for the case of the MPI_Allreduce() call:

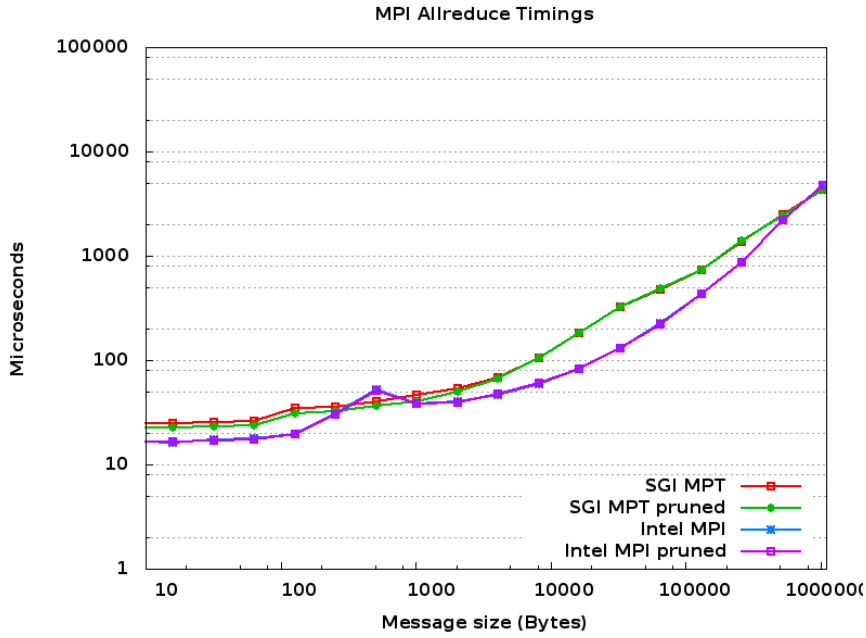

Figure 78: Measured MPI Allreduce timings

Generally, the pruning leads to no significant reduction of performance for this collective call (see Figure 78); the observed differences are within the statistical fluctuations observed from run to run without changing the hardware configuration. Much more interesting is the difference of a factor of two between the implementations by SGI vs. Intel, respectively. According to Intel, a lot of optimization work has gone into optimizing collective calls, in its most recent implementation, giving it a performance edge against SGI's implementation in this particular case.

For the Alltoall call shown in Figure 79, the Intel MPI again shows consistently better timings. The differences between pruned and fully connected network are more marked, but generally still less than between implementations for moderate message sizes. For large message sizes (1 MB and beyond) the effect of pruning appears to become more dominating.

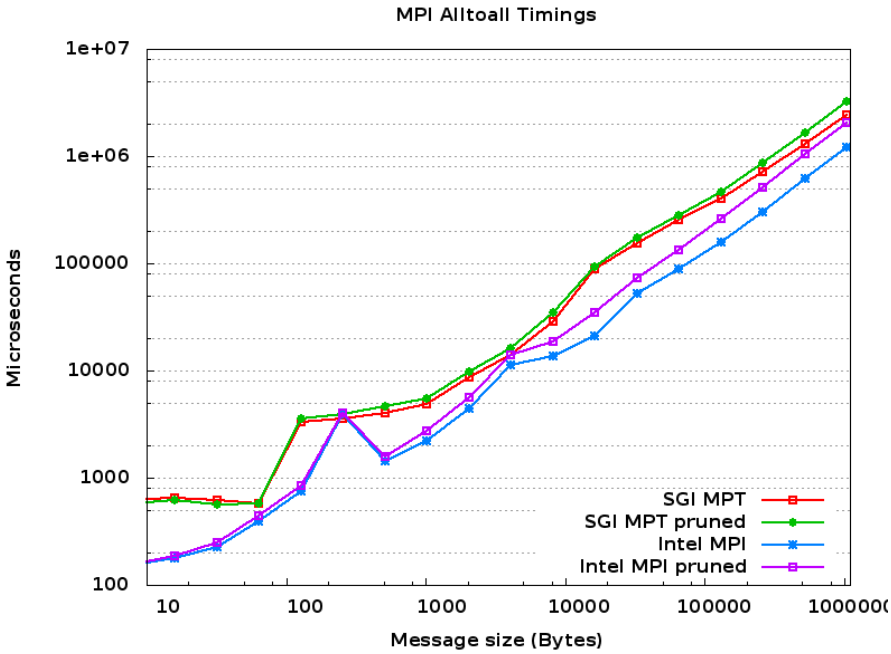

Figure 79: Measured MPI Alltoall timings

For network latency bound applications like GADGET leaf-side[8] network pruning could be a very cost-effective measure to separate MPI traffic from I/O within the IB fabric. Table 11 shows the MPI version used has much more effect on the measured GADGET run times than leaf-side network pruning. A test case of 32,768,000 particles was considered.

MPI-Tasks	MPT Time (s)	OpenMPI 1.3.2 Time (s)	MPI Intel_PRUNED Time (s)	MPI Intel_FULL_Interconnect Time (s)
64	40.81	-	42.05	41.82
128	24.47	57.34	25.08	24.09
256	13.45	29.56	17.57	16.67

Table 11: Influence of different MPI versions and network pruning on execution time of GADGET

UFM deployment at LRZ

The Infiniband Architecture (IBA) standard defines a "Subnet Manager" (SM) entity that is responsible for configuring and managing the switches and routing tables. Many existing SM algorithms today are assigning routes statically without any knowledge of the actual traffic characteristics while real-life traffic is spread un-equally, leading to traffic bottlenecks and congestion. Effective bandwidth may be reduced to 50-60 %, or even down to 10 % per stream, and cause the latency to increase significantly.

In close collaboration with Voltaire and SGI, LRZ has started evaluating Voltaire's Unified Fabric Manager™ (UFM™) which promises to provide important features for large Infiniband networks such as performance and congestion monitoring, performance optimization, device management and troubleshooting. So far the evaluation focus was put on a UFM feature called Traffic Aware Routing Algorithm (TARA). TARA is an intelligent routing algorithm that optimizes the routing within the IB fabric and therefore should be able to provide better communication performance.

[8] The term leaf-side pruning is used for fat tree topologies in which the connection of the first level switches (compute nodes are connected to first level switches) to the second level switch hierarchy is realized with a reduced number of network links (pruned fabrics).

To demonstrate the performance improvement achieved with UFM, multiple Intel MPI benchmarking jobs were run on the SGI ICE system comprised of 48 nodes. The scenarios run include multiple jobs comprised of 8 or 12 nodes, using various task placements across the leaf switches of the fabric. Scenarios were run with and without TARA to be activated.

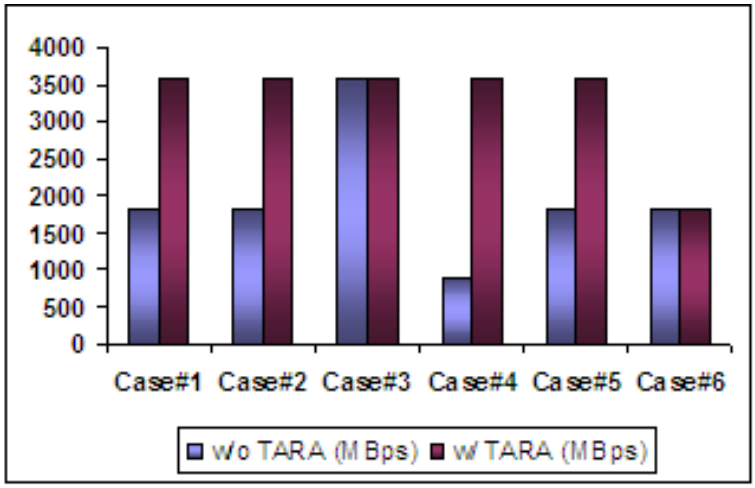

Figure 80: Measured MPI send/receive link bandwidths with and without traffic aware routing

Figure 80 shows, the measured MPI send/receive link bandwidths with and without traffic aware routing for various task placements across the leaf switches of the fabric. By means of traffic aware routing even for this small system the average performance improvement was 100 %. Hence the use of intelligent network routing algorithms will be of major importance for future Petascale systems to reduce network costs and at the same time ensure high application performance due to better utilisation of the network fabric.

3.5 PGAS Languages

3.5.1 *Chapel experiments*

Chapel is a new PGAS-like parallel programming language developed by Cray and the University of Washington within the HPCS project (a DARPA programme). Chapel has been designed to increase programmer productivity by abstracting the underlying hardware in order to lighten the burden of explicit data communication and synchronization for the programmer. Unlike CaF and UPC, Chapel does offer semantic language support not only for data parallelism, but also for task parallelism and is therefore sometimes classified as asynchronous PGAS (APGAS) language. The Chapel compiler and run-time environment are still under heavy development and must be considered work in progress with prototype character.

One of the key concepts of the language is a *locale*, meaning a hardware execution environment with uniform access to memory as for instance a compute node of one or more individual multi-core processors. Chapel is meant to exploit parallelism on a single *locale* through multiple concurrent execution threads much like OpenMP, however without requiring fine-grained control as in the case of OpenMP. Chapel also allows parallel concurrent execution across a number of locales, for instance a cluster of compute nodes. Data distribution and work sharing across different nodes are semantically linked to the language concepts of *domains*, essentially a set of indices of an array, and its associated *distribution* which determine how the elements of a domain are distributed across different *locales.*

In Chapel, data parallel execution on a single *locale* or across multiple locales is invoked by using the *forall* loop over all elements of a *domain*. The associated *distribution* then is responsible for remote data access and synchronization if necessary in order to execute the loop body in parallel.

The standard library of Chapel offers a range of *distributions* for blocked, cyclic, and block-cyclic data distribution, as well as special *distribution* for sparse domains, as for instance sparse matrices. However, in the previously studied version V0.9 of Chapel the actual implementation of these standard modules did often not exploit parallelism but silently was executed sequentially on a single *locale*.

During the lifetime of the PRACE project, the Chapel team has released a number of different versions of the compiler, starting from V0.9 to V1.1 in April 2010. The initial evaluation of Chapel reported in D6.6 was conducted with version V0.9. The subsequently published versions V1.0 and in particular V1.02 addressed some of the issues and shortcomings relevant for the PRACE implementation of the EuroBen kernels. The latest version V1.1 has not been investigated in detail yet as it was published only a couple of week before submission of this document.

In general scalar performance of V1.02 has increased significantly to the point that matrix vector multiplication execution time is in the same order of magnitude as similarly non-optimized C code, if the Chapel compiler is instructed to produce non-distributed, serial code. While most distributions in V0.9 were limited to executing parallel forall loops sequentially on a single locale, all relevant distributions in V1.02 support local-locale, multi-threaded parallel loop execution, and most support parallel execution on multiple locales.

According to the documentation of the Chapel release V1.1, all relevant distributions are fully multi-threaded and multi-locale, i.e. support parallel execution within a single locale as well as across multiple locales. In this sense, Chapel distributions are feature-complete. The examples packaged with the Chapel distribution - as for instance FFT, STREAM and, HPL - mostly show clearly improved performance and scalability. However, first tests indicate that performance and scalability of the EuroBen kernels are still lacking und unsatisfactorily. A preliminary analysis seems to indicate, that the biggest issue is the fact that remote memory accesses, for instance to elements of a matrix stored on a remote locale, are *not aggregated*, but rather *done element-by-element*. Due to the large latency this results in poor performance and does not allow scaling with the number of locales.

The Chapel developers have confirmed through private communication, that aggregation of remote memory access will be implemented soon as an optimization by the compiler. They have also given some hints, how the situation might be alleviated today through low-level Chapel programming and exploiting non-public interfaces. Those hand-coded optimizations have, however, not been implemented or studied before the submission of this document and will be reported in the future as part of PRACE-1IP.

Chapel continues to be an interesting concept for future parallel programming models, but much more work needs to be done on the compiler and runtime environment in order to be a real alternative for production environments. In particular, the technology preview V1.02 and later version reveals some very interesting features like support for GPGPUs through dedicated *distributions*, as well as the new language concept of *realm* which is meant to abstract different hardware platforms and will eventually allow to run Chapel on hybrid systems consisting, for instance, of PC clusters and vector machines, or similar.

3.5.2 *Co-Array Fortran experiments*

The usability evaluation of Co-Array Fortran (CAF) compiler is performed on a Cray XT5 system, which is the latest Cray XT series platform. Recently, the CSCS XT5 system has been upgraded to a hex-core AMD processor called Istanbul from a quad-core Shanghai processor. The resulting system now has a peak performance of 212 TFlop/s, upgraded from 141 TFlop/s. The XT5 system is composed of 3,688 hex-core Opteron processors with a 2.4 GHz clock frequency and now has 1.3 GB memory per core, down from 2 GB per core in the quad-

core configuration. The XT5 system offers a number of enhanced features as compared to its predecessor XT systems including a new generation of high-bandwidth custom interconnect chip called SeaStar2.2, a dual-socket quad-core Opteron processing nodes and computing node Linux environment called CLE (Cray Linux Environment). The X2 processing node is a 4-way SMP node sharing a global address space (the CPU is a 1.4 GHz vector processors with 8 vector pipes). Processing nodes of the two target systems are shown in Figure 81. The memory bandwidth of the X2 system is considerably higher. On the XT5 system, the aggregate memory bandwidth is 25.6 GB/s while on the X2 system there are four 28.5 GB/s channels with two levels of cache. The XT5 processor has three cache levels and access to memory is not uniform unlike the X2 node. The network interface is also different on the two systems. The Cray XT5 system has a Cray proprietary SeaStar2.2 network interface card and the nodes are connected in a three-dimensional mesh topology. The X2 system on the other hand has Cray proprietary YARC router chip connecting nodes in a fat tree topology.

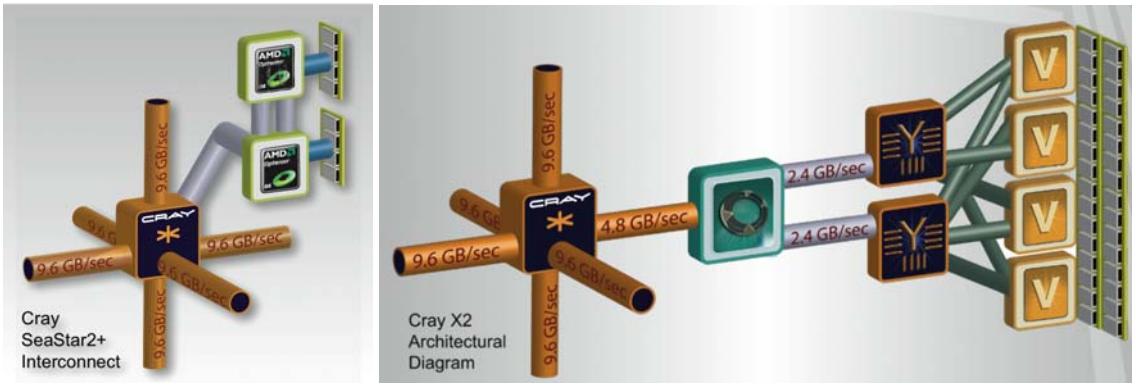

Figure 81: Cray XT5 (left) and Cray X2 (right) processing nodes

The programming environments also have distinct features, for example, the Cray XT5 system runs Cray Linux Environment (CLE), while the X2 system has a different variant of Linux. Although the Cray Compiler Environment (CCE) is used on both platforms, on the XT5 system version 7.2.x was available while on the X2 system version 6.0.0.x of C and Fortran compilers were available. Traditionally, third-party compilers such as GNU, PGI and PathScale have been supported on the XT series platforms.

The main objective of the CSCS task was to evaluate the usability of the compiler for code development and execution for representative benchmarks. However in future , the CSCS team plans to evaluate the usability of the compiler for production-level code development, execution, debugging, performance evaluation and portability across multiple platforms.

In order to evaluate the CCE compiler framework for the target PGAS languages, the CSCS team focused on the components of the CCE framework that a user typically interacts with during code development and execution. This includes the front-end compiler interface for a large-scale production-level Cray XT5 system, the error reporting mechanisms, back-end code generation on a cross-compiled system and execution in a batch environment. The CSCS team also tested the completeness of the CAF and UPC compilers by running some conformance test cases. Note that the CCE CAF compiler is expected to comply with the proposed FORTRAN 2008 standard and the UPC compiler should conform with the UPC version 1.2 specifications.

Currently the GASNet library serves as the communication interface for both CAF and UPC compilers on the Cray XT5 platform with SeaStar2.2 network chips. GASNet enables remote memory access (put and get) operations on shared and distributed memory platforms for PGAS languages. It is implemented using the Cray Portals communication library on the XT5 platform. On the future Cray network chips, a high performance communication library is

expected to replace GASNet, which currently supports the functional implementation of the target PGAS compilers.

Fully functional compilers for the CAF and UPC languages are available on the CSCS Cray XT5 system as the part of the CCE framework. The results included in the document are collected using the latest Cray compiler version, 7.1.2, which are compared and contrasted with the results presented in an earlier report using version 7.1.1 on the Shanghai based processing nodes. After loading the Cray Programming Environment using a simple module load command, the CAF and UPC compilers can easily be invoked using a flag (`-h upc` or `–h caf`) along with the regular compiler command line options. At runtime, the number of images and threads can then be specified in the same manner as a user specifies the number of MPI tasks on command line. Table 12 shows the current and previous XT5 test systems configurations and future plans. Cray X2 details have been provided earlier.

	Previous (last report)	Current	Future
Processing Node	Dual-socket quad-core Shanghai processors	Dual-socket hex-core Istanbul processors	Possibly a an upgrade to the next generation processor
Memory per core	2 GB	1.3 GB	Expected to remain either same or could be lower
Network chip	SeaStar2.2	SeaStar2.2	Possibly GEMINI
Compilation environment	CCE 7.1.1	CCE 7.1.2	Future CCE releases
Communication library	GASNet	GASNet	DMAPP

Table 12: Test system configurations and roadmap

In the earlier report D8.3.1 [2], results of a memory bandwidth benchmark called STREAM that is available as a part of the CAF benchmark suite were presented. It reports bandwidth (MB/s) data for access patterns namely COPY, SCALE, ADD and TRIAD for a single image. In the CAF evaluation mode, the benchmark reports results by splitting the procedure, and accessing the elements remotely from another image. The code builds with some modifications to the makefile and by renaming the CAF files with extension .caf to .ftn. Moreover, some sync operations or API calls that are supported by the Rice CAF compiler but are not part of the CAF standard were modified to build the benchmarks. Additionally, the Rice CAF compiler permits declaration of co-arrays in common blocks while the Cray compiler does not. The NAS parallel benchmarks required several modifications to the source files to address the common block declaration restrictions and to use the syn_all API calls in place of sync_notify and sync_wait calls.

To investigate the performance characteristics of the compiler-generated code and impact of remote communication operations, we select the CAF version of the STREAM benchmark to understand if and how remote memory accesses are scheduled, overlapped and synchronized by the compiler and the runtime systems. The STREAM benchmark gives the data throughput for a set of 4 basic operations on large coarrays (larger than cache size), on the same image and across different images. Table 13 offers some results collected for these operations against 4 different access patterns: local array (as a baseline measure of the operation cost), local coarray (as a baseline measure of the PGAS construct), remote coarray in the same node,

and remote coarray in a different node. For each case, three pieces of information are displayed: (i) the language construct, (ii) the optimization applied by the compiler, and (iii) the operation throughput in MB/s. All tests were performed on the Cray XT5 available at CSCS.

In every case, the copy operation is pattern-matched, and replaced by an optimized library call, offering good performance overall. We could take the performance degradation that occurs when accessing another image (around 60 % for a coarray in the same node and nearly 70 % if it is in a different node) as the baseline cost of a remote versus a local access.

More interesting though is the disabling of vectorization for any construct that contains a coarray, even if it is only referenced locally (i.e. no brackets in the expression). This causes unnecessary performance degradation (~30 %) for the local access with respect to the vectorized version. In the absence of vectorization, the compiler opts for loop unrolling for all operations that do not match a predefined pattern. Even so, the performance of scale, add or triad operation is about 1000 times slower when accessing a different image, which goes well beyond the baseline degradation exhibited by the library calls used for the copy operation.

	Copy	**Scale**	**Add**	**Triad**
Local array (a, b, and c are regular Fortran arrays)	c(1:n)=a(1:n)	b(1:n) = scalar*c(1:n)	c(1:n) = a(1:n) + b(1:n)	a(1:n) = b(1:n) + scalar*c(1:n)
	Pattern-matched	Vectorized and unrolled ×4	Vectorized and unrolled ×4	Vectorized and unrolled ×4
	8524.85 MB/s	8450.93 MB/s	8792.65 MB/s	8716.84 MB/s
Local coarray (a, b, and c are coarrays)	c(1:n)=a(1:n) b(1:n) = scalar*c(1:n)	b(1:n) = scalar*c(1:n)	c(1:n) = a(1:n) + b(1:n)	a(1:n) = b(1:n) + scalar*c(1:n)
	Pattern-matched	Unrolled ×8	Unrolled ×8	Unrolled ×8
	8390.11 MB/s	5766.99 MB/s	6225.04 MB/s	6191.07 MB/s
Remote coarray, same node	c(1:n)=a(1:n)[2]	b(1:n) = scalar*c(1:n)[2]	c(1:n) = a(1:n)[2] + b(1:n)[2]	a(1:n) = b(1:n)[2] + scalar*c(1:n)[2]
	Pattern-matched	Unrolled ×8	Unrolled ×8	Unrolled ×8
	3372.67 MB/s	1.42 MB/s	1.50 MB/s	1.50 MB/s
Remote coarray, different node	c(1:n)=a(1:n)[2]	b(1:n) = scalar*c(1:n)[2]	c(1:n) = a(1:n)[2] + b(1:n)[2]	a(1:n) = b(1:n)[2] + scalar*c(1:n)[2]
	Pattern-matched	Unrolled ×8	Unrolled ×8	Unrolled ×8
	2673.65 MB/s	5.10 MB/s	4.03 MB/s	4.03 MB/s

Table 13: STREAM benchmark results for the CAF compiler

Since the compiler's inability to retain the microprocessor-level optimization, such as SSE vector, is a serious limitation, we ran a similar set of experiments on the Cray X2 vector platform to find if these limitations exist on a vector system. Potentially, benefits of PGAS communication optimization could be offset by un-optimized code generated for the microprocessor. The X2 CAF compiler however retains the vector instruction optimization as shown below and the runtime data presented in Table 14 also confirms our findings.

```
On X2 system
```

```
791. 1      Vr------<      DO  j = 1,n
792. 1      Vr                b(j) = scalar*c(j)[2]
793. 1      Vr------>      end DO
```

	Single image	Two images
Copy	81.25	37.57
Scale	85.63	37.48
Add	57.54	34.95
Triad	60.37	34.95

Table 14: CAF STREAM results (GB/s) on X2

We therefore conclude that there is a limitation in the CCE x86 CAF compiler or some compiler dependency checks that fail on an x86 system preventing the compiler from vectorizing or retaining microprocessor-level optimization. If unresolved, this issue is likely to severely limit performance on a GEMINI based system where an x86 commodity multi-core processor will constitute a processing node. This report includes an additional set of results for a matrix multiplication implementation using the CAF constructs. One dimension of the 2D matrix is declared as a co-array dimension:

```
double precision :: a(blockSize,blockSize)[p, *]
double precision :: b(blockSize,blockSize)[p, *]
double precision :: c(blockSize,blockSize)[p, *]
```

The usability evaluation of the Co-Array Fortran (CAF) compiler is performed on a Cray XT5 system, which is the latest Cray XT series platform. Recently, the CSCS XT5 system has been upgraded to a hex-core AMD processor called Istanbul from a quad-core Shanghai processor. The resulting system now has a peak performance of 212 TFlop/s, improved from 141 TFlop/s. The matrix is then striped using the block sizes and stripe sizes and the number of blocks and subsequently the iterations are distributed across CAF images. Runtime performance of the code is evaluated using 4 images (block size 2×2) and by distributing images across XT5 sockets and nodes to understand the impact of shared memory and remote communication operations. Results are presented in Table 15, which show that currently the CAF code generation mechanisms are unable to exploit local memory references (local to sockets and nodes). Considering the current version to be a functional compiler, these results demonstrate the ability of the compiler to work with different OS mapping and scheduling schemes that could be potentially quite useful once a performance version of the compiler is available.

Striping	1	2	4	8	16
Same node & socket (in s)	2.25	2.55	3.13	3.601	3.72
Different nodes (in s)	1.99	2.27	2.71	3.32	3.77
Different sockets & nodes (in s)	1.805	1.854	1.84	2.07	2.37

Table 15: Impact of alternate image mapping configurations on a CAF code execution

3.5.3 *UPC Experiments*

The usability evaluation of the UPC compiler is performed on a Cray XT5 system. Integrated UPC compiler is part of the Cray Compiler Environment (CCE). Earlier results were reported for version 7.1.x and the current version is 7.2.x. As indicated earlier, the main objective of

the CSCS task is to evaluate the usability of the compiler for code development and execution for representative benchmarks.

A small number of modifications were also required to the makefiles and source code in order to build the UPC benchmarks using the Cray UPC compiler. Initially, there were some issues with the UPC collective operations, which were addressed by the latest release of the CCE framework, 7.1.x. All UPC API conformance benchmarks that are part of the George Washington University test suite were built and tested successfully on the XT5 system. At the same time however, code modifications are needed for function calls such as `upc_local_alloc`, which is not supported by the Cray compiler and `forall` (`upc_for_all`) calls.

For UPC, we considered a synthetic micro-benchmark to analyze the compiler's ability to apply optimisations automatically. For that we tested some variants of the *upc_forall* construct over statically defined shared arrays. A upc_forall loop is an iteration statement where the iteration space is subdivided between the threads running the program. For this purpose, besides the typical initialization, condition, and update terms of a C for loop, there is a fourth term, known as the affinity expression, which defines which thread should execute which iteration. The affinity expression can be chosen so that each thread operates on the maximum amount of local shared data and the minimum amount of remote shared data, to deliver performance. This expression can be either an integer, which defines the loop indices to execute for each thread through a simple module operation, or it can be a shared pointer, in which case the iteration is performed by the thread whose pointed data is local. These two alternatives are shown for a simple array in Table 16.

upc_forall iteration statement using an integer as affinity expression	**upc_forall iteration statement using a shared pointer as affinity expression**

shared int arr[N];
[...]
upc_forall(i=0; i<N ;i++; i)
  arr[i] = get_value(i);
	shared int arr[N];
	[...]
	upc_forall(i=0; i<N ;i++; &arr[i])
	  arr[i] = get_value(i);

Table 16: Two equivalent alternatives to distribute the iterations for array arr between threads according to data locality

After analysing the generated code, we realize that, just as happened with the Fortran compiler, the loop cannot be vectorized even when accessing only local data. This causes the non optimised version to perform ~ 60 times slower than a vectorized version operating on a private array of the same size. In addition, the loop unrolling can be only performed for upc_forall loops with an integer affinity expression. In the case of a shared pointer affinity expression, the compiler is confused by the runtime function call to translate the pointer, which disables loop optimizations altogether, resulting in the unrolled version running twice as fast as the non optimized version. Even when using different blocking factors, including the default round-robin shared array distribution, the compiler is still unable to recognize the contiguous chunk in a thread's shared space, to apply some sort of optimization.

Since we observed different optimization patterns on the X2 platform as compared to the XT platform for the CAF code, we experimented using the canonical UPC matrix-multiply example on the X2 platforms with the UPC compiler. The compiler listings for the UPC code are shown below and correspond to our findings for the CCE CAF compiler. On the X2 platform, the compiler retains the vectorization (v) and unrolling (r) optimization while on the XT5 system only the loops were interchanged (i). Note that the serial version of the matrix-multiply code is vectorized by the Cray C compiler on the XT5 platform.

On X2

```
1------<   upc_forall (i=0; i<N; i++; &c[i][0]) {
1 V----<     for (j=0; j<M; j++) {
1 V            c[i][j]=0;
1 V r--<       for (l=0; l<P; l++)
1 V r-->         c[i][j]+=a[i][l]*b[l][j];
1 V---->     }
1------>    }
```

On XT5

```
1------<  upc_forall (i=0; i<N; i++; &c[i][0]) {
1 i----<    for (j=0; j<M; j++) {
1 i            c[i][j]=0;
1 i 3--<       for (l=0; l<P; l++)
1 i 3-->          c[i][j]+=a[i][l]*b[l][j];
1 i---->       }
1------>    }
```

To sum up, on the XT5 platform, the C compiler suffers the same limitations for UPC that were observed for the Fortran compiler with co-arrays (including failure to adequately schedule network operations), plus new limitations related to unrecognized UPC runtime calls that disable loop level optimization completely.

Table 17 summarizes the key findings for both PGAS compilers available as part of the CCE 7.2.x. Overall, CAF and UPC benchmarks results presented in this document confirm the Cray functional compiler's usability for codes that have been developed using standard CAF and UPC constructs. One of the bugs reported in the earlier report has been fixed. Using their experiences with multiple generations of the CCE PGAS compilers, the CSCS team concludes that the CCE infrastructure available currently for production-level code development would require a level of maturity for code development and performance tools and a performance model. For instance, debugging tools would be critical for development and testing and performance tools would benefit tuning and optimization efforts. Currently, the debugging and performance tools (CrayPAT) do not generate any PGAS programming model specific information to enable code developers to debug code or to tune for performance. We observe that CrayPAT tools on the X2 platform generate timings for PGAS specific API calls such as synchronization and barrier operations while on the XT5 system only flat, function-level profiles are generated. On the final note, the results reported here represent the best performance for three sets of runs. Significant performance variations are noted, in some cases a factor of 3 or more, between best and the worst case performance. This issue needs to further investigate either within the compiler and runtime infrastructure or using the performance tools that can capture and distinguish overhead for remote memory access patterns.

	CAF	UPC
Availability	Yes	Yes
Compilation	Successful	Successful
Execution in batch mode	Successful	Successful
Mapping and scheduling options availability at compile and runtime	Supported	Supported
Source code modification to existing code	API calls, constructs and filename extensions	Some API calls
Makefile modifications	Minimal	Minimal

Error messages (compiler)	Helpful	Helpful
Error messages (runtime)	Unhelpful	Unhelpful
Debugging and Performance Tools	Available	Available
Effectiveness of Tools for PGAS application	Minimal	Minimal

Table 17: Summary of experiences with CAF and UPC compilers available as part of the CCE framework on the Cray XT5 platform (options added since the previous release are highlighted)

According to the CCE roadmap, the future releases of the compiler with a communication infrastructure that is optimized for the Cray platforms for the PGAS languages would offer significant performance and scalability improvements. The CSCS team plans to rerun the tests on future hardware prototypes and compilers to confirm these performance goals and expectations. Our goal for this study is not to define a single metric for productivity and then attempt to measure it but to evaluate the usability of the PGAS development and execution environments and tools (debugging and performance) that enable code developers to port existing applications, to gradually introduce PGAS constructs in their existing parallel applications and to develop code from scratch where applicable. The focus of this study was primarily PGAS benchmarks, where codes have been developed from scratch primarily using either CAF or UPC. In future, the CSCS teams plans to evaluate productivity of the toolset for hybrid applications, i.e. applications that intermix MPI constructs with CAF and UPC constructs or productivity for using PGAS constructs incrementally in applications that have been parallelized using other programming paradigms, for instance, MPI message-passing applications.

3.6 Novel I/O: XC4-IO experiments

In this section, the main tests on XC4-IO prototype, related to SSD and metadata tests with Lustre file system are presented.

3.6.1 *Lustre Architecture*

Lustre is an object-based parallel file system which uses an ext3 modified version to store data. Its network is managed by the LNET component and is composed by five main elements:

- Clients
- Metadata Server (MDS) / Management Server (MGS)
- Object Storage Servers (OSSs)
- Object Storage Targets (OSTs)
- Metadata Target (MDT)

The MGS stores configuration information for all Lustre file systems in a cluster. It provides configuration information to other Lustre components: each Lustre target contacts the MGS to provide information, and Lustre clients contact the MGS to retrieve information. The MDS is a server that makes metadata available to Lustre clients via MDTs. Each MDS manages the names and directories in the file system, and provides the network request handling for one or more local MDTs. MDT, one per file system, is responsible to store data on an MDS. MGS and MDS could be configured on one single node or on different nodes; it depends on the system needs: if multiple file systems have to be configured, MGS and MDS have to be installed on separates machines, otherwise they can be configured on the same node.

OSSes provide file I/O service, and network request handling for one or more local OSTs, which store data on one or more of them. Each of these refers to one OSS and each OSS can communicate with more OSTs. Lustre lets spreading a single file over many OSTs, a Logical Object Volume (LOV) is responsible of file striping management across many OSTs. It can be done defining a specific stripe count, size and offset. Stripe count defines the number of OST on which a single file is spread, stripe size is the block size in which the file is divided and stored on different OSTs and stripe offset defines the starting OST on which data are saved.

To perform I/O tests, the XC4 prototype has been configured to work with Lustre version 1.8.1. The configuration adopted, compatibly with the Lustre architecture, is the following:

- MDS and MGS have been installed on the same machine, one of the HP ProLiant BL460c G6 blades. This assumption does not represent a problem because in the experiments multiple file systems are not used.
- MDT is represented by DDN S2A9900 which provides 1.5 TB for metadata storage.
- We configured 4 OSSes, using 4 HP ProLiant BL460c blades.
- OSTs are represented by DDN S2A8500, we create 24 Logical Units (LUs) if 1 TB each on this device.
- Lustre clients were installed on 32 nodes of the HP Woodcrest cluster.

The Lustre network works on Infiniband over RDMA: Infiniband OFED provided by Lustre was installed on the servers; instead client machines were configured to work with Open Fabrics OFED version 1.4.1.

3.6.2 *Throughput tests*

In order to evaluate the Lustre file system, throughput tests were first performed, to evaluate file system limits and drawbacks. OSTs were configured to work with striping on 24 LUs with striping size of 1 MB and starting OST randomly, selected between the available ones. The benchmark tool selected for this test is IOZone which can perform sequential and random reads and write, using 1 GB file for each client and a record size of 64 MB. This test excludes OSSes cache and executes one thread on each client. The throughput performance as a function of the number of clients is presented in Figure 82.

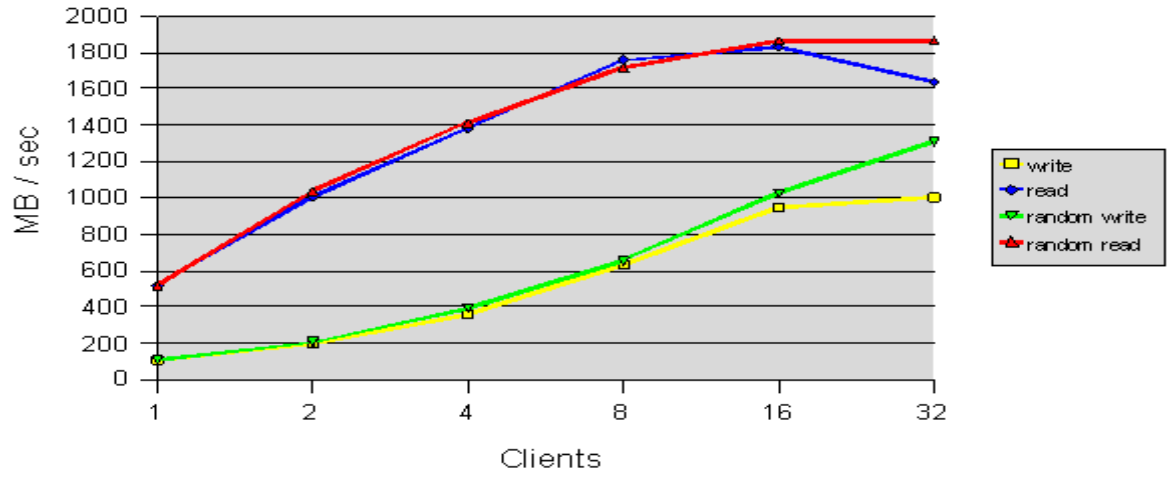

Figure 82: Lustre throughput performances

As evident in Figure 82, in these tests, Lustre easily reaches, with a small number of clients, the theoretical bandwidth limit of the prototype architecture (2.0 GB/s).

3.6.3 *File striping tests*

This second set of tests is focused to measure the suitability and the performance of Lustre for different environments, i.e. HPC, where large I/O operations are usually performed, or contexts where small I/O operations are more typical (i.e. access to Home directories).

Large I/O operations

For these test, the OSTs striping count has been identified as one of the most important parameters which strongly influence the file system performances. In order to assess the changing of Lustre behaviour, two similar tests were performed, using the IOZone benchmark tool, first with 1 LU striping and then with 24 LU striping. In these tests, each client writes and reads files with a size of 20 GB. The results are reported in Figure 83.

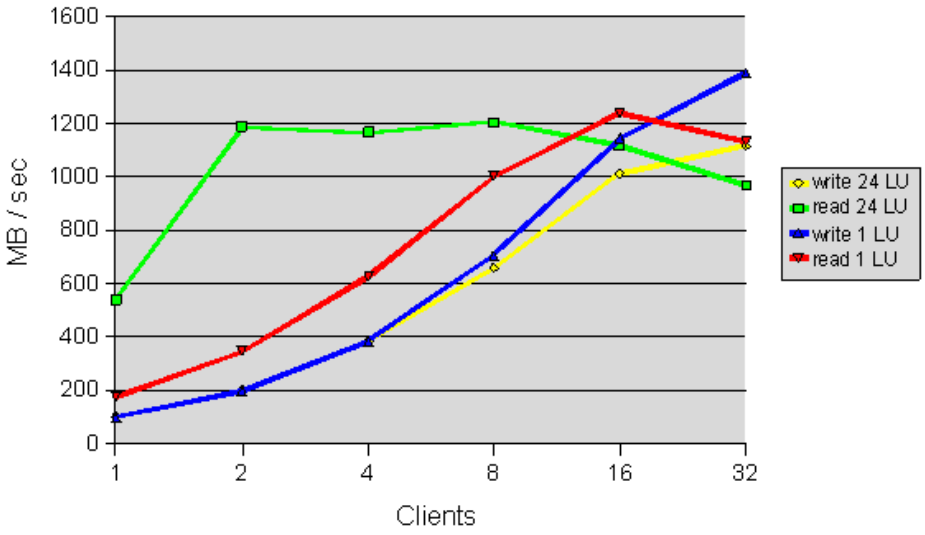

Figure 83: Bandwidth (in MB/s) for Read and Write operations with different striping count

This test shows that with 24 LU striping configuration the Read operations immediately reach high performances, instead a small striping configuration (1 LU) needs more clients to saturate all the available bandwidth. In case of Write operations the test presents similar performances with 1 and 24 LUs. This different behaviour is expected as each write operation implies two different activities on metadata: read and then metadata update. So, when the files are partitioned on 24 LUs, this continuous reading and update activity for the different portions of the file is heavy, neglecting the benefit of striping. Moreover we must consider that SSD disks are less performing in write operations respect to the read ones. The write operations are slower so do not allow to take advantage of the increased bandwidth available with the 24 LU, presenting a trend closer to the case with 1 LU. In case of 1 LU, a single metadata is present for the file (all is on the same disk) and consequently metadata management and upgrade is less expensive.

Small I/O operations

The tests just described have been performed again to evaluate the Lustre behaviour when stressed by multiple small I/O operations. In this case performance was measured in terms of metadata server I/O operations per second, instead of sustained bandwidth. In order to make these tests, we used some scripts which create, read and delete a large number of empty files on each client: in a first step scripts create and delete 10,000 files per node and then, in a second step, read 30,000 files per node from a client specific folder. Results are reported in Figure 84.

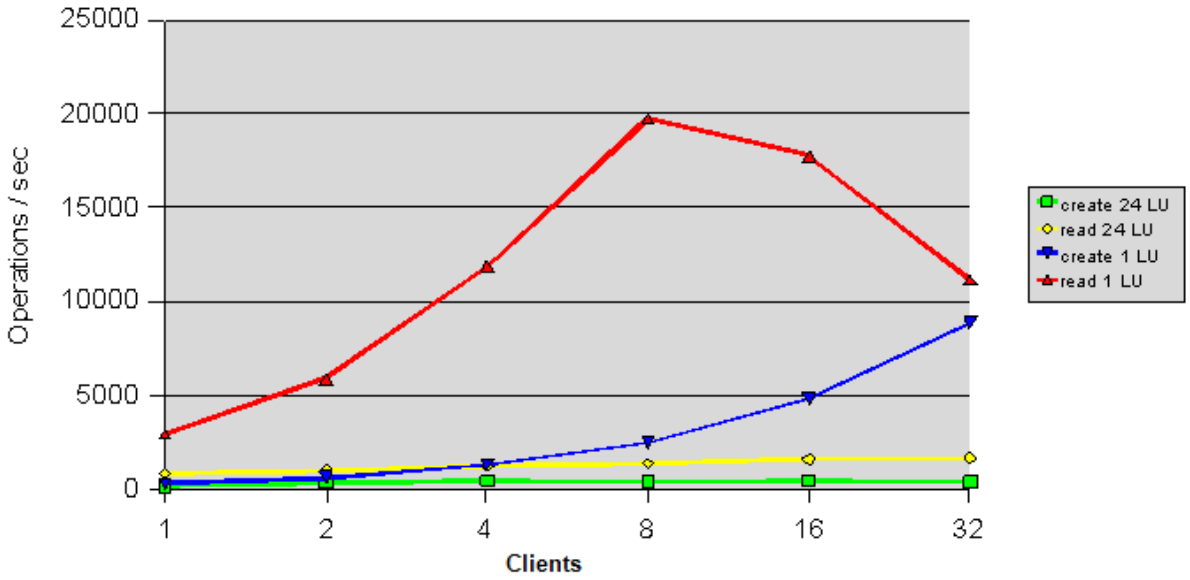

Figure 84: Performance (I/O ops/s) with different stripe count with a large number of files

Results show that 1 LU configuration performs strongly better when the file system has to manage a large number of parallel I/O operations.

CINECA and BADW-LRZ cooperated actively during the evaluation of the I/O prototype. In particular both sites agreed on the way to assemble the tests and to perform the measurements, mainly for the activity related to the small IO operations and the tests related to metadata device technology, described in the next paragraph. This cooperation was very important in order to share the methodology both for preparing the tests and then evaluate the results.

3.6.4 *Tests on the Metadata device technology*

The latest tests stressed the metadata server which had to manage a large number of files, storing and retrieving information for the requested I/O operations. Focusing on metadata server, our intent was to assess performance using metadata target devices with different technologies. We assessed SSD technology compared with traditional magnetic hard disks, in terms of I/O operations per second performed. In order to produce these values we repeated the previous tests based on small I/O operations, considering both the SSD and the magnetic disk technologies. The results are reported in Figure 85.

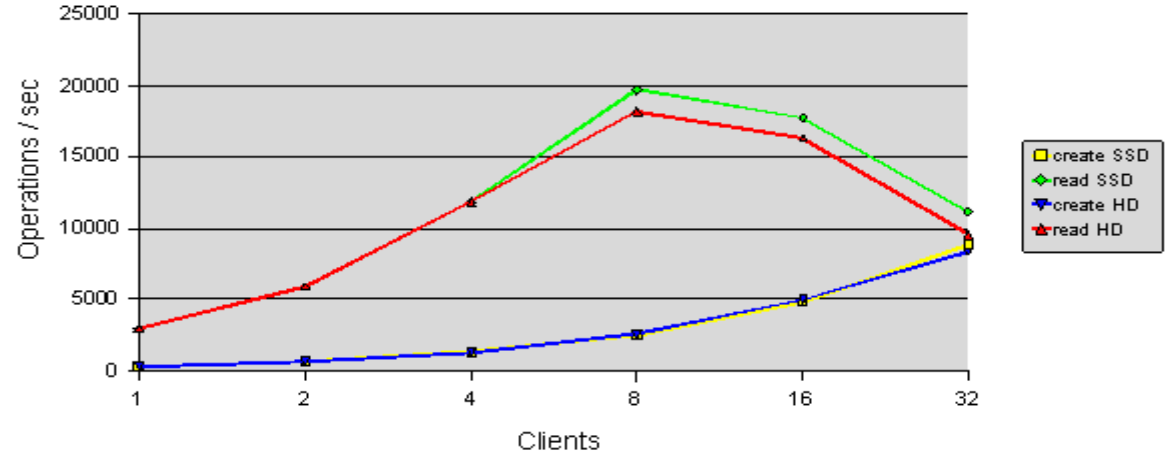

Figure 85: Comparison between SSDs and magnetic Hard Disks (I/O ops/s)

The values obtained, show that SSDs performs slightly better with read operations when the number of clients and I/O operations increase, while the technologies produce very similar

performance for create operations. The slight difference observed, both in reads and creates, is due to the presence of target device controllers. In fact, their caches balance the gap between SSD and magnetic disks performances which are much larger in single device measurements.

3.6.5 *Metadata server load tests*

Additional measurements were made to better investigate the reason why the read performance decays when the number of clients and simultaneous file requests increase. In this case the CPU utilisation of the MDS was measured. The results are presented in Figure 86.

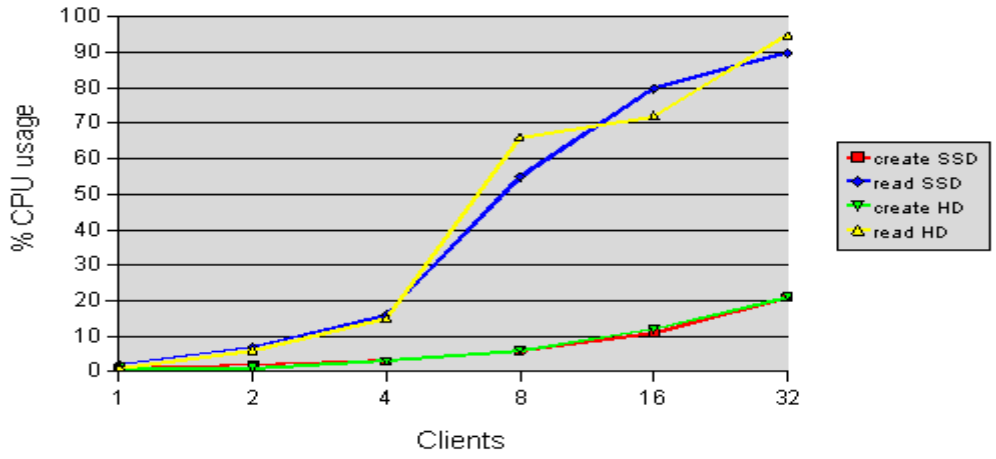

Figure 86: Comparison between SSDs and magnetic Hard Disks (% cpu utilization of the MDS)

The results show that MDS load increases rapidly while the number of clients grows. The saturation is reached with 32 clients. This is the main reason of the decay observed in the tests related to the devices technology.

3.6.6 *Parallel I/O libraries test*

In collaboration with IDRIS some other tests have been made to evaluate the performance of the prototype using parallel I/O libraries, such as MPI-IO, PnetCDF and HDF5, over Lustre. Tests have been performed using two benchmark tools: IOR and RAMSES.

IOR Tests

The first tests have been performed using the IOR benchmark tool with 2 MB transfer size and 256 GB aggregate file size, testing all the previously cited libraries using both individual and collective primitives. Results are presented in Figure 87 and Figure 88. A value ‘0’ means that the test crashes.

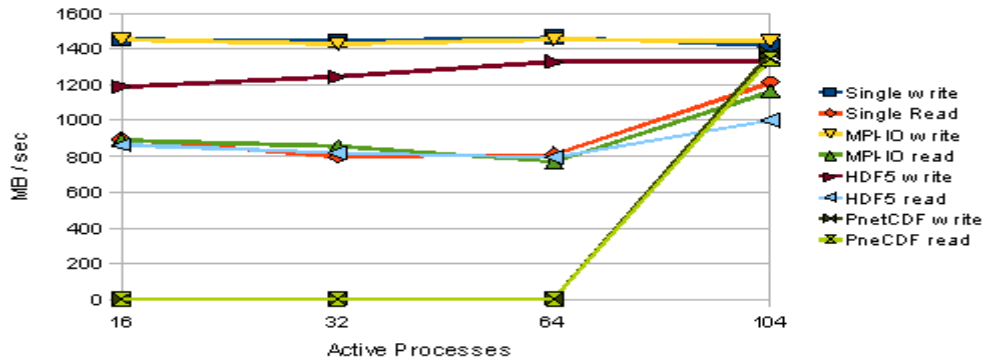

Figure 87: IOR results (MB/s) using individual communication

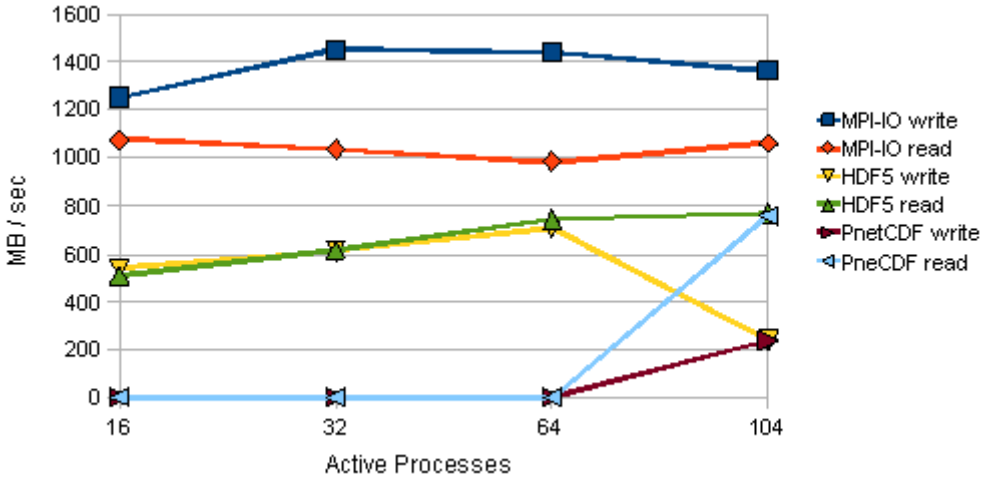

Figure 88: IOR results (MB/s) using collective communication

In terms of performance, the One-file-per-process approach and MPI-IO are comparable; HDF5 is a little behind, whereas PnetCDF crashed on most of the tests due to known limitations of the library. The choice of coordination of the call (collective vs. individual), for all parallel libraries, is a parameter that strongly influences the system performance; results show that individual mode performs better than collective. Another noticeable behaviour of the system is that writes performs about 35 % better than reads.

RAMSES Tests

The RAMSES tests have been done using up to 26 nodes on XC4, so different tests have been produced using 16, 32, 64 and 104 MPI tasks respectively. For these tests, a dataset producing an amount of 75 GB per output (25 GB for AMR file and 50 GB for Hydro file). Note that the AMR file, even if it is smaller in size, is much more complex in its structure than the Hydro file, stressing the management of metadata. Few hints (romio_cb_write, romio_ds_write) have been tested, in attempt to improve performances. Hints usage allows 5 configurations for test:

- No hints;
- romio_cb_write = disbled, romio_ds_write = disabled (1);
- romio_cb_write = disbled, romio_ds_write = enabled (2);
- romio_cb_write = enabled, romio_ds_write = disabled (3);
- romio_cb_write = enabled, romio_ds_write = enabled (4).

Results for AMR and Hydro files are reported in Figure 89 and Figure 90.

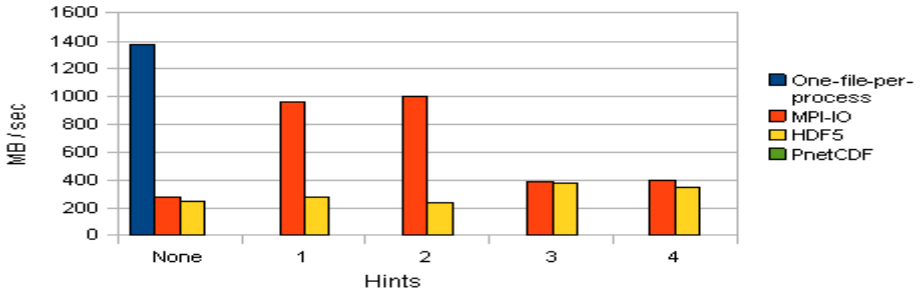

Figure 89: RAMSES Test (MB/s) reads

Without any hints, the reading throughput obtained with the One-file-per-process approach outperforms the one of MPI-IO or HDF5 by a factor of 5. When using the proper hints, we got a real improvement for MPI-IO, whereas HDF5 reaches 50 % of improvement. Nevertheless, even with proper hints usage, the One-file-per-process approach outperforms MPI-IO by nearly 40 %.

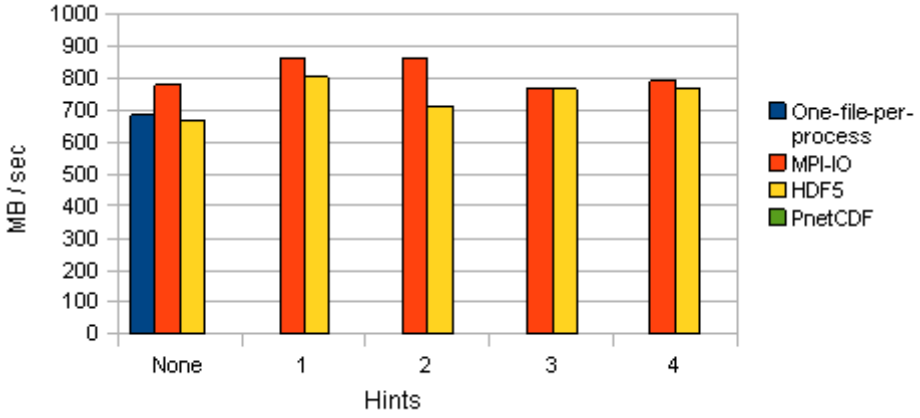

Figure 90: RAMSES Test (MB/s) writes

When writing, this time, HDF5 and the One-file-per-process approach gave similar performance, whereas MPI-IO is 13 % better. If we use hints, then MPI-IO and HDF5 outperform the One-file-per-process approach by respectively 26 % and 18 %.

In conclusion, the performance obtained with the One-file-per-process approach is of the same order to the one obtained with MPI-IO. For parallel I/O libraries based on MPI-IO, the usage of proper hints could lead to a great improvement in terms of performance as well as the mode of coordination of calls (collective vs. non-collective) which should be chosen with care.

3.6.7 *Preliminary tests on pNFS*

Since pNFS technology is still under development, the developers do not yet provide modules that can be compiled on production kernels. In order to test pNFS exporting Lustre or other file systems, we had to install an *"ad hoc"* kernel especially patched for pNFS.

This approach generated hardware and software problems during the installation. We tried to face these issues testing different UNIX distributions or using different development branches or implementations from pNFS Git source tree.

We tested latest pNFS release (vers 2.6.32-rc5) on latest Red Hat (EL 5.4), Fedora (Core 12) and Ubuntu Server (9.10 Amd 64). Red Hat presents a bug in some scripts that generates init RAM disk file causing boot problems. So we moved to Fedora, which implements the latest version of those scripts, but it is not supported by HP Proliant servers. So we finally moved to Ubuntu. This later distribution allowed us to install and boot pNFS kernels and we are now investigating on how to configure the system to make pNFS protocol working, as we still observe some system configuration problems.

3.6.8 *Conclusion*

This test suite demonstrate that Lustre can satisfy the requirements of HPC environments, it easily reaches the theoretical limit of our architecture so, probably, increasing the number of OSSs and OSTs and using links more with a larger bandwidth, the performances should scale up further and reach higher values.

Lustre performance is strongly influenced by Lustre stripe count and by the type of I/O operations performed. For few large I/O operations 24 LU striping configuration is winning, Lustre can immediately saturate the available bandwidth instead 1 LU configuration needs more clients to reach those values.

When performing lots of small I/O operations the MDS is the bottleneck; it's overloaded as shown by its CPU usage (%). In this environment 1 LU configuration is the best choice because MDS has to contact only one OST for each file, on the other hand, striping each file on 24 LU, MDS wastes resources in contacting each OST, informing them that they will be responsible to store portions of specific files, even if they are small.

The most noticeable drawback of Lustre 1.8.1 is the single active MDS configuration; this can be noticed simulating a HOME behaviour, which stresses the MDS with a lot of requests. We tried to configure MDS on SSD disks first and HD after, but both configurations present a performance worsening when the number of simultaneous operations increases. This is due to the MDS CPU saturation which is overloaded when it must manage a huge number of simultaneous requests. Also clients' performance is affected by the MDS bottleneck, their CPU utilisation decreases when the number of files managed by MDS increases, due to its overloading, so they have to wait to be able to perform I/O operations (IO_wait status). Hence, a lack of Lustre architecture is the possibility to scale MDS only vertically, increasing its CPU power or memory, but not horizontally, preventing from parallelize its work. This feature will be available only in Lustre 2.0 version which is still under development.

The last test was aimed to proof if SSD technology can help to reduce the bottleneck of MDS when it is overloaded by multiple I/O requests. We observed that SSDs performs slightly better than traditional hard disks. The hard disks are penalised by the waste of time due to rotational heads, mainly when they must satisfy a large number of simultaneous requests. In any case both technologies cannot avoid MDS bottleneck effects, due to Lustre architecture.

This first phase of tests on the Lustre file system allows us to reach some conclusions:

- SSDs give slightly higher performance than traditional hard disks;
- Lustre is a suitable file system for high performance computing because it reaches high performance saturating the available bandwidth and showing good scalability;
- Lustre 1.8.1 has a main bottleneck: the single MDS. In this version only Active / Passive configuration is allowed. The possibility of using multiple parallel metadata servers should make Lustre more suitable also for environments characterised by small I/O operations;
- Lustre striping configuration strongly influences the performance of the system. Acting on striping count, Lustre can result more suitable for a certain context rather than another one. For example, in HPC environments, where large I/O operations are required, it is convenient to exploit all the available bandwidth, striping files on all the available OSTs; instead, in an environment, where a lot of small I/O operations are required, is useful to store a file completely on a single OST avoiding the MDS to contact a large number of OST to manage every single small file.

This first activity on testing pNFS exporting Lustre or other file systems is progressing slowly with some problems. The main reason for that are difficulties to integrate pNFS on Linux production kernels. At this point, we can observe that although pNFS seems promising for the HPC environment, it is still immature and not acceptable for HPC production contexts.

3.7 Energy efficiency

3.7.1 *The SNIC-KTH system*

The support for the SNIC prototype placed at KTH was decided by the PRACE Management Board (MB) in May 2009 with the objective of studying energy efficiency of systems based entirely of commodity components for cost effectiveness, and without acceleration for preservation of parallel programming models. A target was set to reach a performance level of about 350 GFlop/s/W for HPL, thus comparable to the BG/P but with more than twice the memory per core; 1.33 GB/core for the prototype expandable to 5.33 GB/core. A high-density solution comparable to the BG/P was also sought.

To achieve this goal and to support QDR Infiniband effectively support for PCI Express Gen2 was deemed necessary which required a new server design to accommodate a new chipset from AMD. As described previously, nodes in the form of a 4-socket blade were selected in collaboration with AMD and Supermicro. It was also decided to undertake a new blade chassis design with a built-in 36-port QDR switch. A density of 5.7 sockets/U was achieved resulting in a peak capability of 15 TFlop/s per rack with 2.6 GHz Istanbul CPUs, which compares favourably with the 13.9 GFlop/s per rack for the BG/P. For energy efficiency a 2.1 GHz CPU was however selected for the prototype yielding a peak capacity of 12 TFlop/s per rack.

Two chassis were delivered already in June 2009, about 6 weeks after the PRACE MBs decision. This was clearly too short time for a new board and chassis design to be carried out. The two chassis had 4-socket blades but not the new chipset supporting PCI Express Gen2, and QDR IB. Furthermore, the new chassis that were designed for the PRACE prototype also had more efficient power supplies than the old chassis. By the middle of July, less than two months after receipt of order, Supermicro did have the new board design for in-house testing. The tests were successful. By early August production were to start, but typhoon Morakot hit Asia hard and caused a delay of about a month. When production boards became available in late September and tests were carried on those in the process of building the prototype yield problems were encountered that were traced to the motherboard manufacturer having deviated from the specification. The decision was taken by Supermicro to have all motherboards redone, which caused an additional delay of close to a month. All of the parts of the prototype were delivered November 3rd, but in the interest of avoiding logistical delay the system was not preassembled and tested by the vendor. Assembly, test and installation all took place in the computer room of PDC, KTHs HPC centre.

The two chassis delivered in June ended up primarily being used for evaluation of some design decisions for the full prototype with the new chipset, motherboard and chassis. In particular different memory configurations and memories from different memory vendors were tested for performance and energy efficiency. Preliminary “theoretical” evaluations based on chip data indicated a potentially significant difference in DIMM power consumption depending on what memory chips were used in constructing the DIMMs. DIMMs from Elpida, Hynix and Samsung were acquired, as they seemed most promising from an energy efficiency point of view based on HPL and Stream. The DIMMs were evaluated both in the two chassis delivered in June, and on preproduction boards at Supermicro that had the proper chipset and the new board design as well as new chassis. The differences in power consumption was much less than predicted by the power calculator and more in line with the “common” expectation that differences are minor. However, in the phase-I systems, delivered in June, a difference of over 6 % in HPL MFlop/s/W was experienced between Elpida and Samsung memories, a difference that however diminished with the number of repeated runs made.

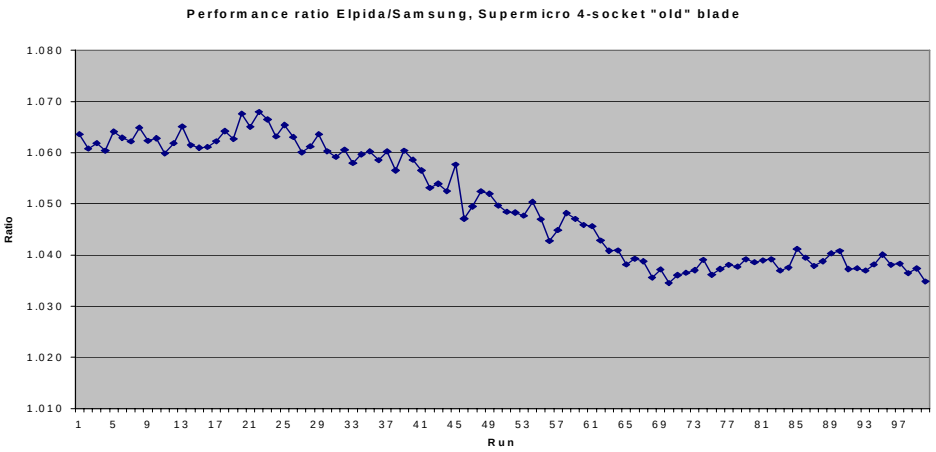

Figure 91: Performance ratio of Elpida/Samsung, Supermicro 4-socket blade

The power consumption was also measured and the graph below shows how the GFlop/s/W compares for Elpida and Samsung 2 GB DIMMs with 16 DIMMs/blade for the Phase-I system. Measurements were made with only 2 blades in a chassis, so the chassis "overhead" is disproportionate and masks the difference to a significant degree. However, since the objective was to decide on which type of DIMMs to use we only sought to get a relative values not absolute values.

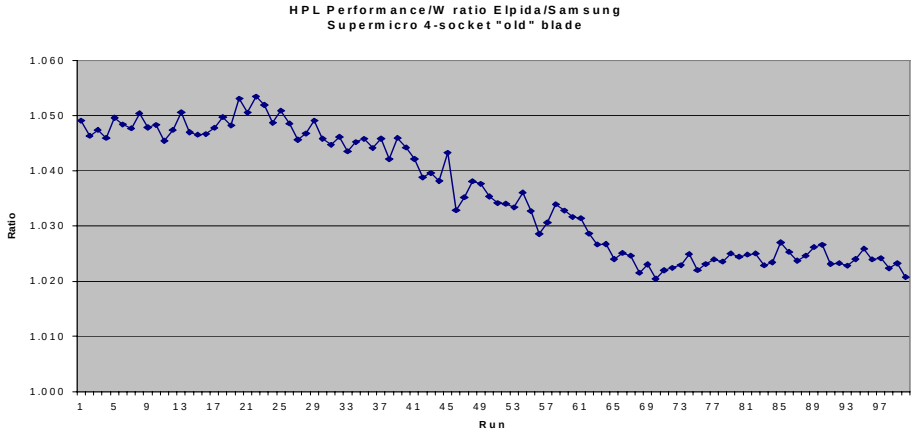

Figure 92: Performance/W ratio for Elpida/Samsung Supermicro 4-socket blade

Test of different DIMMs were subsequently carried out in the preproduction of new boards and chassis at Supermicro. In those tests, 4 GB Hynix DIMMs with half the DIMM slots populated yielded the best performance and were chosen for the Phase-II delivery. Based on those tests a Stream-Copy performance of 42 GB/s per node was expected for the Phase-II system, and a HPL performance of about 335 GFlop/s/kW expected.

For Stream we have so far measured a peak bandwidth of 42.6 GB/s for 24 threads per node, i.e. one thread per core. This performance was achieved with a power consumption of 362 W/blade, or 90.5 W/socket. No attempt at optimisations neither in regards to compiler options or in regards to power management features has been done yet. The preliminary measurements are graphed below. Threads were allocated to sockets in a round-robin fashion.

For HPL we have so far observed a peak performance of 344 GFlop/s/kW at 79 % of peak node performance. This performance is a little better than predicted from the preproduction system at Supermicro. We expect the peak HPL performance to improve somewhat as we get opportunity to carry out code and power management optimizations. For comparison, the BG/P systems on the most recent Top500 list falls in the 364 – 379 GFlop/s/kW range.

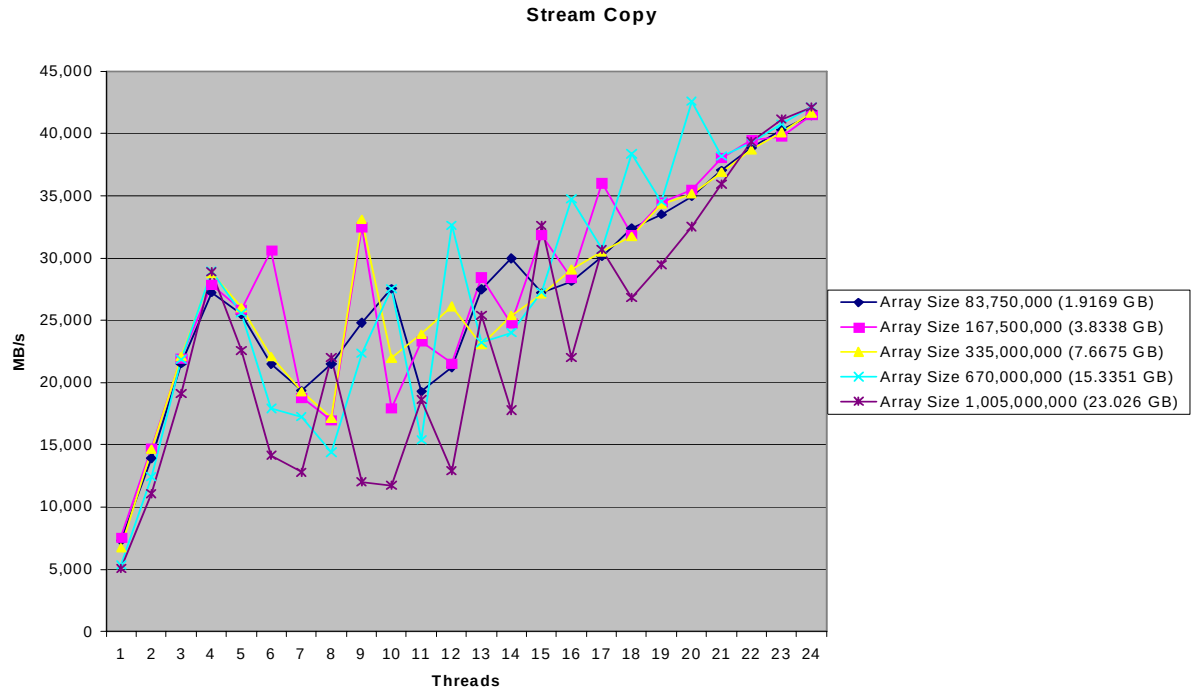

Figure 93: Stream Copy

Random Memory Access Results:

HPCC contains three different random access benchmarks:

- SingleRandomAccess - run on one (randomly selected) MPI process;
- StarRandomAccess - SingleRandomAccess run on every MPI process;
- MPIRandomAccess - using MPI for updates (and run on every MPI process).

For SingleRandomAccess the hitherto observed average results is 0.016002 GUP/s (from a population of 4 with minimum 0.0150959 and maximum 0.0184563).

The results of StarRandomAccess and MPIRandomAccess on single system runs can be viewed in Figure 94 and Figure 95. Figure 94 indicates a near linear relation between tasks and updates for StarRandomAccess. However, the relationship could be more step-like than the plots indicates, considering the platforms memory sub-system grouping (into different caches and sockets). The chosen task-stride (4) could hide the fact.

MPIRandomAccess does not scale well. This can be considered unsurprising as the addition of tasks (cores) will only in a very limited sense increase the MPI message capacity. (Especially as the MPI used in the tests were not optimized for shared-memory performance.) As an example of results from an internode run is, 240 processes over 10 nodes (sharing the same ingress switch in the fat tree) gives 0.011374999 GUP/s or only an average of 0.0000474 GUP/s per task.

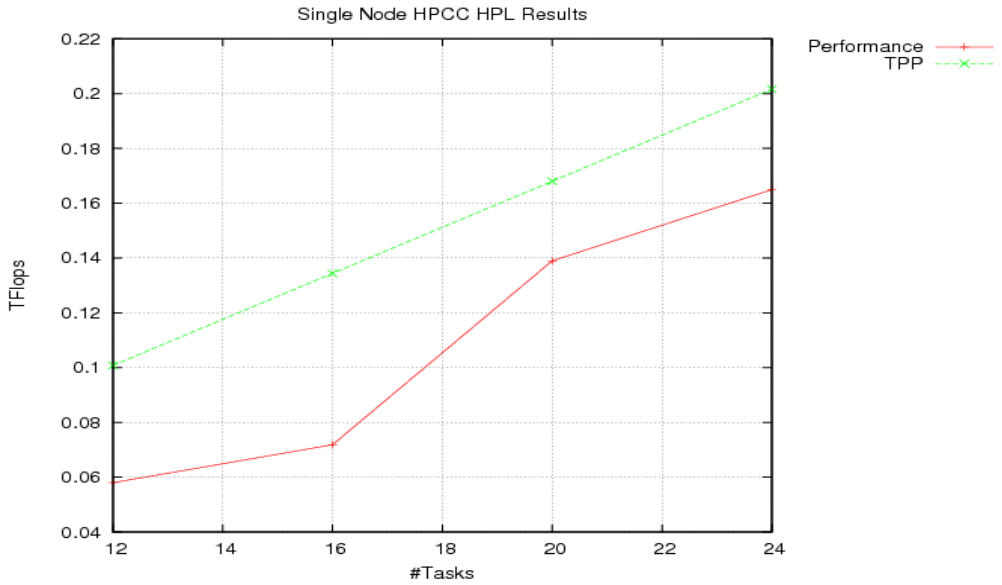

Figure 94: Near linear relation between tasks and updates for StarRandomAccess

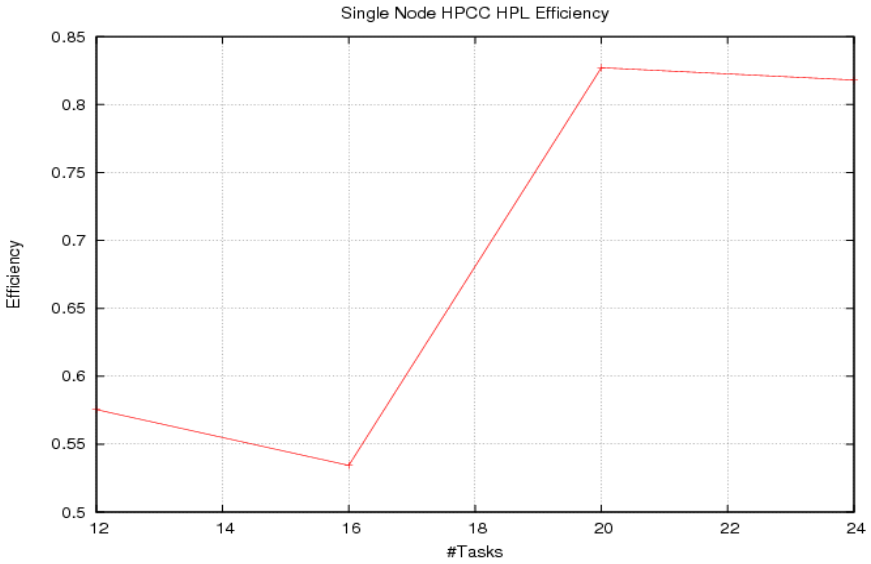

Figure 95: Relation between tasks and updates for MPIRandomAccess

EuroBen on SNIC-KTH System:

The four EuroBen kernels mod2am, mod2as, nod2ah and mod2f have been ported to C together with the Intel Math Kernel Library (MKL) as well as C together with AMD Core Math Library (ACML). One of these four kernels, mod2am, has also been ported to C using the dgemm function of the Gotoblas2 library. The experiments were performed on the prototype and the results have been compared.

Figure 96 shows the performance differences between Fortran, C+ACML and C+MKL implementations of the mod2am kernel. All three implementations have the same behavioral pattern and show poor performance.

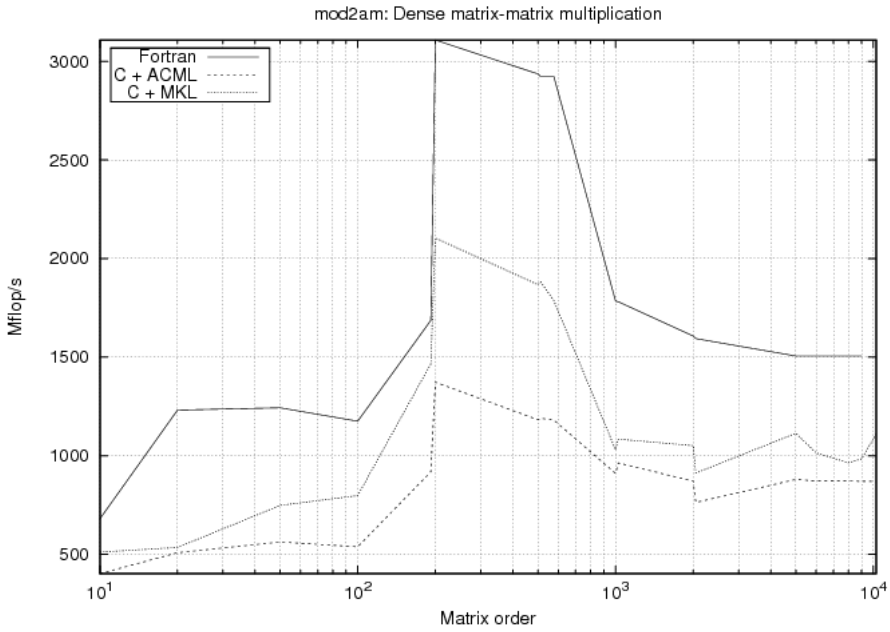

Figure 96: Comparison of the performance of the dense matrix-matrix multiplication kernel for three different implementations: Fortran, C with ACML and C with MKL

To reach the better performance the code has been changed. The core dense matrix-matrix multiplication has been replaced with the DGEMM function from the GotoBLAS2 library. Instead of two dimensional arrays one dimensional initialization has been implemented. The 4×4 block was the building-stone for this DGEMM implementation. The code has been compiled using the PGI compiler version 10.3. Figure 97 shows the performance of mod2am with DGEMM. We clearly see a performance improvement which increases with the increase of the matrix size. About 7.8 GFlop/s, or 93 % of theoretical peak performance, was reached with this implementation.

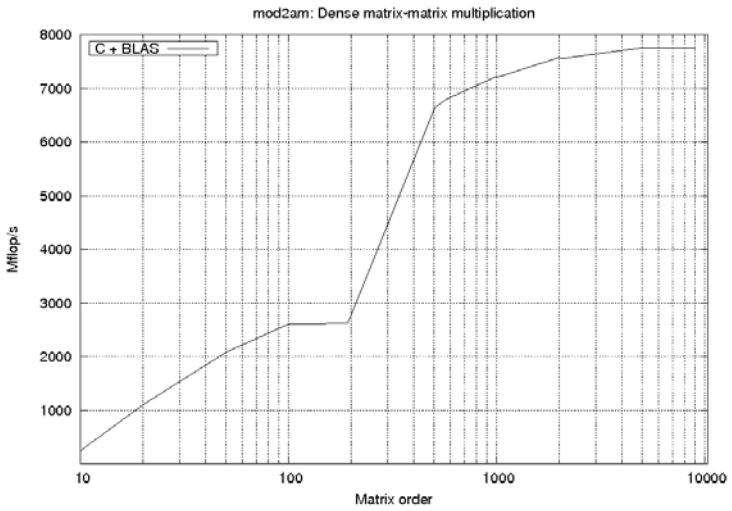

Figure 97: Performance of the dense matrix-matrix multiplication with GotoBLAS2 libraries

Figure 98 shows rather disappointing performance of mod2as kernel. This is not surprising and agrees with the results shown for other platforms such as Nehalem-EP. Similar to mod2am kernel we expect to see much better performance for sparse matrix multiplication using GotoBlas2 libraries.

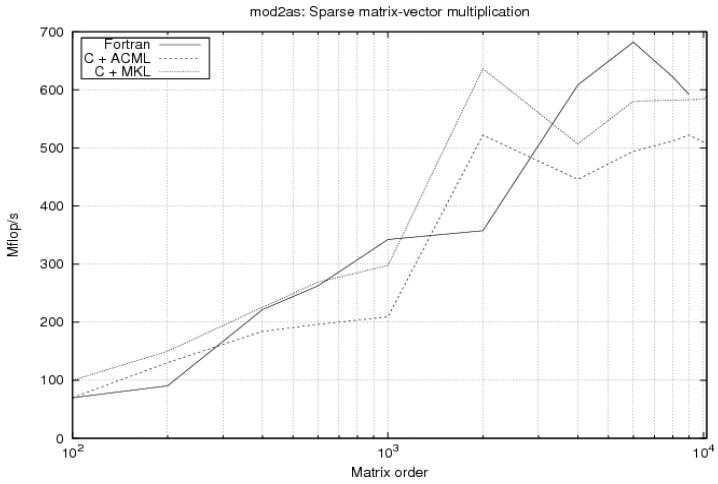

Figure 98: Performance of the sparse matrix-vector multiplication

Figure 99 and Figure 100 show the performance of random number generator mod2h and fast Fourier transform mod2f kernels, respectively. Again, both kernels have been tested in Fortran, C+ACML and C+MKL versions. Both kernels produce poor performance. However, for the mod2h kernel the performance of Fortran version is about 8 times better than of the C+MKL and C+ACML versions. The two implementations of the mod2f kernel show completely opposite behavioral pattern.

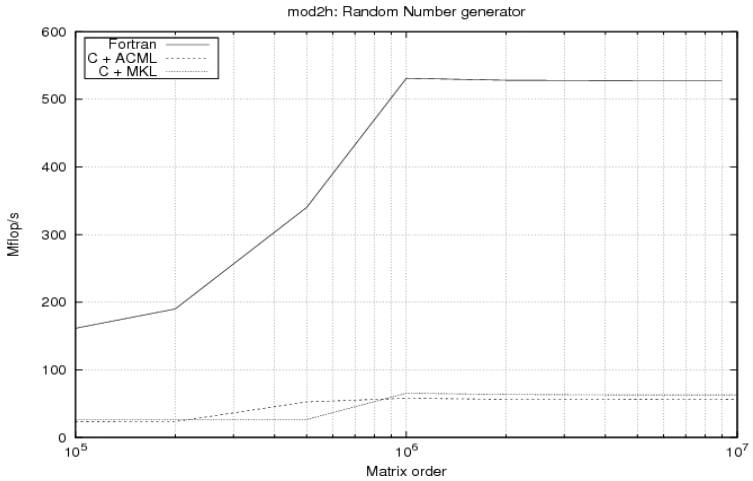

Figure 99: of the random number generator kernel

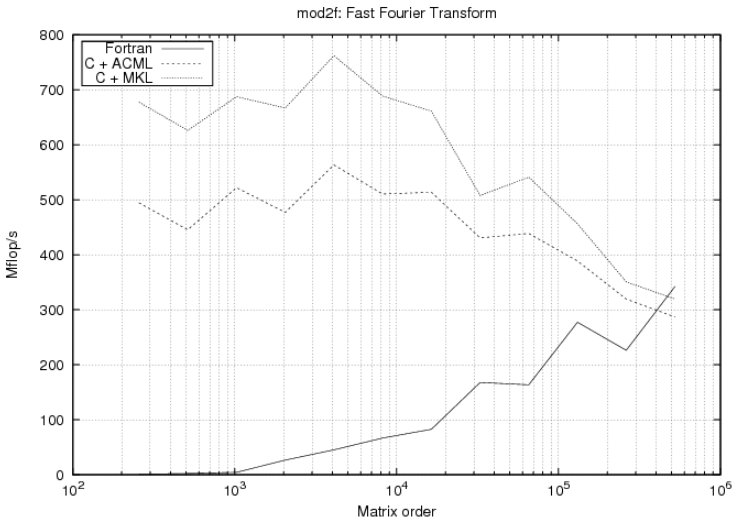

Figure 100: Performance of the fast Fourier transform kernel

Summary:

For the current SNIC-KTH prototype architecture the use of GotoBLAS2 libraries gives the expected maximum performance. It is highly recommended to change the manual implementations of the EuroBen code using functions from the libraries already implemented.

3.7.2 *PSNC results*

In order to get the exact measurements of the power consumption for each component of various servers in a setup as described in Subsection 2.2.2 we should isolate each part and measure all power supply connectors of each chip. Unfortunately, due to construction of the machine it was not possible to do it this way. We have chosen to measure the current of individual power lines from the server power supply to observe how the power consumption profile changes for each power line depending on installed expansion cards and tasks executed.

In all cases, the tests were performed using NAMD (appoa1 test) [19] and GROMACS [36] applications. As the assessment of the performance of the tested machine is not the main goal of the work, the results are used only as a reference for those gathered by other partners. Apart from application tests, we have used some software for generation of artificial load (e.g., *Cpuburn*, *iozone*, *dd*, *memtester)* to ensure maximum load and power consumption.

The power draw was tested in four states of the machine: with power supply plugged in but machine is in power off mode, machine in idle mode with OS running, running benchmark application and finally maximum load possible generated using various software. The equipment used to measure the power consumption of the tested systems encompasses several parts:

- Power meter Lutron DW6090;
- Data logger Yokogawa XL120;
- Multimeter PICOTEST M3500A.

We present the result for 1U Xeon based machine, Xeon Blade and SiCortex 1456 systems.

1U Xeon based machine

Setup

In this section, we will present the results gathered for machines equipped different Intel Xeon processors. The tested systems were: dual core Xeon 5130, 3.0 GHz (Actina SOLAR 212 X2), quad core 5345, 2.33 GHz (Actina SOLAR 410 S2), and quad core Xeon 5520, 2.27 GHz server. All systems were equipped with InfiniBand cards and two 10 kRPM sata HDD. The systems were tested with 4, 8 and 16 GB of DDR2 and DDR3 memory. Additionally we have attached to the tested systems dedicated expansion cards with accelerators ClearSpeed e620 and nVIDIA Tesla.

Results

Power Off

All tested machines consumed surprisingly large amounts of power just being plugged in. Table 18 presents the power consumption of powered off machines.

System	Power [W]
Xeon 5130 (2×2 cores)	16
Xeon 5345 (2×4cores)	44

Itanium 2 (1 core, 2CPU)	17
Xeon + Tesla cards	40
Xeon 5520 (2×4 cores)	10

Table 18: Power consumption of powered off servers

The energy consumption of powered off servers varies between 4 % and 10 % of a fully loaded machine. As a conclusion, we can say that (cluster) power management system should be able to physically cut off the power supply in order to maximize power savings on unused nodes.

Power On and Full Load

Figure 101 presents overall power consumption of tested machines. This does not include the SiCortex machine because the hardware configuration makes the comparison impossible. All other machines are dual CPU servers in 1U or blade enclosure. Among the x86 families, one can observe that the power consumption of entire server is decreasing with each new generation of the servers and the computational power offered by each new generation of the processors is increasing. The computation power is gained by increasing the number of cores rather than the efficiency of a single core and this trend will continue in the next few years.

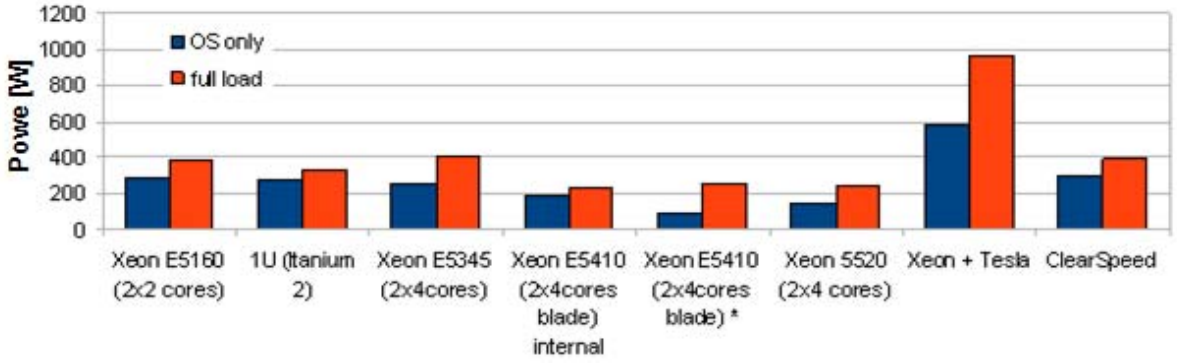

Figure 101: Summary of Power consumption of tested servers

Figure 102 shows the results of the NAMD benchmark for all machines with running on different numbers of cores. The ClearSpeed system was omitted because the NAMD application was not able to use the accelerator features. Not all tests are performed for 4 and 8 cores because some machines were single or dual core machines and we were assessing the single physical servers.

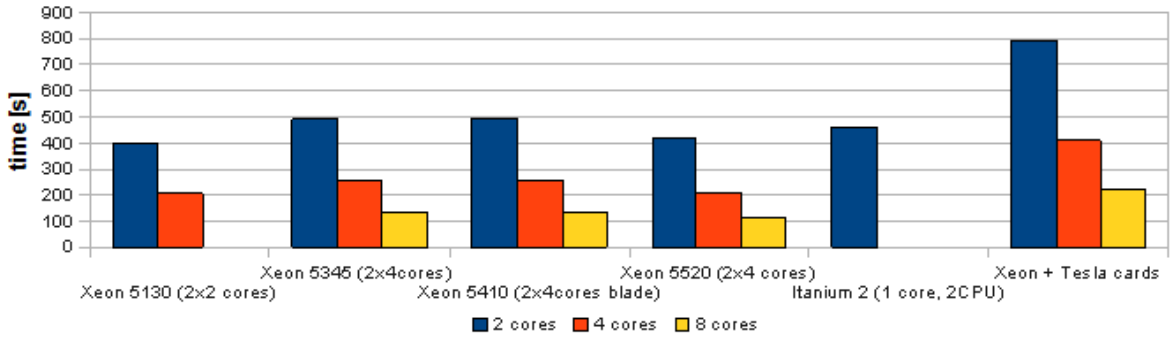

Figure 102: NAMD execution times

It can be observed that basically the performance of single core does not change significantly; even the outdated Itanium 2 CPU performs quite well. It can be observed that the performance advantage is gained mostly by increasing the number of cores. The very bad result of the Xeon server equipped with the Tesla cards where NAMD application works slower with ac-

celeration enabled is caused by inclusion the CUDA code during the compilation. When we look at the energy required by the systems to complete the benchmark (Figure 103), we can observe that the effective energy required for completing the benchmark decreases with each new generation of Intel processors.

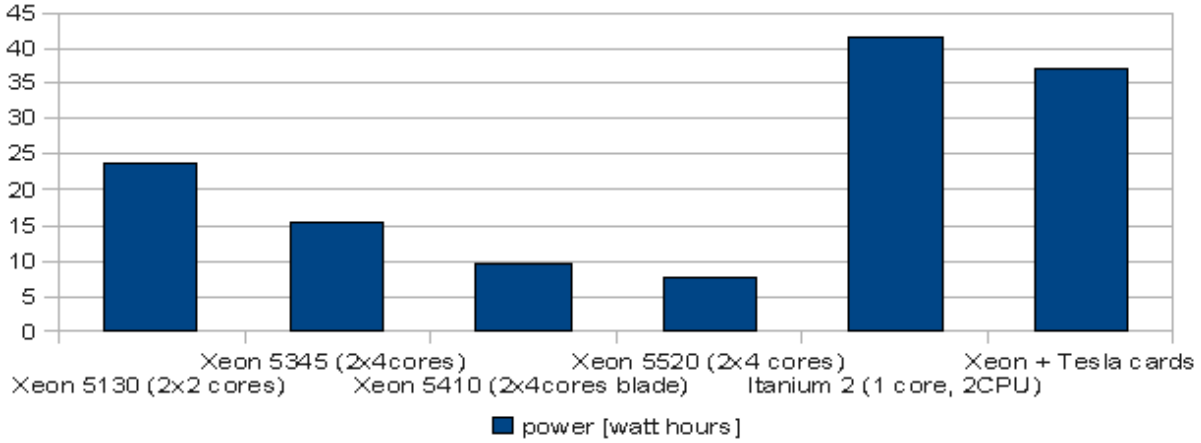

Figure 103: Power consumption - NAMD appoa1 benchmark

Apart from general power assessment we were also able to identify the power consumption of individual parts of the servers. The power consumption profile of the fully loaded Xeon 5130 can be found in Table 19.

Component	4 DIMMS	8 DIMMS
CPU	40.99 %	34.18 %
Memory	19.93 %	33.23 %
Fan	7.36 %	6.14 %
Hard Disk Drive	2.45 %	2.05 %
Ethernet	2.06 %	1.72 %
Infiniband	2.45 %	2.05 %
Controller	3.68 %	3.07 %
PSU	21.07 %	17.57 %

Table 19: Xeon 5130 server - internal power consumption

It can be observed that the most power demanding elements are the CPU, memory and the Power Supply Unit (PSU). For the 55xx system, the overall power consumption was much lower, mostly due to the more energy efficient CPU and DDR3 memory, though the ratio was very much similar.

Xeon Blade

Setup

The tested system was HP ProLiant BL2x220c G5 equipped with two Intel Xeon 5410 CPUs with 16 GB of memory and clocked at 2.33 GHz.

Results

Due to the construction of the blade server and blade center chassis it is very difficult to use external measurement tools. However, the monitoring system provided with the chassis is

able to track the power consumption of components building up the chassis and every single blade. While the measurements of the whole chassis were consistent with the measurements from our power meters, the readings for power consumption of each blade could not be trusted. According to the management system, the power consumption between idle and fully loaded blade was less than 20 W which is in conflict with the values obtained from 1U system equipped with similar CPUs. As a result we have decided to simply measure the power consumption of the entire chassis and divide it by the number of blades. Even this rough approximation proved that the blade solution is more energy-efficient enclosure than the 1U server (see Figure 102 and Figure 103).

SiCortex 1458 systems

Setup

The tested server was a mid-range system SC 1458 equipped with 243 6-core processors and 2 TB of memory. The processors are 60 MHz MIPS compatible CPUs with a power draw of 2 W per CPU. The processors were installed in 243 nodes with two memory sockets per node. The communication between nodes was implemented using dedicated high performance SiCortex network in Kautz topology. The machine was running a MIPS version of Linux kernel with GNU and PATHSCALE compilers and MPI libraries for communication.

Methods

As the SiCortex machine was accessed remotely, it was impossible to perform exact measurements. The access to the server was not exclusive so it was hard to use all processors. The maximum partition size was 1024 CPUs, a little more than 2/3 of the entire machine. Because of the financial problems of the SiCortex Company, the tests were not completed and only GROMACS was compiled and executed successfully. The power consumption is an estimation based on the data provided by SiCortex scaled down by actual number of processors utilized.

Results

In the Figure 104, we present the results of the GROMACS application on the SiCortex 1458 machine.

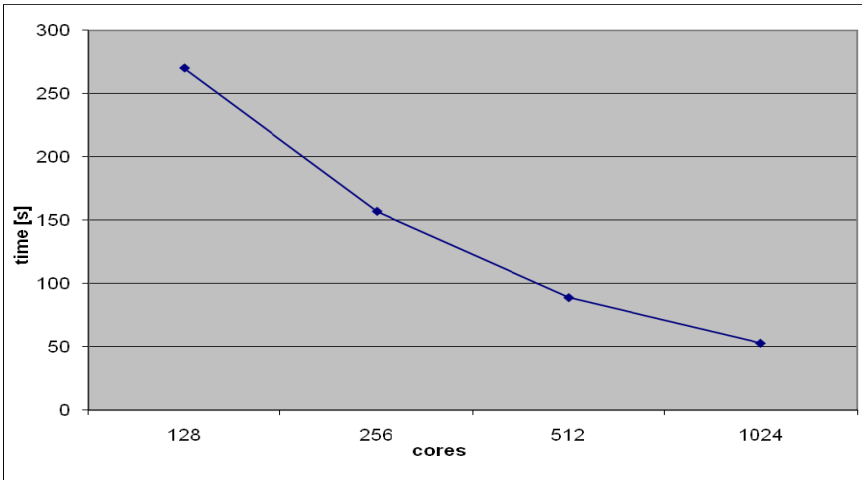

Figure 104: Gromacs run time on SiCortex

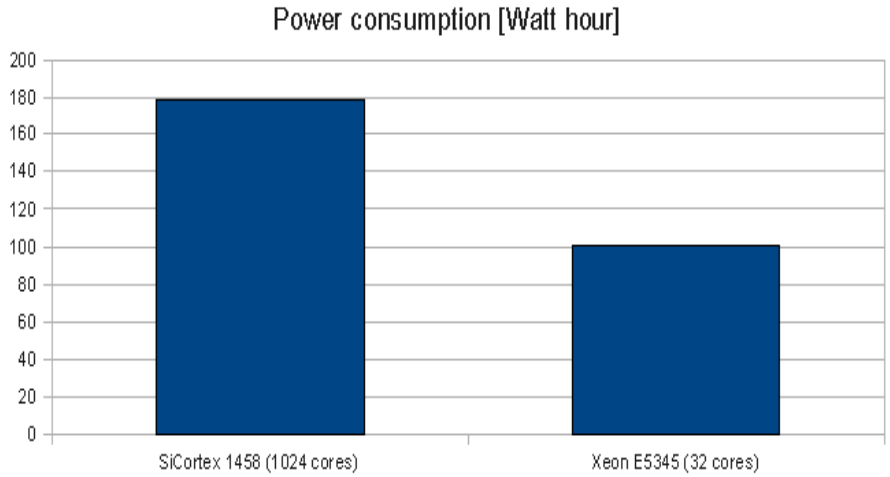

Figure 105: Power consumption of SiCortex and Xeon E5345

It can be observed that the application scales well and almost linear speedup can be achieved.

To assess the power efficiency of the machine we have compared the results with a x86 setup featuring similar performance. We have chosen 4 dual core Intel servers to act as a reference setup because of similar run times of the GROMACS benchmark. The Figure 105 shows the energy required by both systems to finish the benchmark.

It can be observed that the SiCortex machine, despite being advertised as power efficient, requires more energy to complete the work. Perhaps using different compiler or tweaking the placement of the processes on nodes could affect the result but due to the short availability of the machine no further tests were performed.

Conclusion

Results of the various tests indicate that in the area of clusters build of commodity components the power/performance ratio of the machines is increasing with each new generation of the systems. The power efficiency of the servers is increased by re-designing the most important parts of the machine – the processors and the memory. The energy losses introduced by the PSU are minimized by implementing a blade architecture which proved to be superior in terms of energy efficiency over the standard rack-mountable enclosure; however the most recent implementation of the 1U server x86 architecture (Intel Nehalem) matches the efficiency of the blade solution.

The accelerators included in this test (ClearSpeed and nVIDIA Tesla) have great potential for improving the power efficiency because of great theoretical power/performance ratio. The tests however, proved that using these solutions may cause opposite effect – non-benchmark applications rarely showed any significant speedup while the power consumption increased. This affects the nVIDIA solution in a great degree because the GPU-based accelerators are very power-demanding even in idle state.

In order to have a better picture of the problem it is needed to perform similar research on AMD based servers and take into account custom-build machines based on non-x86 architectures – e.g. POWER7. Another step is to take into account the whole environment of a HPC machine including the interconnect components, storage, backup, power supply and cooling.

Clearspeed e710

Setup:

The hardware used in this test was comprised of a Fujitsu server in the following configuration:

- Enclosure size: 2U;
- Power Supply; 2×750 W redundant PSU;
- CPU: 2× Quad core Intel Xeon E5520 at 2.27 GHz;
- Memory: 16GB of DDR3 memory in 4×4 GB modules;
- Storage: 6 hard drives (2×15 k RPM and 4×5.4 k RPM);
- Accelerators: 4×ClearSpeed e710 cards;
- OS: CentOS 5.4 64-bit operating system with kernel 2.6.18-164.15.1.e15.

The server in not a typical server used for building clusters - 2U enclosure and redundant power oversized PSU are making it less efficient in terms of energy. However, due to the need of having several 8×PCIe slots, it was necessary to use a more expandable server.

Method:

Currently, the only test executed is a double precision matrix-matrix multiplication, similar to the *mod2am* euroben. The multiplication was performed using a *dgemm* function from the BLAS library provided by ClearSpeed. The version of the drivers and libraries were 3.1. The test application was compiled using the *Intel* compiler *icc,* version 11.1 and, since the ClearSpeed-provided library does not implement all BLAS functions, the *Intel MKL* v. 10.2.5.035 was also used.

The testing procedure was as follows: several different instances of matrices were used of sizes varying from 1000×1000 to 6000×6000 elements). Since the ClearSpeed BLAS library has an interesting feature that allows sharing the load between the installed accelerator cards and the CPUs, we executed the test program several times, gradually increasing the load of the system CPUs. All the tests started using purely the accelerators (the *CS_BLAS_HOST_ASSIST_PERCENTAGE* environment variable set to 0) and ended using only the CPUs (the CS_*BLAS_HOST_ASSIST_PERCENTAGE* environment variable set to 100). Because we have the access to four ClearSpeed e710 cards, all the tests were repeated with one, two, three and four-card configurations. In Figure 106 we present the performance of the cards in these four experiments.

Results:

The results of the test are shown in Figure 106.

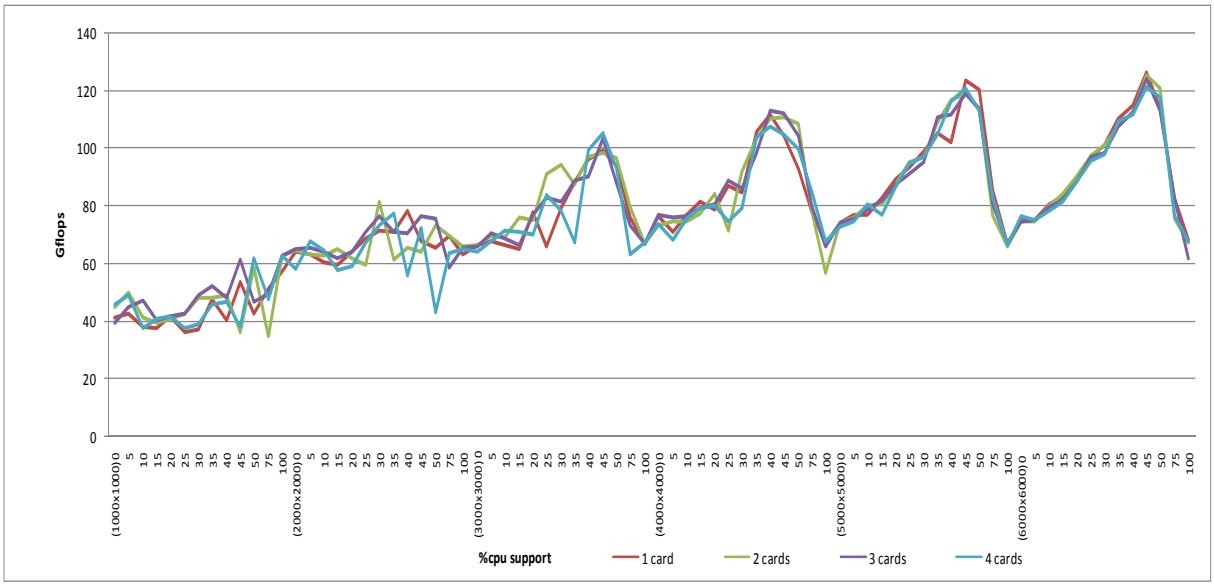

Figure 106: Performance reported by the dgemm application

A significant factor influencing the performance is the size of the matrix. The performance increases along with the growing size of the problem. However, if we start from the 5000×5000 matrix, we do not observe any further performance gains. This was proved by tests with even bigger instances, which are not shown here. In Figure 107 and Figure 108 we present the subsections of Figure 106 showing the performance of the matrix multiplication for the best and the worst case. For the problem instances smaller than 1000×1000 elements, the performance of the e710 cards is significantly lower than running the same calculations on the CPU.

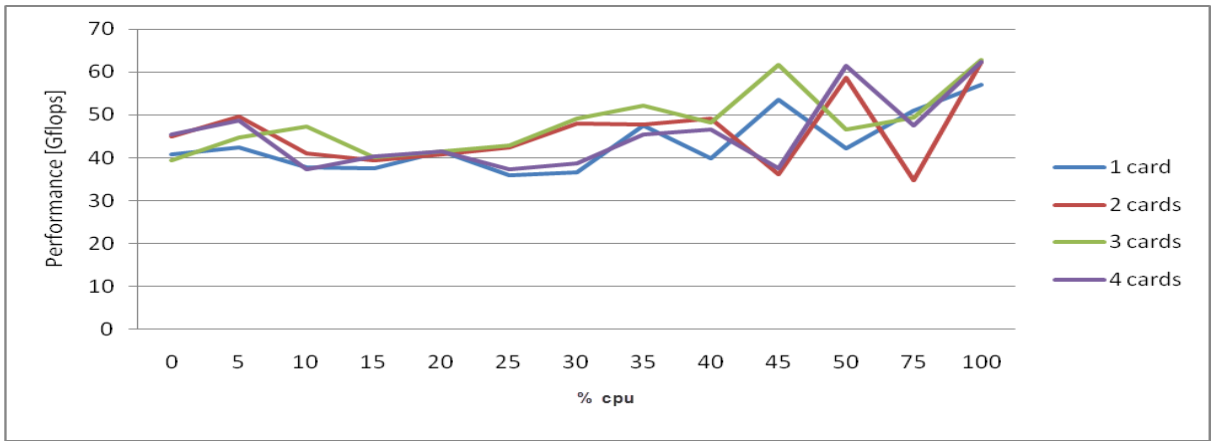

Figure 107: Performance for the 1k x 1k matrix multiplication (dgemm)

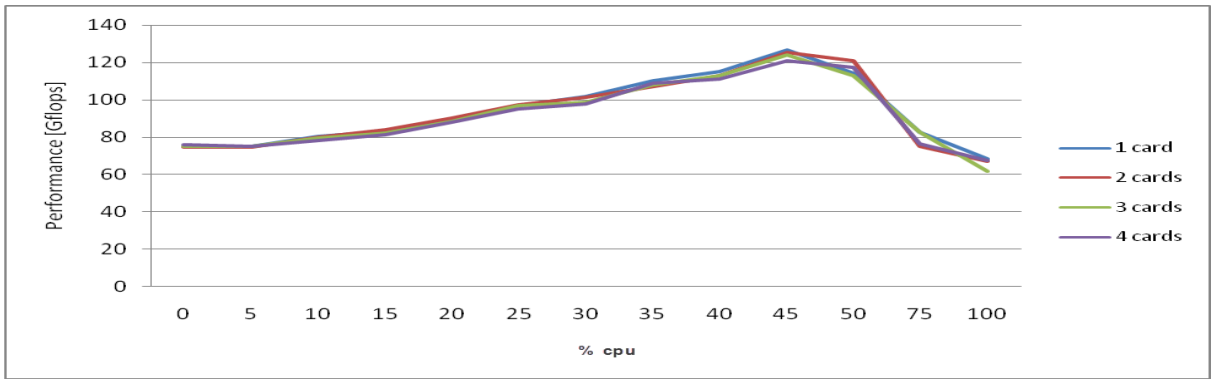

Figure 108: Performance for the 6k x 6k matrix multiplication

It can be observed, however, that the performance of the four-card configuration does not differ significantly from the one-card configuration, especially in the case of larger problems. The reason for this may be a weak support for spanning the calculations across multiple cards. The readings from the *power_mon* function confirm that the implementation of the BLAS library provided by the ClearSpeed does not support the multi-card matrix multiplications, as only one of the cards reports an increased power consumption and, therefore, only one card is used for the computations. It was, however, possible to run several instances of the test application to load all the cards. The performance measured for a single card without the support of the server's CPU is 75 GFlop/s, which is lower than the theoretical peak performance of 96 GFlop/s. It is worth noticing, however, that even a single card is faster, in case of the large matrix multiplication, than a pair of quad-core Xeon 5520 processors. This fact, connected with the reported power consumption on the level of 17W makes the ClearSpeed really power efficient.

For the instances larger than 4000×4000 elements, the maximum performance is achieved with the CPU support set to 45 %, which roughly corresponds to the performance ratio between the CPU and the e710.

During the computations with the CPU support set to 0 %, which theoretically means that the CPU is not used at all, the CPUs are still used (Figure 109), but on average the load is about 7% (this percentage is calculated for all 8 cores in the system) and it may be caused by, for example, the memory transfers.

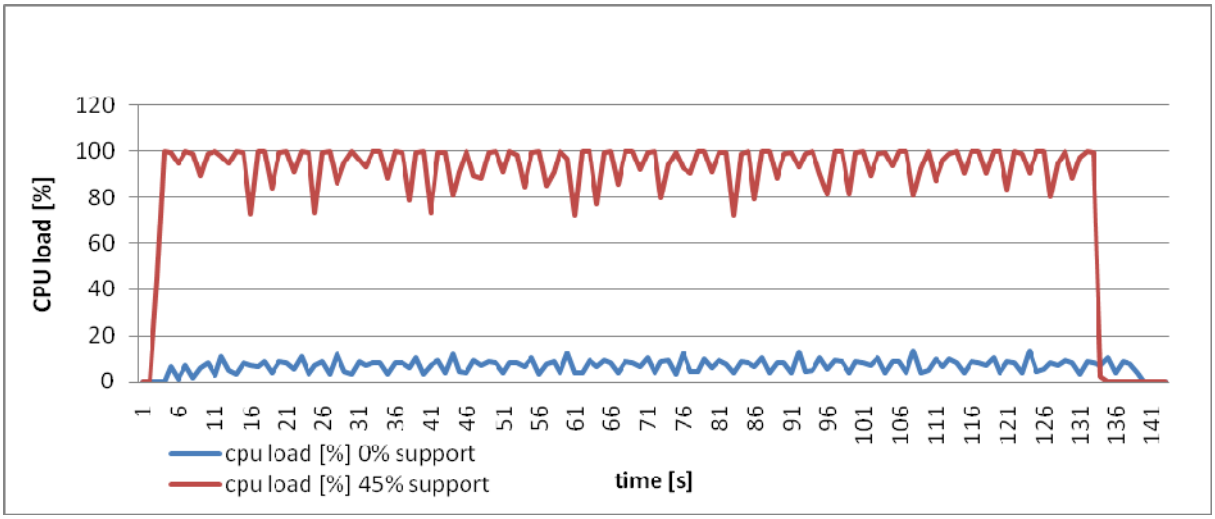

Figure 109: The CPU load while running the four instances of the dgemm test

The power consumption of the server was measured using both the internal measurement of the e710 cards and the external power meter measuring the power draw of the whole machine. By comparing the values, we were able to assess the accuracy of the internal power meters.

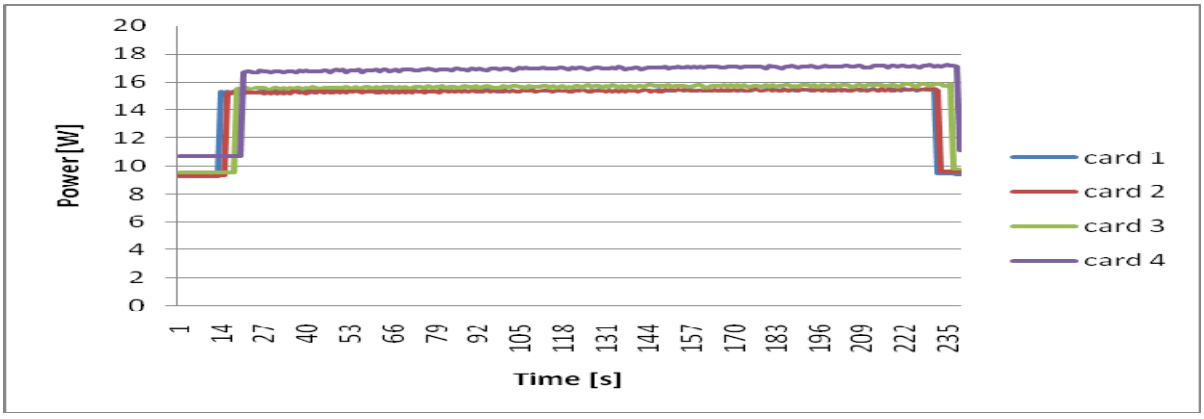

Figure 110: Power consumption reported by the cards while running a single dgemm application

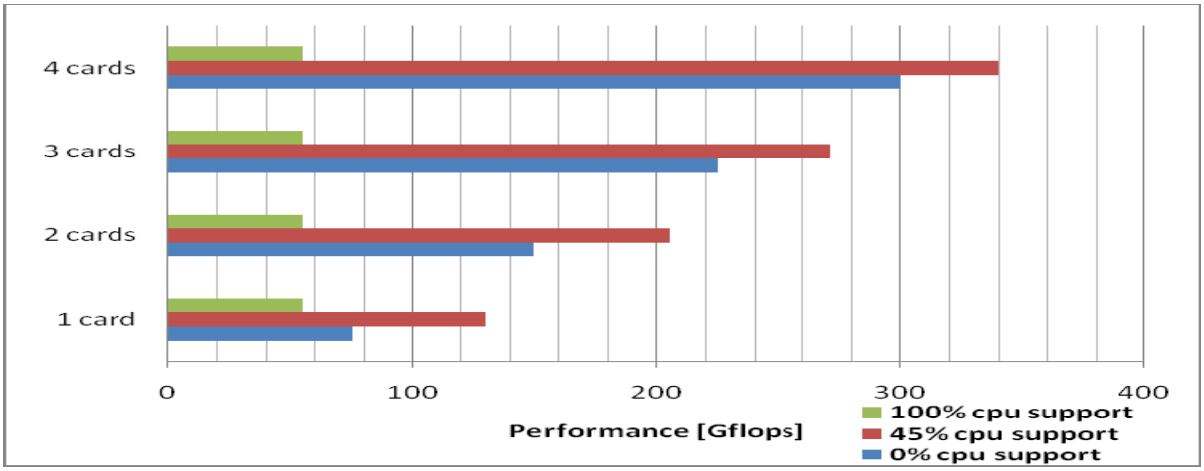

Figure 111: Computational performance

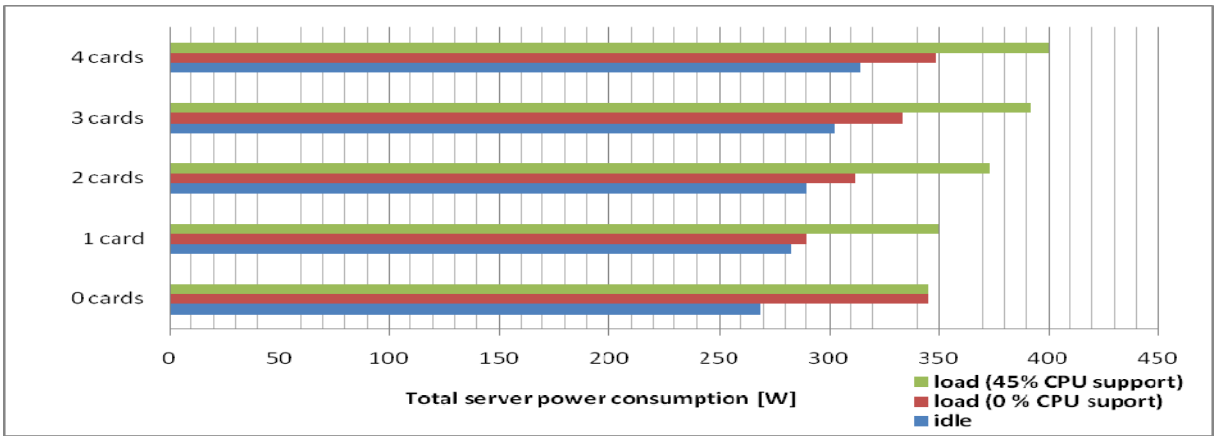

Figure 112: Server power consumption

The *power_mon* tool provided together with the cards reports (see Figure 110) a little lower power consumption than the relative power consumption increase measured with the power meter (Figure 112). The difference, however, is not very significant (about 1-3 W per card) and can be explained by the additional power loss caused by the power supply of the server, since the external meter was measuring the AC draw and the internal meters were measuring the DC power consumption. In most cases a single card consumes between 15.5 and 16 W while performing computations and between 9.5 and 10.5 W when idle. One of the cards consumed about 1.5 W more than the rest, but it is still in the power range specified by the manufacturer.

Conclusion

The cards offer an exceptionally good power efficiency comparing with both GPU and CPU (at least comparing to Intel Xeon servers) while running tasks. By adding less than 20W to the maximum power consumption, which is less than 10% of the power consumed by the fully loaded most efficient Nehalem-EP blade servers, we can gain more than 100% computing power (Figure 111). As we can observe in Figure 113, by installing the e710 cards, it is possible to achieve more than fivefold improvement of the GFlops/Watt ratio, comparing to the CPU-only configuration. An important thing is the possibility of adding several cards to a single machine without any serious power and cooling problems, as the power requirements of the four cards are comparable to the power requirements of a single four core processor. One of the big problems that affect the energy efficiency of GPU processors in context of HPC computing is the high energy consumption in idle state. Building an efficient computing environment is a major challenge because in order to be more efficient than "traditional" cluster, one has to maintain a very high load on the cards which is not always possible. In contrast to GPU solutions, one does not have to have the ClearSpeed cards utilized 100% of time because of the low idle power consumption.

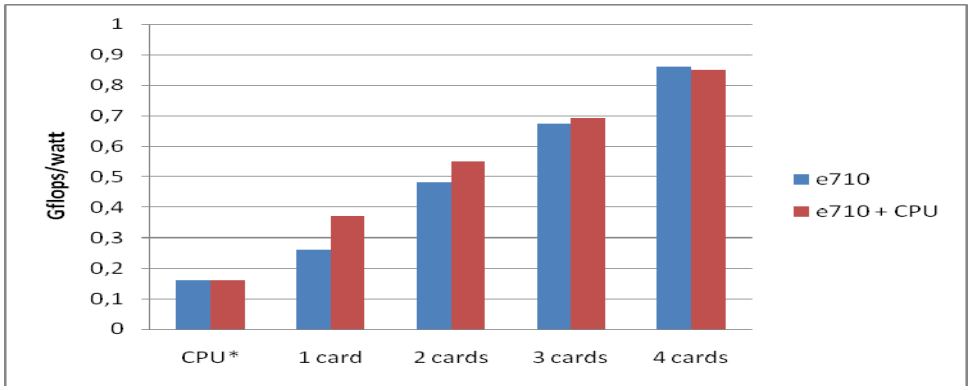

Figure 113: Performance/power ratio of the test server

3.7.3 STFC results

A series of experiments was conducted to look at the overall power consumed by servers in an HPC environment and more specifically the power requirements of nVIDIA Tesla enabled solutions. Towards this end, an energy profile logger SP Max 512 was employed for power measurements. It allowed not only measuring power at any given time but also observing the whole power profile during the execution of an application. The device requires connection to the mains to read voltage whereas current is measured using induction clamps. While three phase power measurements are also possible we limited our experiments to tests on a few servers only and therefore operated in single phase mode. The logger was placed inside of the server rack together with a laptop through which it was controlled remotely. The readings were taken in the smallest increments of 1 second at which rate the device is capable of storing up to 5 hours of data. The device offers true RMS measurements with the accuracy of +/– 0.25 % of the range plus CT (Current Transformer) error. A 10 A CT input lead was used which therefore determined the accuracy.

Results

Idle Power

The idle power measured for single Nehalem-EP server was 230 W. It was found to be somewhat lower than the figures we measured on our Intel Harpertown (Xeon E5472) and Woodcrest (Xeon 5160) servers which feature a different configuration and in particular a higher processor clock frequency. These were 240 W and 270 W respectively. The idle regime of our server operation does not have any power saving features enabled because we assumed that in an HPC setting the servers are supposed to be heavy loaded most of the time and be more responsive therefore disabling energy saving features makes sense. The idle power of the host server together with an nVIDIA Tesla server attached to it was measured at about 380 W. However once the Tesla cards were activated the power went up to 570 W and stayed there. In other words, even if the Tesla cards are not used they are still dissipating up to 340 W in addition to whatever is being dissipated by the host. These results agree very well with the power measurements reported in the previous subsection by PSNC.

Loaded Power

The popular molecular dynamics code NAMD [20] was employed as a test application which can use both CPU and GPU. The code is well tested and scales very well. The current release at the time (version 2.7b1, 2009-03-23) was used in our benchmarking. Two tests were run: APOA1, the standard NAMD benchmark (model of a lipoprotein particle found in the bloodstream) and STMV (satellite tobacco mosaic virus). The former is the most commonly used benchmark because it is rather representative of a typical simulation. The test comprises 92 k atoms including lipid, protein and water. APOA1 benchmark is a moderately sized simulation suitable for long timescale studies. STMV test case is a larger benchmark comprising 1M atoms.

In agreement with PSNC results, it was initially found that unaltered inputs give no acceleration when using GPUs. A consultation with NAMD developers revealed that by default the energy of the system is computed every step and since it is done on CPU in double precision this mode operation is very inefficient. In fact, algorithms used in molecular dynamics simulations aim at retaining the energy exactly (symplectic integration) or approximately. In practical applications, the energy is seldom needed and used mostly for control purposes. Therefore the input files were modified by adding parameter “outputEnergies” and setting it to 100, i.e. the energies are computed only each 100^{th} step. With this in place, the speedup obtained on four Tesla GPUs versus four Nehalem-EP cores was 6.4 on APOA1 and 7.0 on STMV.

For power measurements, it was clearly necessary to fully load servers. In other words, we had to use 8 processes on a single Nehalem-EP server, 16 processes on two Nehalem-EP servers and only 4 processes on a Nehalem-EP server equipped with a Tesla server therefore making in the latter case four Nehalem-EP cores idle. Table 20 summarises of our results. Presented are the elapsed times in seconds, average power in Watts, total energy in kilo Joules and the ratios of these numbers for the two server scenario versus one server plus Tesla server. We observe that the speedup is almost a factor of two but the Tesla configuration takes more power. Therefore the overall energy saving is lower than the speedup. In real deployment, the economical advantage of using GPU accelerators is going to be a fine play between the cost of the hardware and the power requirements. For instance, blades typically take less power but they are also more expensive. On the other hand it is entirely possible to have a low power host which drives two Tesla servers. It is clear though that the often quoted GPU versus single CPU core speedup factors do not tell the whole story.

NAMD, STMV benchmark, 500 steps	1 Nehalem server	2 Nehalem servers	1 Nehalem server + 1 Tesla server	ratio
Elapsed Time /sec	1161	599	338	1.8
Avg Power /W	313	581	740	0.8
Total Energy /kJ	365	353	254	1.4

Table 20: NAMD, STMV benchmark, 500 steps

We were also successful in running the popular LINPACK benchmark [21]. The CUDA port was kindly provided by nVIDIA. The average power of a Nehalem-EP server running LINPACK was about the same as that of running NAMD whereas the power of Nehalem-EP server plus Tesla server increased to 810 W on average peaking as high as 1 kW. The difference with NAMD was that LINPACK is not only able to place heavy load on all Tesla servers, it also makes use of all the host cores through threaded parallelism offered by Intel MKL which explains the much higher power values than for NAMD. Also, unlike NAMD, LINPACK requires double precision but the theoretical peak performance of C1060 in DP is eight times lower than in SP.

Linpack	Nehalem server	Nehalem server + Tesla server	Intel server + ClearSpeed server
GFlop/J	0.24	0.27	0.2 *
* - an estimate based on not fully loaded server			

Table 21: LINPACK power efficiency on different architectures

Using suitably selected parameters, we were able to achieve 68 % efficiency on a single core and a single GPU (59.5 GFlop/s). The efficiency of LINPACK on the whole cluster (8 nodes, 3144 GFlop/s peak in DP) dropped to 52.5 % but the performance was still quite impressive achieving 1650 GFlop/s. In contrast, Intel processors are known to have very high LINPACK efficiency with the latest HPCC entries achieving as high as 96 %. Translating the efficiency into GFlop per Joule, we obtain 0.24 GFlop/J for our Nehalem processors and 0.27 GFlop/J for Nehalem plus Tesla. Although the Tesla delivers more GFlop/J, the difference is clearly rather small. The power efficiency numbers are summarized in Table 21. It is instructive to compare power consumption and LINPACK efficiency with that we measured on our ClearSpeed setup [22]. It comprised four Intel Woodcrest servers as hosts and two ClearSpeed CATS600 servers (i.e., 2:1 host to accelerator ratio). Firstly, the idle power of CATS600 was 470 W i.e. not as high as S1070 but also rather high. Loaded power was measured at 590 W which is also lower than S1070 but then the ClearSpeed e620 cards were meant to be low power. The efficiency running LINPACK was also around 50 % resulting in computation per

unit of work ~ 0.2 GFlop/J i.e., quite comparable to the numbers for Nehalem-EP and Tesla. It must be noted that the latest CATS700 are more power efficient; however the Tesla servers are much more affordable.

Conclusion

First of all, it is indeed possible to see a real benefit of using GPUs. nVIDIA Tesla S1070 provides an affordable hardware ready to be used in HPC clusters. Secondly, it is also very clear that the application speedup achieved on a GPU has to be really significant in order to be efficient in the production environment because the power dissipated by the host needs to be factored in. Finally it is important to be aware that the power consumed by the Tesla servers is significant even when they are idle. Therefore either the accelerator servers need to be busy all the time or there must be a mechanism to reduce the idle power to a much lower level. It must be noted however that, following our finding we were informed by nVIDIA engineers that the idle power issue will be rectified in the forthcoming driver.

3.7.4 *BAdW-LRZ results*

Intel's new processor micro-architecture codenamed Nehalem provides new power management features for all platform components: the processor, chipset, and memory. This allows operating systems to put processor power and memory into the lowest available states needed to support current workload without reducing application performance. Also individual cores of a node can be idled independent of the others.

Measurements on our SGI Altix ICE 8200LX systems show that this new power management feature reduces the power consumption of idle dual socket blades by 45 % when compared to fully loaded blades running the LINPACK benchmark. Running LINPACK on all 48 blades of the ICE system we could reach 91 % of the peak system performance. The memory-filling ratio was 2.58 out of 3 GB/s per core and hence 86 %. Measuring the power consumption for the LINPACK run showed a power efficiency of 230 MFlop/s per Watt for this system.

Since at the time of writing, Nehalem-EX processors were not officially released by Intel we were not allowed to report detailed performance and power efficiency results for this processor line.

3.8 Performance predictions

In this section, we apply the analysis tools and techniques described in Subsection 2.2.4 to one of the benchmark applications used in WP5 to compare prototypes. The objective is to identify issues that may be relevant when scaling applications or porting them to future architectures. Although only one application is analysed in detail, the experiences match to those of many other partial studies done during the last year in WP8. The selected application is GADGET and we use the data set "inputA".

3.8.1 *Impact of basic system components*

A bunch of parametric simulations with Dimemas were performed in order to explore the impact of four major components in the total performance:

- Processor/node performance;
- Network Latency;
- Network Bandwidth;
- Contention.

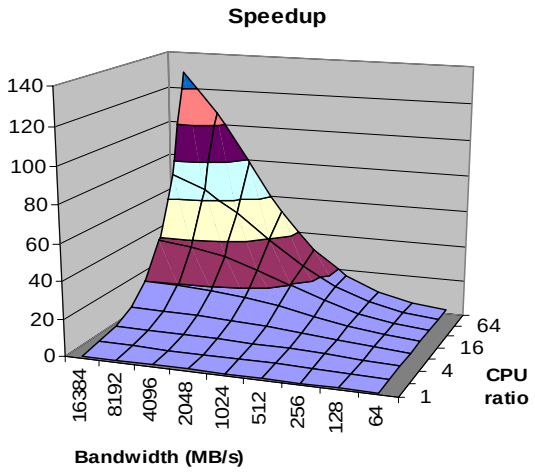

Figure 114: Impact of node performance and Interconnect bandwidth (in MB/s) for 64 processes

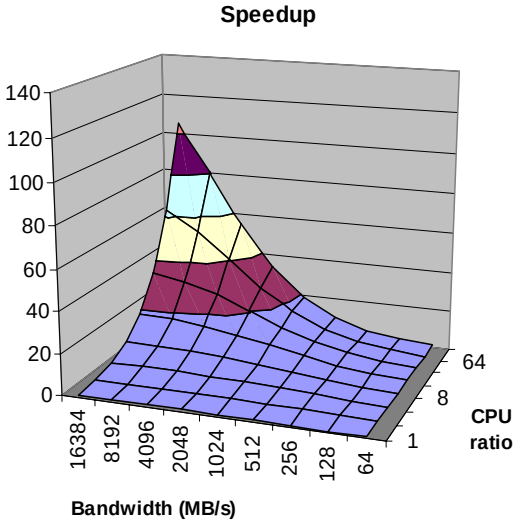

Figure 115: Impact of node performance and Interconnect bandwidth (in MB/s) for 128 processes

Figure 114 shows the speedup achievable when changing the node performance (CPUratio representing acceleration of all sequential computations between MPI calls) and bandwidth. The node performance factor represents the speed that every sequential computation phase between MPI calls is assumed to have relative to a PPC970. Such node speedup may be achieved on a hypothetical machine by using a faster core, assigning the computations to accelerators or parallelising with OpenMP on standard multicores. The application speedup reported in the figure is computed taking as reference the PPC770 node performance and a network of 256 MB/s. The figure is computed assuming that there is no contention obtained by performing additional Dimemas simulations. It can be roughly estimated by assuming that the effect of contention is a reduction of the effective bandwidth.

Overall, the curve shows how the network performance should improve when accelerating the node performance. For example, just improving the network bandwidth by 4× would result in a mince speedup of 1.25. Accelerating the node performance by 16× (using a faster processor, a multicore using OpenMP within the node or an accelerator and assuming every single piece of computation is accelerated by the same ratio) would result in just a 6.5× speedup if keeping the same network, 9.9× if doubling the network bandwidth or 13.5× if increasing the network bandwidth by 4× times.

The hypothesis that all computation bursts between MPI call achieve the same acceleration is also very coarse. In a real system, most probably only relatively large sequential computation phases would be ported to an accelerator or parallelized with OpenMP as complexity and/or parallelisation overheads may discourage a programmer from parallelising some of the very short bursts. The result would be less global speedup than reported in the figures. An analysis of this effect will be shown in Subsection 3.8.2.

Figure 115 and Figure 116 show the same results for 128 and 256 processors. An interesting result is that the execution with less processes benefits more from a given increase in node performance and communication bandwidth. This is somewhat natural as we assume that the node parallelisation is ideal. Subsection 3.8.2 shows the behaviour when not such an ideal parallelisation at the node level is made.

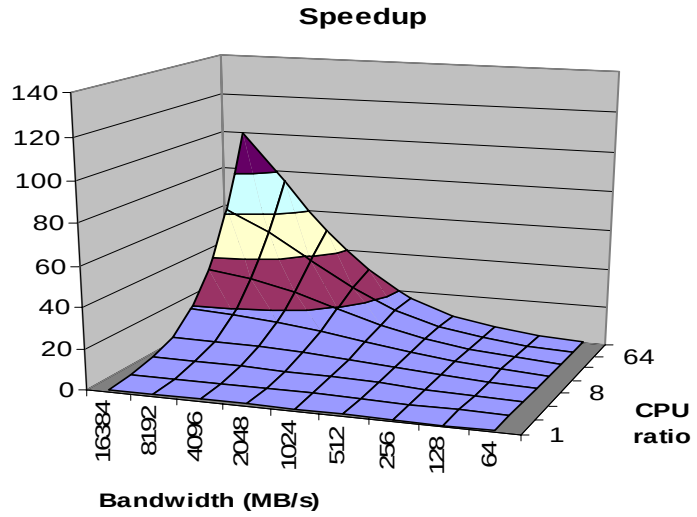

Figure 116: Impact of node performance and Interconnect bandwidth (in MB/s) for 256 processes

3.8.2 *Non ideal node level parallelisation*

The previous figures were computed assuming that every single computation burst between MPI calls is accelerated by the specified CPUratio factor. The typical optimisation process of an existing MPI application will nevertheless focus on the more computationally expensive regions as identified by a profile. Either using accelerators or introducing OpenMP, a partial optimisation will be done, leaving the less time consuming regions unmodified. It is also possible that some speedup may be obtained on those regions by using a somewhat faster processor (i.e. better memory bandwidth).

In this section we study the actual impact on total application performance assuming that a global 75 % improvement in performance (1.75 speedup) can be achieved on all computation bursts and then some of the most intensive computation regions are sped up by an additional factor between 1 and 128. We apply the study to the trace of 128 processes. The results should then be compared to those in Figure 117.

By applying clustering techniques to the trace of 128 processes we are able to identify the 13 more relevant computation regions. Figure 118 shows the percentage of the total computation time that each of these regions represents. A typical optimisation practice would start porting to an accelerator or parallel node level programming model the most expensive regions of code. Three plots of the predicted global speedup are presented in Figure 118 to Figure 120 assuming that we optimize the first 6, 9 and 13 computation regions respectively. In turn, they correspond to accelerating a total of 93.67 %, 97.49 % and 99.11 % of the computation time. In all the figures the CPUratio dimension represents by how much we assume the optimized

parts are accelerated. The Bandwidth dimension represents in MB/s the assumed bandwidth in the network.

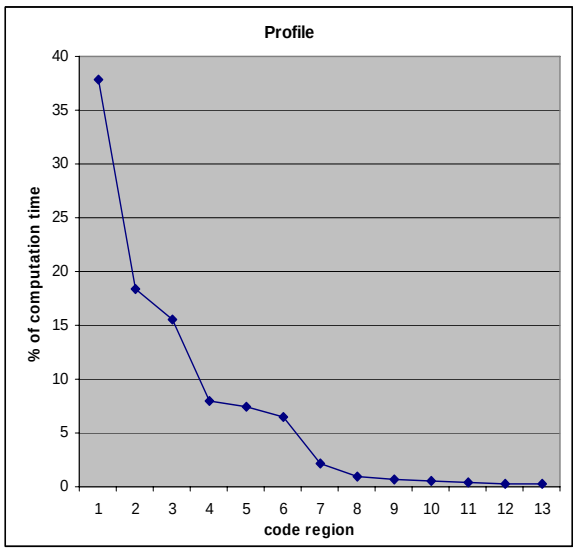

Figure 117: Profile of computation regions in GADGET for 128 processors in MareNostrum

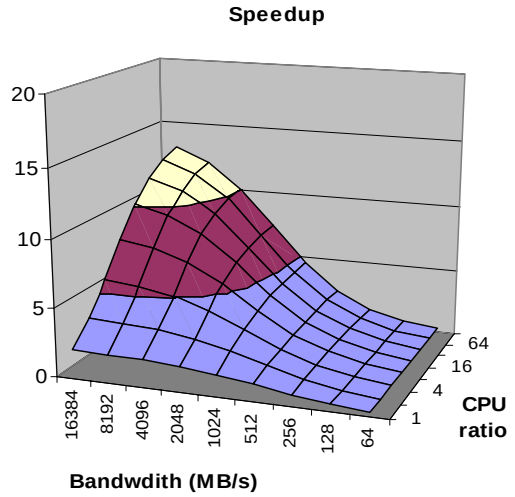

Figure 118: Impact of bandwidth and acceleration factor applied to major computation bursts representing 93.67 % of the original computation time

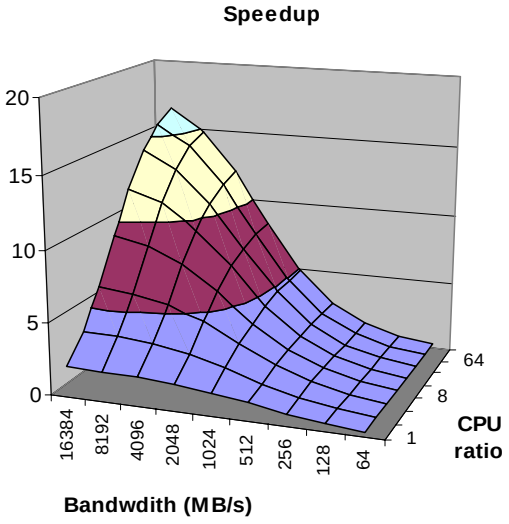

Figure 119: Impact of bandwidth and acceleration factor applied to major computation bursts representing 97.49 % of the original computation time

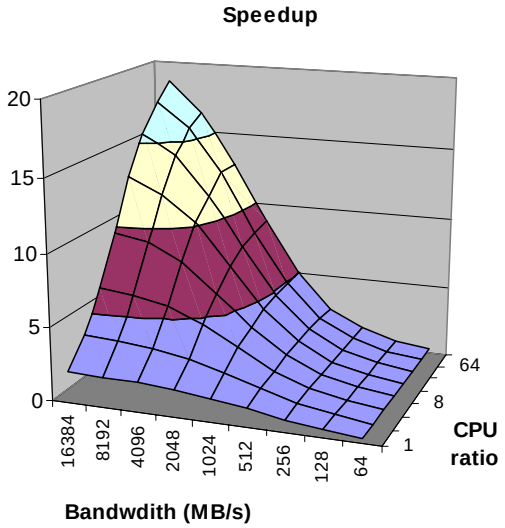

Figure 120: Impact of bandwidth and acceleration factor applied to major computation bursts representing 99.11 % of the original computation time

As can be seen in Figure 120 the speedups are much less than in Figure 116 even if Figure 120 assumes that 99.11 % of the code is accelerated. The results in Figure 119 and Figure 120 correspond to cases with smaller percentages of the original code accelerated and show that very limited total speedups are achieved even if very good accelerations are achieved on the optimized regions.

In this model, the impact of network bandwidth is also important. As we said, Figure 120 shows data when 99.11 % of the computation time is accelerated, but because communication does represent a non negligible part of the total elapsed time and unless it is also accelerated, the total performance stays very low. For example, if only 512 MB/s links are used, there is not big difference in accelerating the computation parts by 4 or by 128. For bandwidths of 2

GB/s, going from 4× acceleration to 128× in the selected sequential parts would only result in a 2× global speedup.

These results show that hybrid parallelization where MPI is combined with node level programming models with synchronous (fork-join or data parallel) structure does have important scalability limitations deriving from Amdahl's law. If very fat nodes (or powerful accelerators) are used, it will be necessary to parallelize a very large fraction of the computations between MPI calls. This actually implies that a continued optimisation effort may be needed as more and more performing multicore nodes are adopted. Such perspective highlights the importance of exploring alternative approaches relying on asynchrony and overlap between communication and computation.

3.8.3 *Prediction for ICE*

We thus apply the above numbers to the traces of 64, 128 and 256 to make a prediction for the ICE at those sizes. We can compare the Dimemas predicted duration for each iteration with its corresponding actual run on the ICE system.

Processor	Time (s)		Error (%)
	Measured	Predicted	
64	40.41	38.71	4 %
128	21.65	19.06	12 %
256	12.15	10.73	12 %

Table 22: Prediction and error with respect to actual iteration time on the ICE prototype

The Dimemas prediction underestimates the real iteration duration. We can further analyse the errors in the case of 64 processors using Figure 121. We present a timeline of the MPI calls in the real run and a view of the Dimemas prediction for one iteration. There is one row per process and light blue means computing while different colours correspond to different MPI calls.

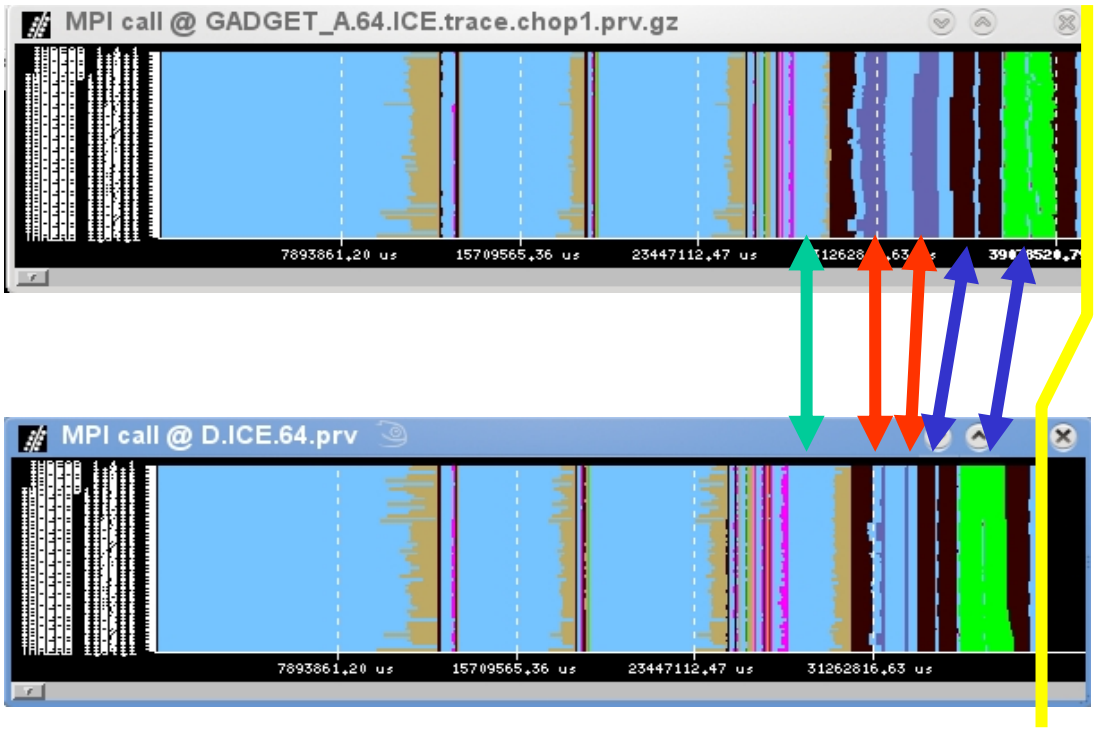

Figure 121: MPI calls for real run (top) and prediction (bottom) for 128 processes on ICE

Both figures are in the same time scale. The total difference in total elapsed time between the reality and the prediction is highlighted by the yellow line and corresponds to 4 % as indicated in Table 22. The major substantial difference is in the duration of two Alltoalls that according to Dimemas should be significantly faster than what they are in reality. The red arrows point out the regions with differences. The blue arrows show that the approximation of the simulation for the point to point exchange phases (Sendrecvs in brown, Isends and Irecvs not visible because of the scale and Waitalls in green) is sufficiently accurate.

The green arrow points to another region where there is some deviation in the sequential performance from the average compute ratio of 1.75. It is clear that different regions of sequential computation will perform with different efficiencies on cores with different architecture and memory bandwidth. The factor of 1.75 is an average acceleration applied to all computation regions in our experiment. More detailed accelerator factors for each region could be used to achieve better prediction.

The views for 128 processes are given in Figure 122. In this case the total error in the prediction is larger (12 %) and the same effect of duration of the Alltoalls appears (red arrows). In this case, the real ICE trace does show a potentially additional problem in the Alltoalls (red circle). Some of the invocations to the first Alltoall get delayed while others have already exited the collective call. This is related to progress issues in the internals of the MPI implementation.

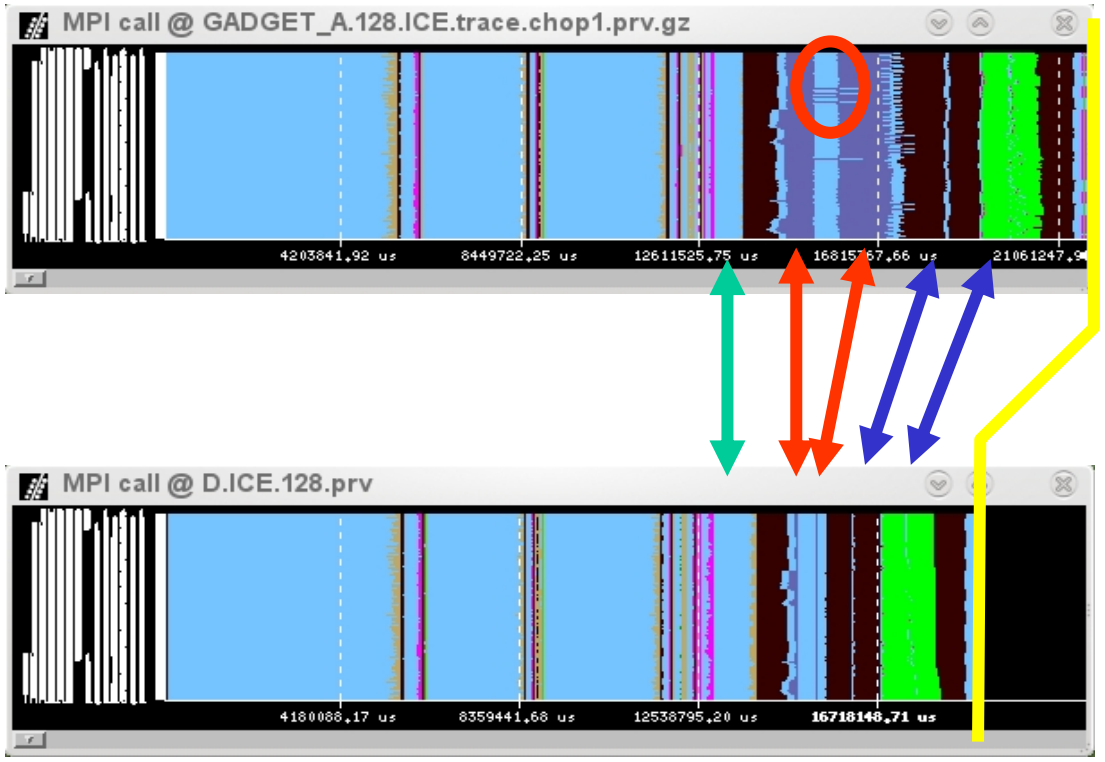

Figure 122: MPI calls for real run (top) and prediction (bottom) for 128 processes on ICE

3.8.4 *Prediction for BG/P*

A similar study has been done to predict the behaviour on the Blue Gene/P prototype of WP5.

Processor	Time (s)		Error (%)
	Measured	Predicted	
64	250.41	258.60	3 %
128	131.78	123.13	7 %

256	79.10	63.82	9 %

Table 23: Prediction and error with respect to actual iteration time on the Jugene prototype

The results show that the error for 256 processors is relatively large, motivating a detailed analysis. Figure 123 compares the real run and the prediction for one iteration. Three issues deserve special comments.

First, the Sendrecs and in this case also the Alltoalls are decently matched by the prediction. The major conclusion out of this comparison relate to our previous analysis of the ICE platform. We tend to consider that the linear model for the Alltoall used in the Dimemas simulations is a fair approximation of what the actual duration of the Alltoall should be, and the BG/P prediction seems to agree with that. Our interpretation is that there are issues in the ICE implementation of the Alltoall that should be looked at, and that the achievable speedups in that machine could be slightly improved.

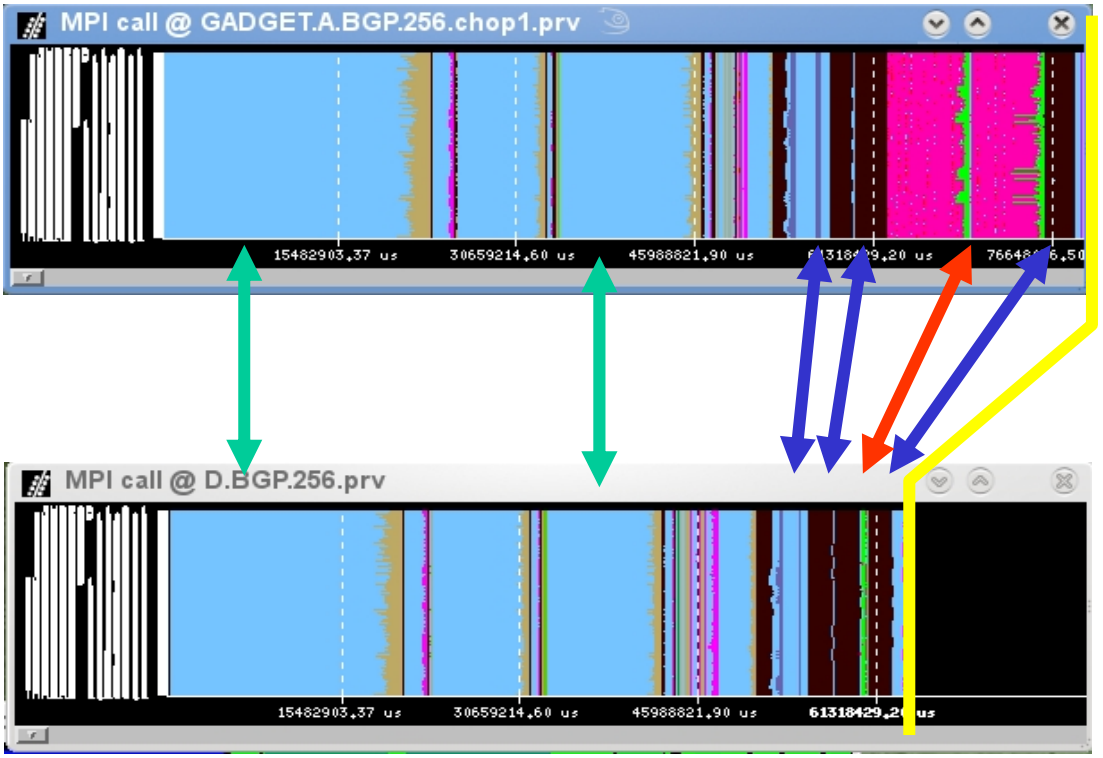

Figure 123: MPI calls for real run (top) and prediction (bottom) for 128 processes on Jugene

Second, the predicted duration of the computation phases is slightly different in the simulation than in the reality. In fact, Figure 124 shows the histogram of the duration of the major computation phases. We see that some areas are underestimated and others overestimated with error in the order of 10-15 %. This shows how more precise characterisation of the change in scalar performance when moving from one processor to another would be needed to be developed to improve the accuracy of performance analysis tools.

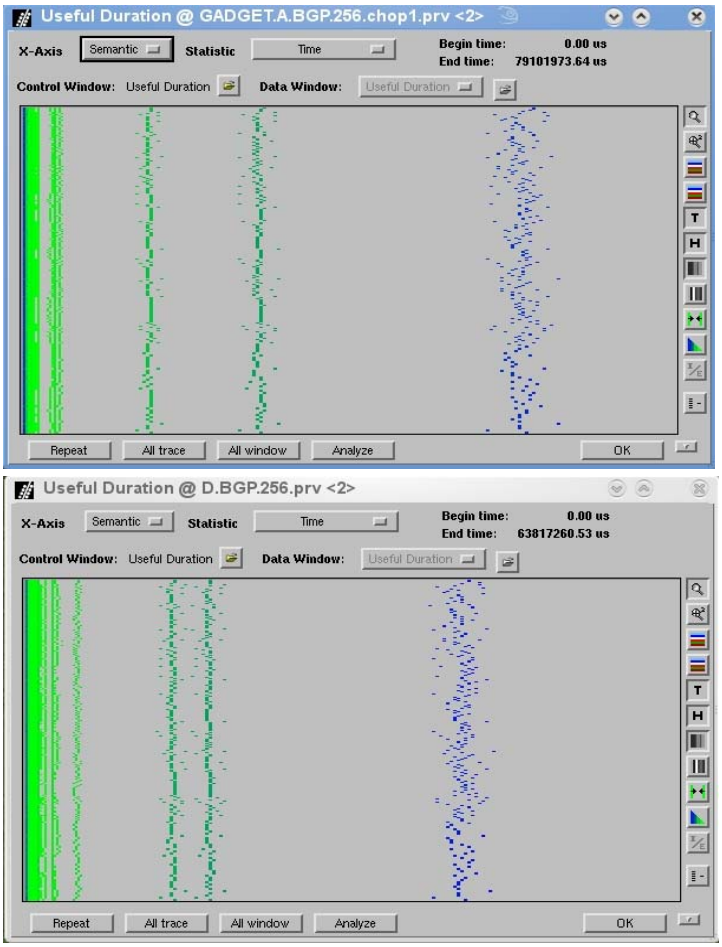

Figure 124: Histogram of duration of major computation bursts in the real run (on top) and prediction (bottom)

Third and most striking, the red region which corresponds to exchanges implemented by Isends and Irecvs followed by a Waitall (in green) are much larger in the reality than the prediction. A detailed view of that region is presented in Figure 125. The view on top shows the dark blue regions that correspond to significant computation that takes place intermixed between the Isends and Irecvs. These bursts of computation were not seen neither on the MareNostrum nor the ICE traces. They do not show up either on the BG/P run with 64 processes, but for 128 processors they have an average duration of 345 ms and 174 ms for the 256 processor run. Unfortunately, their number grows more than linearly with the number of processors. There are 1500 in the 128 processors run and 7700 in the 256 run. This region may be the cause for the limited scalability reported for this test case in Deliverable D5.4 [7]. Further detailed analysis of the source code and potential interactions with the BG/P MPI implementation should be done.

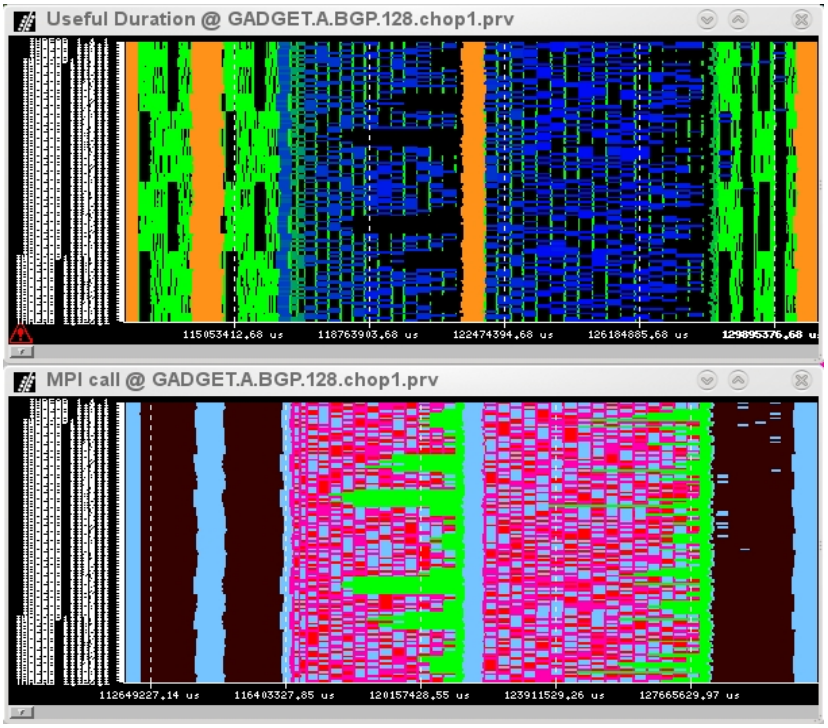

Figure 125: Zoom of duration of computation phases (top) and MPI calls (bottom) in exchange region

3.8.5 *General analysis*

We applied the model described in Subsection 2.2.4 to the different GADGET runs in MareNostrum, ICE and Jugene (BG/P). Results are given in Table 24. We highlighted in green factors where the performance loss is below 10 % (values higher than 0.9) and in red factors where the degradation is higher than 25 %.

Platform	Processors	Input	Iteration time (s)	Efficiency	LB	microLB	Transfer
MN	64	A	78.71	0.66	0.90	0.94	0.78
MN	128	A	44.22	0.58	0.97	0.92	0.65
MN	256	A	22.78	0.56	0.95	0.77	0.76
BGP	64	A	250.41	0.87	0.90	0.99	0.97
BGP	128	A	131.78	0.86	0.96	0.98	0.91
BGP	256	A	79.1	0.75	0.95	0.89	0.90
ICE	64	A	40.407	0.88	0.91	0.97	0.83
ICE	128	A	21.65	0.93	0.98	0.90	0.73
ICE	256	A	12.154	0.89	0.94	0.95	0.68

Table 24: Global performance model for GADGET

The column efficiency reports the global parallelisation efficiency (fraction of active processors over the whole iteration). The numbers show that the parallel efficiency is relatively low in MareNostrum and significantly better in the BG/P and ICE, but very seldom above 0.9.

The major responsible for the difference is that the ratio between communication performance and computation performance which is lower on MareNostrum. This is identified by the Transfer factor in Table 24, which shows much more degradation of the performance (about 25 % performance loss due to data transfers on MareNostrum versus around 10 % performance loss in the others).

The global load balance is fair, actually better with 128 and 256 processors than for 64. This factor is very dependent on the data set and processor count. For the problem sizes of both test cases used as PRACE benchmarks and the processor counts we have analysed the load bal-

ance is good. We should measure it at much larger processor counts as those used by WP5 for the GADGET performance values on BG/P reported in Deliverable D5.4 [7]. It is our experience that for other real runs in production at MareNostrum, the load balance factor was very low (< 0.25). The implication for the selection of future machines is the need for mechanisms to handle load balance in flexible ways, being able to enter into play for datasets of processor counts that require them and not disturbing cases where not required.

The micro load balance/serialization factor gets worse on MareNostrum and BG/P as the number of processes increases. Figure 126 represents the percentage of processors doing useful computation as a function of time for 64, 128 and 256 processors on ICE. The time axes are scaled so that for each processor count each figure represents the total iteration time. The traces are actually Dimemas predictions assuming no communication latency and instantaneous data transfers. This means that the drops in instantaneous efficiency observed are due to intrinsic application characteristics.

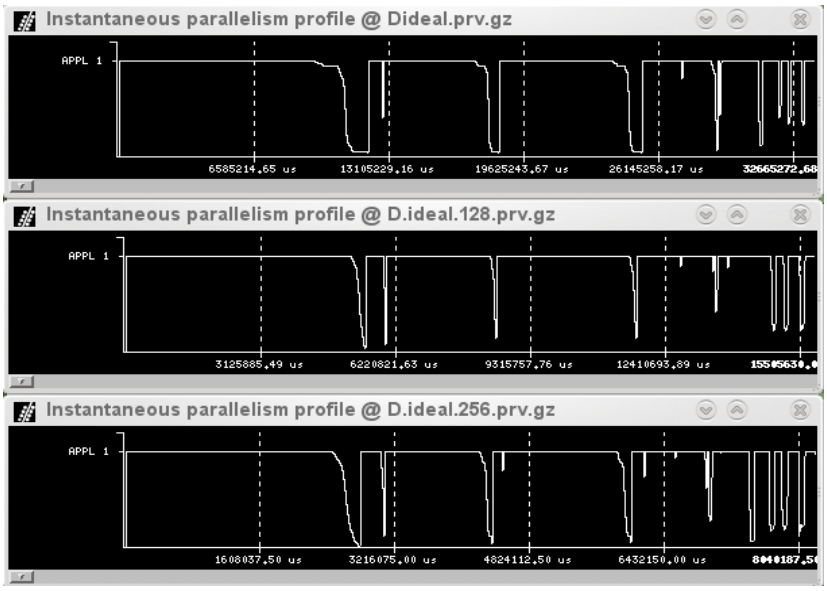

Figure 126: Instantaneous parallelism profile for 64, 128 and 266 processor runs on ICE

The drops observed during the first ¾ of the iteration are due to actual load imbalance. The loss of performance towards the end of the trace is due to serialisations in the communication phase towards the end of the iteration. A zoom of the behaviour in this phase for the 256 processor run is shown in Figure 127 along with the MPI calls. The serialization in the Sendrec phases (brown in the MPI Call timeline) may be intrinsic to the algorithm or may result from a specific schedule of the communications. The actual pattern of computations that get involved in this serialization process is shown in Figure 128 .

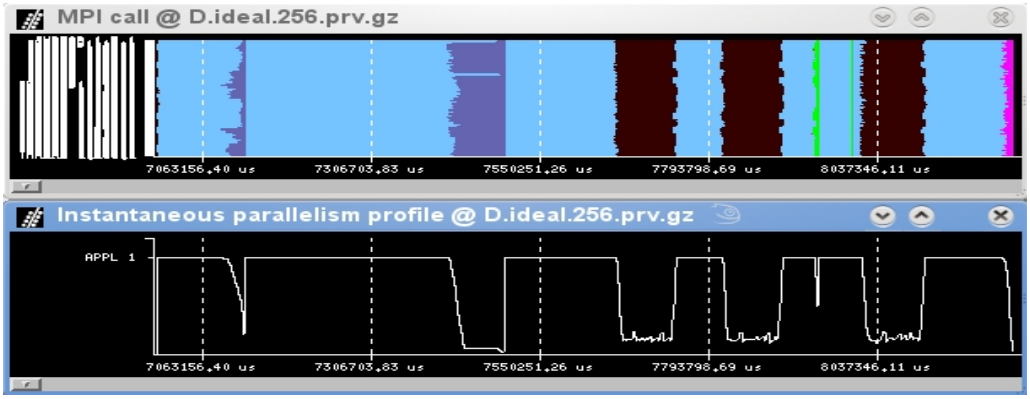

Figure 127: Instantaneous efficiency in the communication phase of the 256 processor trace from ICE, assuming ideal interconnect

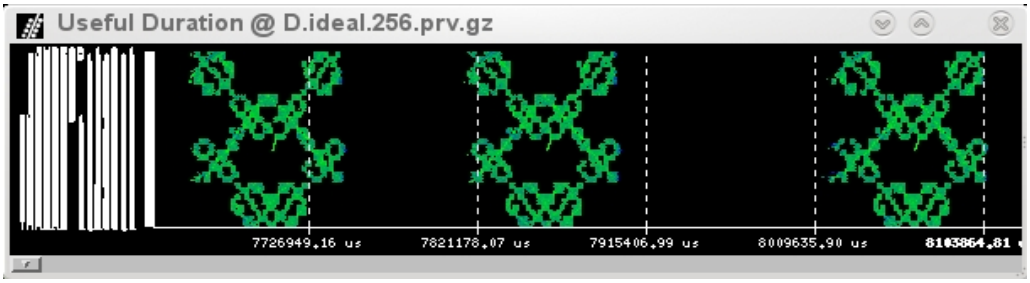

Figure 128: Serialized computations pattern

The code developers/BCOs could easily provide information to identify the actual case and decide whether rescheduling communications could be applied. If so, we envisage that important scalability improvements could be achieved.

3.9 Summary of conclusions

3.9.1 *Node/core performance: Accelerators vs. general purpose CPUs*

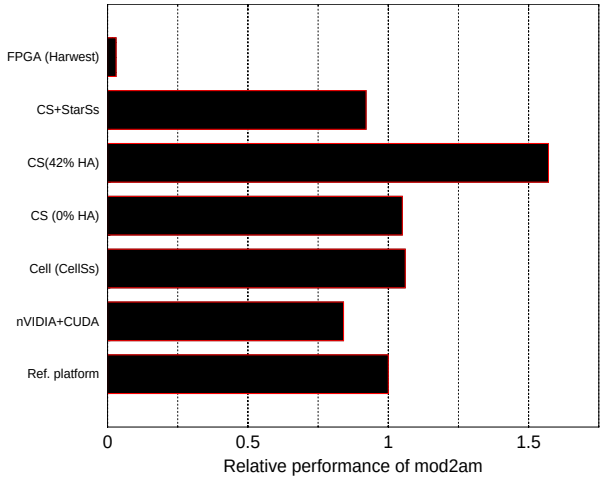

Figure 129: Performance relative to reference platform of mod2am

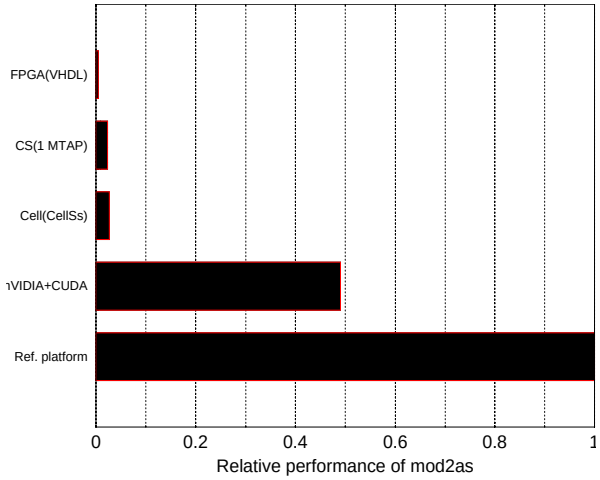

Figure 130: Performance relative to reference platform of mod2as

In Subsection 3.1.1, the reference performance for a selected set of computational kernels is reported in order to compare it on an equal footing with those of the accelerators present in the WP8 prototypes.

We can thus see the benefits of the accelerators with respect to standard CPUs.

Mod2am

The first such kernel is mod2am, a dense matrix-matrix multiplication with a very high computational intensity (defined as the ratio of arithmetic operations vs. memory accesses). The computational intensity is *O(n)* where *n* is the order of the matrix. As all systems considered, general purpose and accelerator alike, are memory-bound this kernel should perform close to the maximum attainable speed for all platforms. Especially for the accelerators with their many cores, this would be an opportunity to excel. The relative performance of the accelerators is given, normalised to the 8-core (1-node) performance of a Nehalem-EP 2-socket node which is 76 GFlop/s.

In the graphs, only the maximal attained performance is shown for the majority of accelerator platforms. The “CS” in the bar graph labels stands for one ClearSpeed CSX700 card and HA stands for Host Assist, i.e., the ClearSpeed card and the host node share the work for the percentage indicated, a unique feature of the ClearSpeed platform. As can be seen from Figure

129, the benefit of using an accelerator instead of an 8-core reference node is quite limited, only the tuned ClearSpeed accelerator with a 42 % host assist percentage is about 50 % faster than the reference platform.

Mod2as

The second kernel is mod2as, a sparse matrix-vector multiplication. It can be seen from Figure 130 that the situation of mod2as kernel is worse. No accelerator is able to attain a decent fraction of the performance of the reference platform (1392 MFlop/s). There are several reasons: First the computational f =2/3, which is quite low and the intrinsic speed of the accelerators in no way can compensate for the transfer of data from the host to the accelerator and back. Second, none of the accelerators provide decent support for reduction operations, which are an essential part of this kernel. So, offloading of this type of computations to the (present-day) accelerators must be avoided at all costs.

Mod2f

The third chosen kernel is mod2f, which does radix-4 complex-to-complex Fast Fourier Transform (FFT).

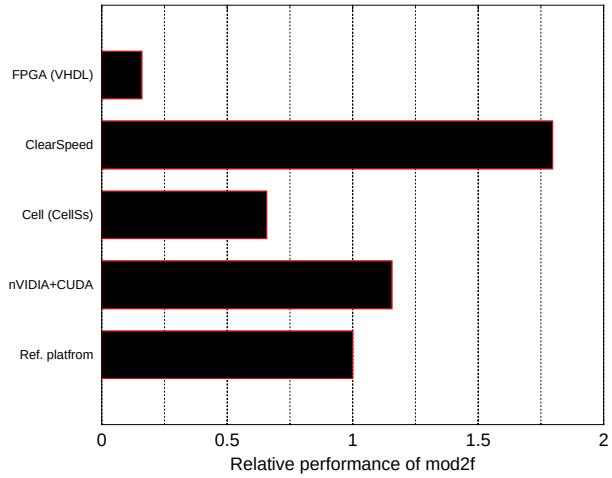

Figure 131: Performance relative to reference platform of mod2f

Comparisons from Figure 131 show that relative to the speed on the reference platform, both for one core and for an 8-core node, modest performance gain of about 3 GFlop/s can be expected from the nVIDIA and ClearSpeed accelerators.

Mod2h

As for the last kernel, mod2h (random number generator), none of the accelerators, except the ClearSpeed card were able to implement it. For the ClearSpeed card, a library routine with a Mersenne Twister algorithm was used, one that was equivalent to the one in the original source code. It was found that the ClearSpeed library version was in some cases about 1.5 times faster than the reference implementation (at 3700 MoP/s). Hence using a random number generator on this accelerator can in some cases be worthwhile.

Summary

In summary, one can conclude that the present accelerators only incidentally will show performance benefits as compared to an 8-core, 2-socket Nehalem-EP node at 2.53 GHz. This does not mean that the accelerators are to be disregarded as potential boosters of performance for several reasons: First, the comparison is made relative to an 8-core node, where possible, against a single accelerator processor (which admittedly in turn may contain up to 240 cores).

However, a Petapath Feynman e780 accelerator unit contains 8 ClearSpeed cards and can be addressed as a single accelerator. In such a situation a speed of about 520 GFlop/s has been demonstrated for kernel mod2am. A clearly larger speed than can be obtained by the reference node. Second, there are many algorithms that cannot be run on all the cores of a reference node transparently. So, in that case the performance per core is a more appropriate measure and in this situation one can expect the accelerators to have an advantage. Lastly, the performances of the accelerators are highly dominated by the host-accelerator bandwidth. Presently none of the accelerators are directly connected to the processor fabric, i.e., HyperTransport for AMD processors and Quickpath for Intel processors. If that would be the case, the bandwidth would be much more favourable and the node memory would be directly accessible by the accelerators which would reduce the transfer overhead dramatically. One may expect this to occur in the near future, say 1 to 2 years. One may add that the functionality of accelerators will be enhanced, for instance by adding support for reduction operations, thus widening the field of applications for which they can be employed. Therefore, accelerators will continue to play a significant role in the compute nodes of future machines.

3.9.2 *Memory bandwidth*

The memory bandwidth within a node and, where appropriate, the host-accelerator transfer speed from a node are one of the decisive factors for the performance of a node or node-accelerator combination. Where the internal node bandwidth can be measured in the 20 GB/s realm and a sub-μs latency, for instance, latencies for host-accelerator data transfers are in the hundreds of μs (see, e.g., the host-accelerator bandwidth experiment on the Petapath system) while the effective bandwidth will be significantly lower than the nominal bandwidth provided by PCIe Gen.2, i.e., 8 GB/s except for very large block data transfers. Although no other such experiments were done there is little reason to assume that the situation for other accelerators will be much better, also due to software overhead. As stated above, this large disparity could be improved by a direct connection to the native interconnect fabric of the processors in the node.

3.9.3 *Network bandwidth*

The reported studies of network bandwidth for point-to-point communication are summarized in Table 25.

Prototype	Env.	Lat. (μs)	BW (MB/s)	Links	Network	Reported
QPACE	MPI	4.7	845	4	Proprietary	3.4.1
Altix XE	MPI	1.7	2500	1	IB QDR	[2]
ICE	MPI	1.9	1800	1	4×DDR IB	3.4.2
UV	MPI	1.6	7200	1	NL5	3.4.2

Table 25: Point-to-point performance

Table 26 compares the performance of MPI collectives measured for 256 MPI tasks on different systems. The Alltoall and Allreduce latencies measured using two different MPI implementations on the same Altix ICE system diverge substantially. Hence very low latency networks and highly optimized MPI versions are of major importance for future Tier-0 systems.

Call	"Latency" (μs)	System	Reported
Alltoall	108	Altix ICE	3.4.2

Call	"Latency" (μs)	System	Reported
		MPT 1.24	
Alltoall	136	Altix ICE Intel MPI 3.2	3.4.2
AllReduce	24.8	Altix ICE MPT 1.24	3.4.2
AllReduce	16.0	Altix ICE Intel MPI 3.2	3.4.2
Allreduce	31.8	Altix4700 MPT 1.24	3.4.2
Allreduce	18.6	Ultra Violet MPT 2.0	3.4.2

Table 26: MPI performance of collective calls

3.9.4 *Hybrid*

The hybrid programming prototype allowed us to learn important facts, should PRACE choose to provide a hybrid system including GPUs at some time.

On this prototype, we focused our efforts on using HMPP, a solution based on source code annotation. HMPP is a very appealing tool since it will offer a solution to our users' community to migrate their old code to GPU usage in a hardware independent manner (though some platform specific optimizations are possible). The evaluation of the tool has been done on a small set of kernels to make our points clearer.

The initial lessons learned from this prototype could be summarized in four points:

- The modification of an existing code to HMPP is lightweight to get a first non optimized running version. Yet, more modifications are needed if the code architecture and programming has not been done taking into account such as vectorisation as well as clean module isolation. Furthermore, some constructions (such as reductions) are very difficult to parallelize and won't achieve decent performance on a graphic card. Those code sections are best left running on the CPU. This is an important point since it will have a big impact on code design. Code programmers will have to decide whether they want to download the data from the GPU to the CPU, thus paying the data transfer in order to compute the reduction efficiently, or leave the data on the GPU, hence accepting lower performances, yet avoiding data movements that can be expensive;
- Producing an optimized version on a hybrid machine of any code requires having an in-depth knowledge of the hardware. This is NOT specific to HMPP and can be noticed in the CUDA porting, too;
- Astute directives for code generation (such as loop reordering, loop fusion, etc.) are a great help to boost performances. We need to see more of them in the future to avoid obscure program construction. HMPP does implement some of them;

- With a bit of effort, performances offered by HMPP programs can be equal or better than those offered by the vendor's library. This is an exciting result, since it shows that most programmers could get to this level of success while investing only a limited amount of efforts to port their code to a GPU. On the other hand, it will encourage the community of GPU experts to explicit some programming patterns that are very likely to appear in everybody's programs.

Yet, more investigations are needed to fully grasp the advantage of using HMPP. We will, in the coming months, compare the usage of the same HMPP coding on different platforms (nVIDIA CUDA and ATI graphic cards).

3.9.5 *I/O*

Robust, scalable and performant I/O subsystems are a key component for Petascale systems, and for the time being it seems there is not a proven solution that can sustain the expected I/O throughput of a Petascale class machine. The XC4-IO prototype mainly investigated innovative aspects of I/O: accessing metadata using SSD technology and performance of the Lustre file system in its native configuration or in combination with pNFS over RDMA technologies.

The experiences achieved during the assessment phase of the prototype were important. After the setup of the prototype, the first benchmark tests had the goal to check the communication between the metadata node and the SSD disks and evaluate their performance in read and write operations. Also the influence of SSD technology on Lustre metadata performance was evaluated. From the experiments done, it appears that this technology although partially immature, is promising and suited to speed up the metadata performance of parallel file systems required by HPC environments.

Lustre performance is strongly influenced by Lustre stripe count and by the type of I/O operations performed, so some work and tuning activity is still required to get better performance in this area.

When performing many small I/O operations, the Meta Data Server (MDS) is the bottleneck. So in this environment, a one LU configuration is the best choice for performance because MDS has to contact only one Object Storage Target (OST) for each file.

The most noticeable drawback of Lustre 1.8.1 is the single active MDS configuration, hence a handicap of the Lustre architecture is the possibility to scale MDS only vertically, increasing its CPU power or memory, but not horizontally, preventing from parallelize its work. This feature will be available only in Lustre 2.0 version which is still in the testing phase (alpha version).

SSD technology can help to reduce the bottleneck of MDS when it is overloaded by multiple I/O requests and SSDs perform slightly better than traditional hard disks. However, the most severe bottleneck concerning Lustre metadata performance is the lack of a distributed metadata service inherent in today's Lustre software architecture. In any other aspect the actual version of Lustre is suitable for HPC: it reaches high I/O performance, saturating the available I/O bandwidth and shows good scalability. However concerning its applicability for future multi-Petascale systems, the metadata performance of Lustre has to be largely improved.

This first experiment of testing pNFS exporting Lustre or other file systems is still immature and it is problematic to integrate pNFS with Linux production kernels. Although pNFS seems promising for the HPC environment, it is still immature and not acceptable for HPC production contexts yet.

3.9.6 *Energy Efficiency*

To what extend the power and cooling efficiency of current supercomputing architectures could be enhanced by using highly energy efficient processing elements as well as through direct liquid cooling of components is impressively demonstrated by the PRACE eQPACE prototype, ranked number 1 on the November 09 Green500 list. On the other hand, the SNIC-KTH prototype illustrates that even with commodity hardware highly energy efficient HPC systems can be realized simply through cherry picking of power efficient hardware components such as processor, memory and power supplies. Here blade-based solutions often provide better energy efficiency values compared to standard servers. Another important result of WP8 concerning power efficiency is the fact that despite the efficiency of the compute servers is increasing with each new system generation, even for present state of the art server technology the power consumption of powered off servers is in the order of several tens of Watt. Hence future power management software should be able to really physically cut off the power supplies. Finally it is important to be aware that the power consumed by many of today's accelerators is significant even when they are idling. Hence mechanisms to reduce the idle power of these devices to a large extend are a major prerequisite to enhance the energy efficiency of accelerated systems.

4 Recommendations for next generation Petascale machines

4.1 Foreseeable architectures

4.1.1 *General architecture*

Figure 132 shows the basic structure and dimensions of current systems in the top 500 list in terms of their peak performance and how they achieve it.

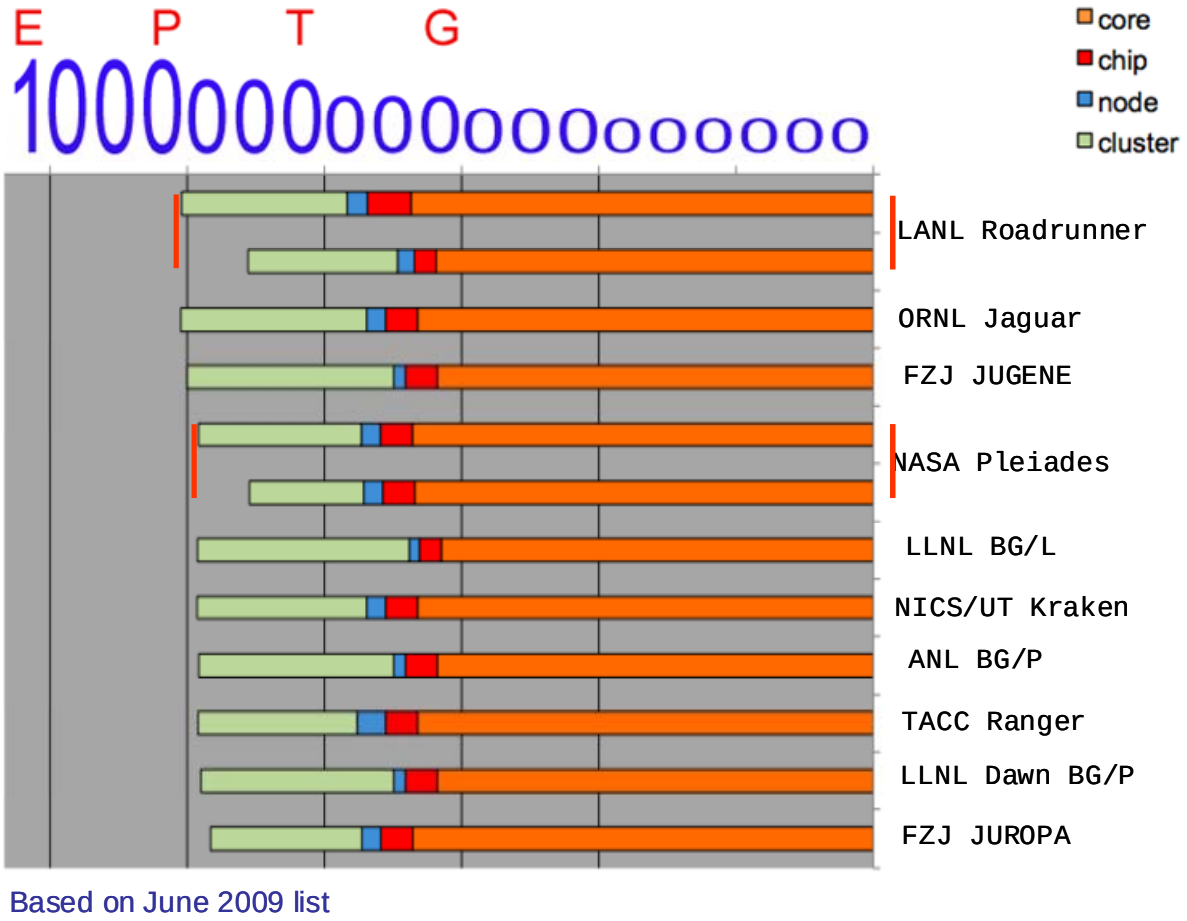

Figure 132: current structure and performance of top machines in Top500 list

The figure is logarithmic in character and shows the scale of each level of constituents with respect to the total system size, and how each type of constituent contributes to the system total performance. The symbols E, P, T, and G indicate the ExaFlop/s, PFlop/s, TFlop/s, and GFlop/s levels, respectively. For the given system or cluster, built out of the following nested *bricks*: cores, chips (or sockets) and nodes, the floating-point computing powers of the different components are represented with bars as follows with respect to the right to left scale of values previously described:

- Orange bar for computing power of 1 core;
- Orange + red bar for computing power of 1 chip;
- Orange + red + blue bar for the computing power of 1 node;
- Orange + red + blue + green bar for the computing power of the whole system.

For instance, in the JUGENE system, a core is capable of 1.8 GFlop/s peak performance and therefore just passing the “G”-line. The four cores on a chip contribute a factor of 4× to the

performance with 2 chips in a node; doubling the performance per node to 14 GFlop/s. The clustering of all the nodes together leads to the final PFlop/s peak performance.

The figure does not show the two approaches followed regarding the operation frequencies. While one tries to minimize the individual per core power, with operation frequencies below 1 GHz, the other uses higher performance cores, with frequencies in 2-3 GHz range. They result in per core performance of 3-4 GFlop/s for the first group and 10-12 GFlop/s for the second. The latter type is presently the major contributor to the total peak performance, as noticed from Figure 132.

The next major contributor is the number of nodes, ranging from 6 to100 thousand. The size of the nodes, which can be decomposed in terms of sockets and cores per socket, is still quite limited, contributing a small fraction to the total parallelism and performance.

Foreseeable architectures for the near future will certainly evolve in the direction of increasing the degree of parallelism at the node level. A possible evolution is depicted in Figure 133. It includes an already announced system targeting 20 PFlop/s, the Sequoia system.

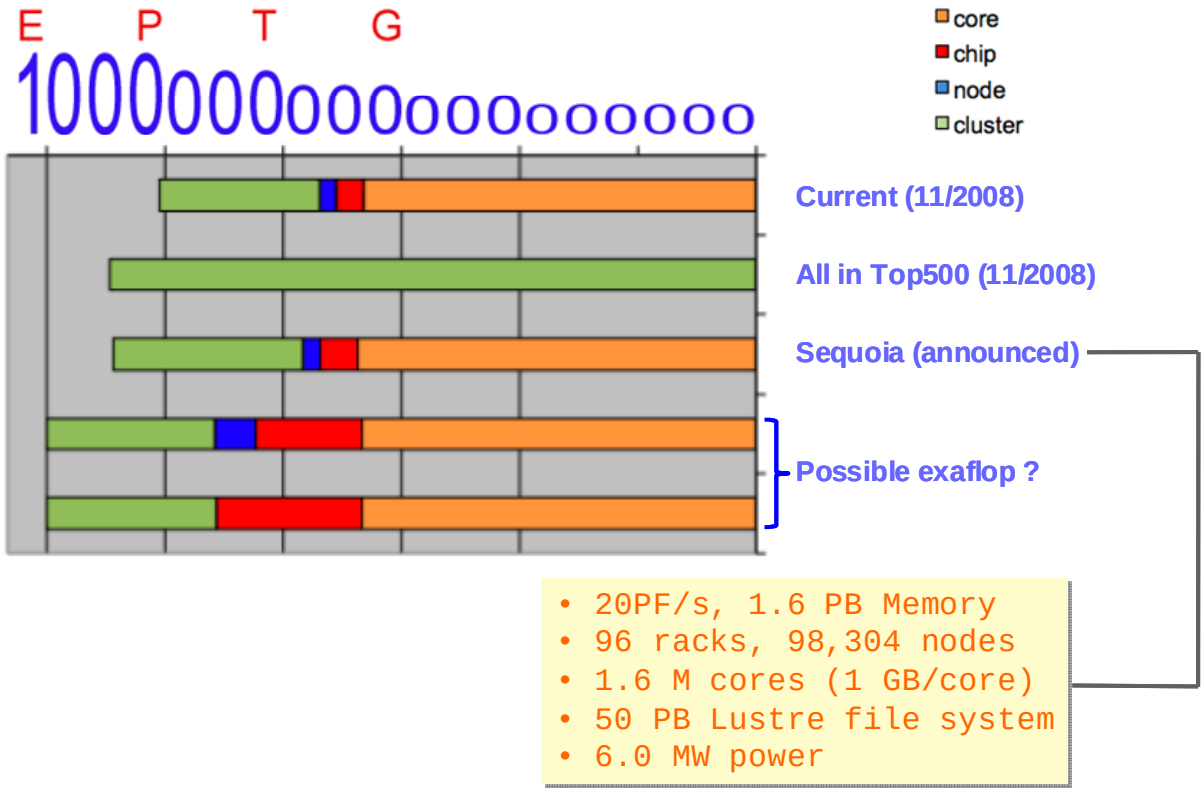

Figure 133: Possible evolution of supercomputer dimensioning

Two alternatives can be followed in terms of the node structure: homogeneous and heterogeneous node, which can further be classified as heterogeneous performance and heterogeneous functionality. Today, we have systems (as those evaluated in WP8 prototypes) that expose the functional heterogeneity, but it is foreseeable that future systems may have cores with identical functional capabilities but different performances. Furthermore, heterogeneity may occur on chip, off chip or both.

The combinations are thus enormous and the best choice will depend at any specific point in time on economic and technological cross points. In general, one can expect that specific off chip devices may be first in achieving a given performance or performance/power ratio, though general purpose multicores may catch up few years later. This has enormous implications on software. The cost of application development, porting, tuning and maintenance will grow exponentially if we intend to efficiently use the available machines. It is of paramount

importance to develop programming models and practices that screen us off from the hardware details and thus help us in this respect.

4.1.2 *Some straw man examples*

In the 10-20 PFlop/s range, up until around 2014, we still will be able to rely on a more or less "classical" design. The already mentioned Sequoia system, the Blue Waters system and the projected 10 PFlop/s system to be built in Japan have one thing in common, that the nodes are still homogeneous (based on the IBM POWER7 and SPARC64 8+ processors, respectively). However, the interconnect networks will be somewhat different. The POWER7 based systems will be on the pruned fat tree type, while the Japanese system will feature a 6D-torus topology to handle the relative fragility that occurs in the classical 3D-torus networks and that would be too unreliable for systems of this size.

As already stated earlier, apart from the approach to use processors in the 3 GHz range and higher to satisfy the performance requirements, the alternative is to turn to very many low-power processors as exemplified at the moment by IBM's Blue Gene series of machines. Such systems may or may not consist of many-core implementations. Presently, already experiments are underway with systems consisting of embedded processors (Wehner, Olliker, Shalf, Int. Journ. HPC Appl., May 2008) in which, a system built from 2×10^7 8-core Tensilica Xtensa processors will be used. This so-called "Green Flash" system envisioned would require 4 MW of power at a projected peak performance of 200 PFlop/s. It should be noted that a performance of 10 GFlop/s per processor can still be expected but only at 200 mW power usage per processor, while the processor frequency would be in the MHz range. It should also be noted that classical silicon technology can still be employed. We will comment on the consequences with respect to reliability and manageability of such massively parallel systems later on.

Further out in the future, say from 2015 we cannot rely anymore on technology as we know it today. It is not possible to just multiply the number of cores/nodes by, say, 10 to get at the desired performance level. For one, the power requirements would be unacceptable as would be the management problems for the given amount of processing elements. This means that the speed per core will have to increase, either by incorporating accelerating processing elements or by new device technology and probably both. Furthermore, also the memory technology must change in speed, bandwidth, and power consumption. Technologies, like magnetic RAM, memristor or graphene-based RAM may be of sufficient maturity by then (2015—2016) to constitute the system memory. As it is foreseeable that around 2015 the usual lithography techniques to create processor chips will fail, further improvements in this area will heavily depend on progress in nanotechnology in order to yield devices that can deliver the required performance per computational core. Assuming that 1 TFlop/s per processor can be realised and with 16 processors per node we need 6400 nodes to arrive at a 100 PFlop/s system. This assumption is based on, for instance, 64 cores/processor at a peak performance of 16 GFlop/s per core or a combination of less cores per processor but with the assistance of computational accelerators, accompanying either the cores or on the processor level. Such a system seems a viable target, in particular when one assumes that per socket an accelerator device is deployed.

It is expected (hoped) that by 2019—2020 we will arrive in the ExaFlop/s range. This will need further technological breakthroughs on the processor, memory and interconnect fronts. For processors the employment of SQUIDs (Superconducting Quantum Interference Devices) as the basic logical switching devices could be a possibility. Switching times are close to the THz regime and the power uptake is extremely low, enabling systems that still would have acceptable energy consumption. A serious drawback, at least for the present, is that these devices only work at the temperature of liquid helium. Apart from SQUIDs, prospects for gra-

phene- and carbon nanotube-based technology is well imaginable. The time horizon is too far of, however, to predict what the hardware implementation actually will be. Very roughly, when would assume 10 TFlop/s sockets by the technology in that time frame the macro-structure of an Exaflop/s system could be similar to that of a 100 PFlop/s system: 16 sockets/node and about 6400 nodes would get us there. This is by no means certain of course as we have to keep in mind that also the memory and interconnect must be able to keep pace with the faster processors. This will require additional innovations in these fields.

Below we will briefly discuss some of the relevant issues in somewhat more detail for the nearer future.

4.2 Relevant issues

In this section we discuss issues that will have to be addressed for future supercomputers. Even if we have mentioned different potential architectures, these issues are broadly applicable and in most cases will affect several or all of them. The answers to these issues are certainly not yet known but may have implications resulting in modifications of the basic architectures described above. For each of the issues we will describe some directions that might deserve important research efforts and the possible architectural implications.

4.2.1 *Power and Energy efficiency*

With the present silicon-based hardware, developers would certainly need to improve the ability to control the activity and level of performance of the components in a chip. By 2015 it would be impossible to power up all the logic in the chip at the same time as its power budget will be far above what will be possible to cool.

Although some automatic hardware mechanisms will be put in place, it is foreseeable that a fair amount of knobs will be provided for the software to drive the power saving capabilities. The most desirable situation is one where the run time will automatically control such knobs, although power control APIs may be a good initial approach to let the application explicitly perform such function. Research is needed in these directions. Now already some processors are aware of the activity of their functional units and are, in principle, able to react on that activity level. In the Nehalem-EP, the clock cycle can be stepwise increased to run certain tasks faster with the constraint that it stays within a pre-assigned power envelope. What will be needed is the reverse: lowering the clock cycle when a certain resource is predicted to be used less intensively.

Already today the power usage of the memory part of a system can be as much as 30 % due to the fact that for the moment all memory technology used is volatile. This means that whether the memory is actually used or not, it consumes energy. Hence a large reduction in power consumption can be expected with the introduction of new non-volatile memory types such as Spin Torque Transfer Magnetic RAM (STT MRAM), memristors, or graphene-based memory, large power usage reductions can be expected in future system architecture. Of these SST MRAM is the technology that is the closest to the market. It is believed by the two main companies associated with this technology, HYNIX and Samsung, that it can be implemented with a feature size < 65 nm, fairly fitting with other present-day technology. Products are expected to hit the market by 2011. Both other technologies are further out, but with great potential with respect to switching power and speed. They are repeatedly demonstrated but the fabrication methods differ from those for silicon-based technology current today and various nanotechnology hurdles have still to be overcome.

With respect to the I/O-part of the systems a similar development takes place. Already now, Solid State Disks (SSDs) based on non-volatile flash NAND- or NOR-technology are offered

by various vendors for data caching and metadata serving (see the XC4-IO prototype earlier in this report). As yet this technology is too expensive to use for the full I/O subsystem but this may change rapidly, especially when power costs are taken into account. Furthermore, the first Phase-Change RAM (FCRAM) will be introduced to the market in 2010. This technology is faster, more power efficient and more reliable than the present-day flash-based products though it is too slow to employ it for main memory. It is expected that it will replace present SSD products in the next few years, first for meta-data serving and soon thereafter for the full I/O subsystem. This will have a major impact on the total power budget for large-scale systems from about 2014 on.

It is often not realised that also the interconnect networks and the associated switches and Host Bus Adapters (HBA) consume a significant fraction of the power in a system. This situation may change in about five years when photonic networks will be introduced. Already complete Network-on-a-Chip (NoC) implementations in silicon have been demonstrated (Petrarca, Lee, Bergman, Carloni, IEEE Micro, July/August 2009) with the potential of seamless photonic connection between the chip and system level without any intermediate electronic conversion. The use of a purely photonic network would reduce the power consumption of the interconnect network by several orders of magnitude.

A further line of research is to develop power control capabilities at the job scheduler level. Interest has been shown in some of the current job scheduler providers but we believe that it is at a very rudimentary state and in particular much tighter coordination between the low level run time scheduling and the job scheduler would be needed to address the problem holistically.

4.2.2 *Programming models and compilers*

The theoretical peak performance of multi-Petascale systems is of little benefit unless applications can make use of it to a large extent. This fact has been recently recognised in the USA and Europe and is reflected in the activities of STRATOS, IESP and EESI. However it is a matter of fact the vast majority of application codes is written under the assumption of homogeneous cores. Re-writing these codes from scratch is not feasible since the development time for many large-scale applications is in the range of decades/ten years and more. So in order to keep application codes maintainable in the presumably hybrid multi-Petascale era, programming models are needed which are able to handle the heterogeneity of compute cores while keeping the impact on the programmer side as low as possible. To prepare legacy codes for hybrid computing needs easy tools which keep the portability of the code. Examples for such tools are the PGI compiler or CAPS HMPP. New codes could be written using new languages that allow transparent use of different accelerators (e.g., OpenCL, StarSs, RapidMind). However, the survey on new languages carried on in cooperation of WP6 and WP8 found that most of these languages and tools are still too immature to be used in an HPC environment.

For FPGA-based products the situation is even worse today. While an OpenFPGA initiative was created back in 2005 which today consist of 200 members spanning 40 countries worldwide, no usable OpenFPGA API is available to the HPC community so far.

Also PGAS languages such as CAF, UPC, X10 and Chapel do still exhibit many signs of immaturity. Except for UPC, these languages are currently no real alternatives for program development. Nevertheless, the developments in this field are very rapid and PRACE activities may be instrumental to enhance the maturity of these novel languages.

4.2.3 *Accelerators*

The evaluations show that for numerical calculations based on DP arithmetic present accelerators only casually show performance benefits as compared to an 8-core, dual-socket Nehalem-

EP node. Applications using only SP floating-point arithmetic can experience a substantial speedup. But also for this case any performance boost is very much dependent on the actual match of the characteristics of the algorithm and the device. The programmability is still not ideal, especially when aiming at very high performance and non-trivial codes. This does not mean that the accelerators are to be disregarded by PRACE as potential performance boosters for several reasons:

- New accelerator versions from Intel, nVIDIA and AMD will provide a much higher double precision performance- compliant with IEEE-754 2008 standard and to some extend also provide ECC memory (e.g., Intel Knights Corner and nVIDIA Fermi) as well as enhanced functionalities for scientific usage;
- Many of the techniques to be developed for accelerated systems will be useful in optimizing the performance of future many-core processors;
- We can expect that processor vendors and/or board manufacturers will eliminate one of the major bottlenecks in the use of accelerators today - the bandwidth limits between host processor and memory, and accelerator - leading to future systems that will have tightly coupled heterogeneous cores;
- Due to the high LINPACK performance of future accelerator based approaches, it is very likely that the processing cores used in future supercomputer systems will be heterogeneous e.g., high performance many-core processors and accelerators or heterogeneous many-core processors will constitute the processing elements in these systems. At the time of writing an accelerated system is holding position two of the June 2010 Top 500 list. Also HPC vendors such as Bull, CRAY, HP, IBM and SGI already provide accelerators support in their latest system architectures.

However, finding the right balance in node architecture and programming models requires significant further research and development efforts. The programming models should be able to handle the heterogeneity that these two types of resources constitute and intelligently decide where to run the different tasks based on their bandwidth and computation needs.

Finally it is important to be aware that the power consumed by today accelerators is significant even when they are idle. Therefore mechanisms to reduce the idle power of these devices to a large extend are of major importance for future accelerated systems.

4.2.4 *Network Interconnects*

Infiniband will presumably remain the most dominant HPC interconnect technology in the upcoming years. In order to keep the cost of the network interconnect within 20 % of the total hardware investment budget the topology of the interconnect topology of future systems will most likely be realized as 3D network torus or n-dimensional hypercube, as dragonfly network, as pruned fat tree or some other type of hierarchical network interconnect with position dependent communication latencies of the communication partners and even position dependent per processor core network bandwidth.

Today the impact of the network interconnects bandwidth and latency on application performance are often not thoroughly understood by many users and code developers. Hence sometimes the interconnect is blamed for problems that are in fact on the application side (for example load imbalance) or caused by network contention due to a highly suboptimal placement of MPI tasks on the network topology. There are nevertheless two general observations to make. The first observation is a need for ever growing network bandwidth per core, or at least per node. Secondly, the available network bandwidth per core will most probably decrease in future. In order to enable applications to efficiently use a large fraction of future Petascale systems the HPC community has to put large efforts in the development of numerical algorithms avoiding “bursty” or “wrong order” communication patterns and thus allowing

an orders of magnitude increased scalability without requiring an increase in available per core network bandwidth. Also the application of topology-aware batch scheduling and intelligent network routing mechanisms will be of key importance in future systems.

4.2.5 *Memory bandwidth and latency*

Presently, memory latency is in the range of hundreds to a thousand clock cycles and although memory bandwidth has been increased by the transition from DDR2 to DDR3 memory, latency actually has become worse. Until radically new memory types, like memristor or graphene-based memory will become available, hopefully in 2015/2016, this situation will not improve. Memory bandwidth however, will increase as soon as 3-D memory packaging becomes available. Intel, TSMC, and Samsung are working on this concept. This would at least solve the shrinking perimeter problem for processor chips that cannot have enough memory ports on their perimeter to accommodate sufficient bandwidth with ongoing technology shrinks. For large regular data block transfers where also operand pre-fetching can be applied, 3-D stacking is a solution and for the newer expected memory types even mandatory, but the latency problem still remains. While on another scale, it is similar to that the problems encountered in distributed-memory parallel applications or in executing applications on a computational grid. The upshot is that applications should be implemented in a hierarchical way where parts that are the most latency sensitive are performed close together, e.g., on cores within one processor, while parts that are less latency sensitive can be executed on one or a few neighbouring nodes, etc. This will require re-writing of many applications, partly or entirely.

Chip architectures and programming models will have to tightly evolve together in order to let applications tolerate the huge latencies to come. Determining the right amount of on chip memory structure (cache, local store, capacity and architectural support block moving engines, prefetchers, etc.) as well as the appropriate ways to optimally handle it by intelligent runtimes is a key area of research for the near future.

4.2.6 *Memory per node and core*

Memory constitutes an important component in the procurement as well as for the operation (power consumption) cost of current and future supercomputers. However, in many cases this resource cannot be used very efficiently. For instance, BSC reports that less than 40 % of the system memory is used on average. Still, some applications do require a lot of memory but many others do not. Current practice is that the user specifies his memory requirements in a batch oriented way which fixes the amount of memory claimed for the whole runtime of the application. Also the time to solution is often the decisive factor in choosing the amount of nodes while the memory that automatically comes with it is a mere by-product of that decision.

A possible solution to decouple the computational requirements of the application from its memory requirements could be a complete virtualization of the memory on the total system. Companies like ScaleMP and 3Leaf already have products to realize memory virtualisation on the system level, albeit their products are mainly targeted to create large virtual shared memory nodes. Admittedly there is an overhead associated with these solutions but, ScaleMP has already demonstrated that using an assisting ASIC this overhead can be made quite small. Furthermore such approaches could also be used to accommodate applications with extremely high memory requirements on standard dual socket server hardware or to provide a very cost-effective solution for shared-memory parallel programming models and PGAS languages.

In summary, unless at least a certain level of memory virtualisation takes place, many future HPC systems will need to be equipped with more memory per node than a large fraction of the application codes require. But so far, these virtualisation techniques are not heavily used

by the HPC community due to limitations concerning performance and supported hardware extensions. A PRACE activity may be instrumental to better address the needs of the European HPC community in this field.

4.2.7 *Performance tools*

Performance predictions and performance modelling tools will be mandatory on the software side to effectively use the multi-PetaFlop/s systems we are targeting. Subection 2.2.4 and Section 3.8 already show the benefit of such tools. When both systems and applications increase in size a rational basis for new development or re-implementation (see paragraph above) can be provided by such tools. Also they can point to inconsistencies in the perceived system performance.

Performance prediction and modelling could be greatly eased when applications were decomposed into their main (computational) kernels in a machine independent way. When at the same time the performance of these kernels would be known for a wide range of actual platforms one could construct the performance of the application by factoring in the contribution of each kernel. A model how such a kernel-based analysis tool might work is shown in Figure 134.

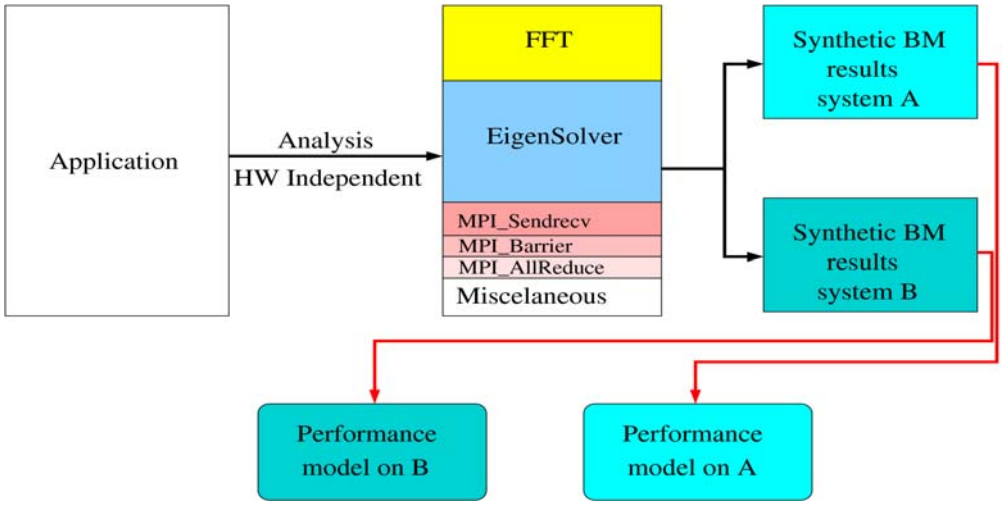

Figure 134: Kernel-based performance modeling

It would also allow us to pose "what if" questions regarding the performance for systems not (yet) available. Little work has been done in this field [23], but no tools exist at the moment to do such a kernel analysis of applications. Obviously, a set of kernels, for computation, communication and I/O must be identified as a basis for this analysis tool.

4.2.8 *Load balance*

Load balancing is often a major bottleneck when doing strong scaling of applications as the point arises where the number of processors approaches the underlying granularity and heterogeneity of the physical model being simulated by the application.

As described in Subsection 2.2.4 two types of load imbalance show up in applications. Global load imbalance happens when some processes have consistently more work than others. This is probably the type of imbalance that mostly concerns current developers and for which approaches based on libraries to recompute partitions are used and will still be relevant in the future. Microscopic load imbalance, where different processors are more loaded/delayed that the others in successive synchronized phases of computation are the ones that will be more important in the future. Introducing asynchrony in the programming model is one way to ad-

dress this issue. The reason why current hybrid approaches such as MPI+OpenMP have failed to properly address this issue is that they do lack this asynchrony.

A frequent situation is the lack of awareness of the actual level of imbalance in an application. The impact of input data set, processor count, variability in cache performance for different domains, network performance, operating system noise etc. can have significant and different effects on each individual run of one application. Run time techniques are thus needed to dynamically adapt to the instantaneous imbalances and being applicable at relatively fine levels of granularity (sub millisecond, as opposed to the typical repartitioning schemes that are applied at granularities coarser than seconds).

4.2.9 *Runtime Systems*

The runtime system is the part of the software infrastructure where actual and more accurate information is available about system resource availability and performance; thus this component has the potential to make better-informed decisions on behalf of the application.

With ever more complex architectures and variability in the behaviour of systems, static approaches where the programmer or compiler statically schedule operations will result in poor efficiencies. The static approaches are quite common practices in current parallelisation approaches and run times. It will be necessary in the future to increase the amount of intelligence introduced in the runtime with the objective of maximising performance and minimising power consumption.

An important issue to address will be the identification and exploitation of parallelism. In this area, intelligence has to be devoted to optimise the use of heterogeneous platforms both in performance and functionality. For example, experiments in Subsection 3.1.5 show the importance of combining the effort of both host and accelerator cores. The approach used is relatively static but sufficient to identify the need and potential benefits that an automatic and dynamic approach would have.

Example techniques that are foreseen to be of relevance in the future are run time dependence analysis or just in time compilation. More mechanisms to expose and exploit asynchrony in the execution of computations as well as their overlap with communication have to be developed.

Another area where runtime will play a key role is the handling of the memory hierarchies in a system. We envisage that techniques (such as renaming, caching, coherence and consistency management, etc.) that have been used in the past at the architectural level to address both fine grain parallelism and the memory subsystem will in the future be migrated to the run time level. By applying those general ideas and techniques at a coarser granularity level than what current hardware does, very innovative perspectives will rise.

In general, we understand that intelligent run times are the direction to follow in order to address many of the architectural and application issues identified in the following sections.

4.2.10 *Resilience*

Resilience is expected to become a huge problem at the multi-Peta and Exascale level. Presently most vendors place the burden on the user side, expecting them to write user level checkpoints within their applications. However, these user checkpointing mechanisms will have to be largely improved in many cases in order to be able to scale to Millions of cores and efficiently handle such large application footprints. Regardless, this repair approach may be only sufficient until the 2015-2016 time frames.

After 2016, an approach based on fault avoidance schemes will be required since at that time the mean time between failures (MTBF) will be in the same range as the time which is re-

quired to write an application checkpoint to a non-volatile storage medium. Hence the applications, the run time environment and the corresponding hardware will have to be able to go on without disruption even if individual components do fail. For this, it is needed that all processors at least have a chip-kill facility that enables them to retire gracefully on failure.

In general end-to-end data integrity mechanisms are urgently needed for all data transfer subsystems of future Petascale systems to avoid data loss or data corruption. We can expect that this feature will be generally available until the 2012 time frame.

4.2.11 *Arithmetic*

All standard processors adhere to the IEEE 754-2008 standard for floating-point arithmetic. However, this is not (yet) the case for some present-day accelerators. For large-scale computations this arithmetic model is absolutely necessary in order to yield reliable results of sufficient precision. It may even be necessary to augment the present standard to include 16-byte floating-point data types to maintain an acceptable accuracy in results that in lower precision would deteriorate because of round-off errors.

4.2.12 *Benchmarks*

As already argued above, a consistent set of computational, communication, and I/O-kernels is needed to be able to model and predict the performance behaviour of applications with respect to core, node, and system performance and scalability. However, this is a mere basis. Also a set of sample applications is required that is representative for their respective application areas. In PRACE already such a benchmark set has been selected but it will be necessary to continuously scrutinise this set (as well as the kernels) to guarantee that they continue to reflect the computational practice at each point in time and that they are able to employ the target systems in an effective way. With regard to the kernels, these must be updated to reflect new algorithmic approaches that better fit the future systems than the classical ones do.

4.2.13 *Libraries*

With the advent of the very large systems ahead the (numerical) libraries as they exist today will not suffice. Firstly, language bindings for present-day libraries are only available for C, sometimes C++, and for Fortran. Accelerators may or may not have their own libraries that are a small subset of what is offered for standard processors. Furthermore, all libraries are built to satisfy one optimisation criterion: speed. For future systems energy efficiency might be another, equally important criterion. In this case one would have to choose between alternatives that guarantee the choice of an algorithm with a minimum power envelope even taking into account that for different algorithms, different hardware components of a system might be involved. In this situation it can hardly be asked from the average user to take all these considerations into account. The user must therefore ideally be oblivious of the details underlying his/her library request. The consequence is that an enormous effort is needed to restructure and augment the libraries as we know them today. In addition, the platform-specific libraries as now offered for accelerators should be completely subsumed into the new standard library interfaces.

4.2.14 *Applications*

Presently, some fields that are in principle amenable to parallel processing are very much under-represented. A majority of these fields are characterised by unstructured parallelism like those associated with large graph analysis problems. Neither the standard processors of today, nor the existing accelerators are fit for processing these kinds of problems. Yet, they represent very important problem classes, like routing problems, automated machine learning algorithms, and pattern recognition in large unstructured data volumes. For such problems mas-

sively multi-threaded systems with a (virtual) shared memory are required, somewhat like the now discontinued Cray XMT system. In this respect the massive core alternative to multi-PetaFlop systems may play a positive role.

5 Conclusions and Final remarks

With the observations made in the previous chapters we are in the position to make some recommendations regarding the course to be taken in acquiring and evaluating the machines to come.

- We believe that both courses taken for the development of multi-PFlop/s systems are viable:
  - 10^4—10^5 nodes built from of relatively high-frequency processors or
  - Massive node systems ($\geq 10^6$ nodes) with very low power requirements are valid, at least until 2015.

After that year technology changes will dictate the direction(s) to go.

- For reasons stated above we cannot disregard the inclusion of accelerators into systems that we deem of interest for the HPC community. Furthermore, we have to continuously assess the technology developments in general.
- Tools for performance analysis and prediction are of critical importance to enable the efficient use of the target systems. Development of such tools should be actively pursued. In this regard also the development of a rational and complete set of kernel benchmarks and example applications is of high importance.
- Even if we are able to build Exascale hardware in the future, unless we address the software issue, those systems will not be usable. Programming model is the key component to cut the Gordian Knot of Exascale architectures. At the node level, models to ease the use of accelerators, and improve the exploitation of parallelism and locality are of huge relevance. Important contributions are being made in this area by European developers. At the cluster level the evolution of MPI and the new PGAS languages have to be monitored. The hierarchical integration of cluster and node level models (hybrid models) is another important direction. It is crucial to further develop and evaluate these alternatives and contribute to their maturation.
- Libraries as presently available are not up to the demands that the future systems will pose. Therefore active participation in developing libraries that accommodate the new programming models as well as a larger variety of optimisation criteria, like energy efficiency.
- With respect to system resiliency both memory and processor chips need chip kill capability and provisions for dynamic re-assignment of tasks/threads while interconnect networks must have full dynamic re-routing capabilities.

6 Annex

6.1 Benchmarks

This section describes the most commonly used benchmarks across the different WP8 prototypes (both hardware and software). The focus of this section is on general overview of the benchmarks.

6.1.1 *EuroBen – Synthetic Benchmarking Suite*

The EuroBen [13] benchmark provides benchmark programs for scientific and technical computing to assess the performance of computers for these fields. All programs are originally written in Fortran 90/95.

Since many SDKs that are available for the PRACE prototypes can only deal with C, the PRACE members agreed on starting all code-porting from an initial C port.

Four of the EuroBen benchmarks were chosen for PRACE:

Mod2am

A dense matrix/matrix multiplication C = A×B;

Mod2as

A sparse CSR (compressed sparse row) matrix/vector multiplication c = A×b. The values of the matrix A are generated randomly. However, with the same seed the same matrix can be reproduced;

Mod2f

A 1-D radix-4 Fast Fourier Transform;

Mod2h

A random number generator;

The requirement was to adapt the C baseline-code to the prototypes as good as possible (e.g., use libraries) to get the best result.

6.1.2 *High Performance LINPACK*

The standard parallel LINPACK benchmark is used to obtain an estimate of the peak performance of the system; this benchmark also gives an impression of the quality of the vendor's BLAS implementation. For a given problem size N, αN^2 bytes of memory storage are required; ideally, α should not be much larger than 8 if eight-byte floating-point words are used. If this storage is distributed across P tasks, the amount of memory per task required will be

$$M = \frac{\alpha N^2}{P}.$$

For execution on a large parallel system, a compromise may need to be made between the long runtime needed versus the high fraction of peak performance achieved for large problems. The performance L(P, N) is determined by

$$L(P, N) = \frac{\frac{2}{3} N^3 + 2 N^2}{T_{measured}}$$

where, $T_{measured}$ is the execution time for problem size N and task count P. Note that the implementation is required to preserve the operation count specified above, i.e. use of a Winograd Strassen or related algorithm for performing matrix multiplications is prohibited. Otherwise, vendor-specific implementations with respect to coding, communication mechanism and (possibly multi-threaded) BLAS library implementation are encouraged. The reference implementation HPL [21] has recently been updated; the new 2.0 release contains improvements of the scalability of the initialization phase to systems with many hundred thousand cores of memory, bug fixes for systems with large memory and improvements in the verification algorithm. The reference implementation uses MPI and standard C. It contains a BLAS implementation, but a vendor specific C BLAS library may replace the integrated BLAS calls.

6.1.3 *Intel MPI Benchmark (IMB)*

This set of benchmarks allows to evaluate interconnect properties with respect to sustained bandwidths and latencies for point-to-point operations as well as its global properties with respect to collective operations.

The IMB [24] package consists of 3 parts:

- IMB-MPI1;
- two MPI-2 functionality parts;
- IMB-EXT (one-sided communications benchmarks);
- IMB-IO (I/O benchmarks).

For each part, a separate executable can be built. If the MPI-2 extensions are not available, it is possible to install and use just IMB-MPI1. Only standard MPI-1 functions are used, no dummy library is needed. The IMB code itself is written in standard C, so should be portable to any multi-purpose platform by using the suitable modifying compiler calls and options in one of the include files provided for Make.

6.1.4 *Triads (RINF1) Benchmark*

This benchmark tests the floating-point and integer performance of various one-dimensional loop kernels; depending on the access pattern various aspects of the processor architecture and the memory hierarchy are tested. This benchmark is derived from the RINF1 Genesis benchmark originally developed in the HPC Department at the University of Southampton, England. It can be regarded as a generalization of the well-known STREAM benchmark.

The basic loop kernels are typically of the format

```
do i=1, n, is
    a(i) = b(i) op c(i) [+ d(i)]
```

The arrays **a, b, c** and **d** may vary in type and kind, and "***op***" represents a binary arithmetic operation (normally multiplication). This kernel structure would be characterized as requiring 2 or 3 loads, 1 store and two integer or floating-point operations. The kernels are executed for various values of the vector length n, yielding the performance dependence on temporal locality, and for varying strides, which allows evaluating the performance degradation resulting from loss of spatial locality. To ensure that timer resolution does not impact the reliability of measurements, a number of iterations of the kernel is performed, such that a

given measurement interval is approximately achieved. The loop structures optionally can be compiled with OpenMP directives, allowing an assessment of multi-threaded performance. Furthermore, an option is provided to run the benchmark on multiple compute nodes concurrently via MPI; in this case an identical setup is replicated across all MPI tasks. To ensure that a given system delivers the performance homogeneously, MPI barriers are executed before and after each loop kernel, prior to taking the timestamps for start and end of the kernel; performance variation across MPI tasks is recorded as a statistic. To enable a quantitative comparison of different systems, various performance averages are calculated from the profiles.

6.1.5 *Random Access Benchmark*

In scientific simulations, there are an increasing proportion of cycles with non-contiguous memory access patterns. Compared to contiguous access the achievable performance is then limited by memory latency rather than by bandwidth, especially if only one or few elements per loaded cache line are used. The Random Access [25] or GUPS benchmark, which is also part of the HPCC benchmark suite, measures the memory Giga-updates per second achieved for randomly distributed memory access patterns. It can be run in serial as well as parallel (MPI) mode; the aggregate – depending on the properties of the system interconnect – may not scale linearly with the number of MPI tasks or nodes used.

6.1.6 *APEX Benchmark*

The starting point of the Application Performance Characterization project [26] (Apex) is the assumption that each application or algorithm can be characterized by several major performance factors that are specific to the application and independent of the computer architecture. A synthetic benchmark then combines these factors together to simulate the application's behaviour. Thus, the performance of the benchmark should be closely related to that of the corresponding application. Such a benchmark can be used as a realistic indicator of achievable application performance and enables the users to directly evaluate a new platform based on their own interests. At the same time, synthetic benchmarks are substantially easier to use than real applications. Real applications are often too complex to be run by simulators, thus prohibiting their use in the development of new architectures.

6.1.7 *STREAM Benchmark*

The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. It measures "real world" bandwidth sustainable from ordinary user programs, not the theoretical "peak bandwidth" provided by most vendors. The STREAM [27] benchmark is applicable to a wide range of computers, from PC's and Mac's to vector supercomputers and massively parallel processors. The STREAM benchmark is specifically designed to work with datasets much larger than the available cache on any given system, so that the results are (presumably) more indicative of the performance of very large, vector style applications. STREAM benchmark can be run on both uniprocessor and multiprocessors. The STREAM benchmark has four kernels as described in the table below:

Name	Kernel
COPY	a(i) = b(i)
SCALE	a(i) = q*b(i)
SUM	a(i) = b(i) + c(i)
TRIAD	a(i) = b(i) + q*c(i)

Table 27: STREAM Benchmark Kernels

6.1.8 *IOR Benchmark*

IOR benchmark tool, a tool developed by LLNL, which tests system performance by focusing on parallel/sequential read/write operations that are typical in scientific applications. It uses IOR (Interleaved or Random), a script used for testing performance of parallel file systems using various interfaces and access patterns. IOR uses MPI, as parallel programming model, for processes synchronization, all nodes involved in IOR commands must use ntpd (Network Time Protocol daemon) , as IOR timing depends on it. With IOR, clients can first write data, and then read data written by another client, avoiding the problem of having to clear a client's cache.

IOR tool exposes plenty of optional parameters that allow performing different kind of test, modifying the most significant parameters of the application. This shows the advantages and drawbacks of using a particular hardware and software architecture stressed by an application that works in a specific way.

By modifying the invocation parameters in the command, it is possible to simulate a client that starts an application with particular requirements and uses specific data structures. It is possible to experiment one file per processor or shared and parallel file access patterns for common set of testing parameters.

The most useful and interesting tests could be made by varying the following parameters:

- a S api -- API for I/O [POSIX|MPIIO|HDF5|NCMPI];
- b N block Size -- contiguous bytes to write per task;
- c collective -- collective I/O;
- F filePerProc – file-per-process;
- N N numTasks -- number of tasks that should participate in the test;
- s N segmentCount -- number of segments;
- t N transferSize -- size of transfer in bytes.

6.1.9 *CPU Burn-in*

CPU Burn-in [28] is a stability testing tool for overclockers. The program heats up any x86 CPU to the maximum possible operating temperature that is achievable by using ordinary software. This allows the user to adjust the CPU speed up to the practical maximum while still being sure that stability is achieved even under the most stressful conditions. The program continuously monitors for erroneous calculations and errors, ensuring the CPU does not generate errors during calculations performed under overclocking conditions.

CPU Burn-in constantly cycles FPU intensive functions for a user specified period. The resultant calculations are constantly checked for data integrity. If the program detects erroneous data the user is immediately informed. Applications such as SETI@home and Distributed.Net perform no such data checking. The user must rely on those programs to crash or the system to hang before a problem can be noticed.

6.1.10 *CacheBench*

CacheBench [29] is a benchmark designed to evaluate the performance of the memory hierarchy of computer systems. Its specific focus is to parameterize the performance (raw bandwidth in MB/s) of possibly multiple levels of cache present on and off the processor.

The goal of this benchmark is to establish peak computation rate given optimal cache reuse to verify the effectiveness of high levels of compiler optimization on tuned and untuned codes. CacheBench incorporates nine different tests, each performing repeated access to data items

of varying vector lengths for a number of iterations. Computing the product of iterations and vector length gives total bytes of data accessed. These nine tests are:

- Cache Read;
- Cache Write;
- Cache Read/Modify/Write;
- Hand tuned Cache Read;
- Hand tuned Cache Write;
- Hand tuned Cache Read/Modify/Write;
- Memset() from the C library;
- Memcopy() from the C library.

6.1.11 *IOzone*

IOzone [30] is a file system benchmark tool. The benchmark generates and measures a variety of file operations. IOzone has been ported to many machines and runs under many operating systems.

IOzone is useful for performing a broad file system analysis of a vendor's computer platform. The benchmark tests file I/O performance for the following operations: Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write.

6.2 Applications

This section describes the applications used by few of the WP8 prototypes. This section gives the general overview of the applications. Further descriptions are given under each of the prototype sections.

6.2.1 *GADGET*

GADGET is a freely available code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory written by Volker Springel [31] [32], Max-Plank-Institute for Astrophysics, Garching, Germany. GADGET uses an explicit communication model that is implemented with the standardized MPI communication interface. The code can be run on essentially all supercomputer systems presently in use, including clusters of workstations or individual PCs.

GADGET computes gravitational forces with a hierarchical tree algorithm (optionally in combination with a particle-mesh scheme for long-range gravitational forces) and represents fluids by means of smoothed particle hydrodynamics (SPH). The code can be used for studies of isolated systems, or for simulations that include the cosmological expansion of space, both with or without periodic boundary conditions. In all these types of simulations, GADGET follows the evolution of a self-gravitating collisionless N-body system, and allows gas dynamics to be optionally included. Both the force computation and the time stepping of GADGET are fully adaptive, with a dynamic range that is, in principle, unlimited.

GADGET can therefore be used to address a wide array of astrophysics interesting problems, ranging from colliding and merging galaxies, to the formation of large-scale structure in the Universe. With the inclusion of additional physical processes such as radiative cooling and heating, GADGET can also be used to study the dynamics of the gaseous intergalactic medium or to address star formation and its regulation by feedback processes.

6.2.2 NAMD

NAMD [33] (NAnoscale Molecular Dynamics) is a molecular dynamics program designed for high performance simulation for parallel computers. It is written using the Charm++ parallel programming model, noted for its parallel efficiency, and often used to simulate large systems (millions of atoms). It has been developed by the joint collaboration of the Theoretical and Computational Biophysics Group (TCB) and the Parallel Programming Laboratory (PPL) at the University of Illinois at Urbana-Champaign.

NAMD uses spatial decomposition coupled with a multithreaded, message-driven design, which is shown to scale efficiently to multiple processors. Also, NAMD incorporates the Distributed Parallel Multipole Tree Algorithm for full electrostatic force evaluation in O(N) time. NAMD can be connected via a communication system to a molecular graphics program in order to provide an interactive modelling tool for viewing and modifying a running simulation.

6.2.3 RAxML

RAxML-VI-HPC (randomized accelerated maximum likelihood for high performance computing) is a sequential and parallel program for inference of large phylogenies with maximum likelihood (ML).

Phylogenetic trees are used to represent the evolutionary history of a set of organisms. The reconstruction of such trees is usually based on an alignment of DNA or protein sequences from these organisms. Due to the rapid growth of sequence data over the last years there is an increasing demand to compute large trees that often comprise more than 1000 organisms and sequence data from several genes (so-called multi-gene alignments). Since alignments continuously grow in the number of organisms and in sequence length the application of high-performance computing techniques is essential to keep pace with the forthcoming data flood.

RAxML [34] is currently among the fastest programs for phylogenetic inference under the Maximum Likelihood (ML) criterion that has repeatedly been shown to be one of the most accurate models for phylogeny reconstruction. Efficient parallelization for OpenMP, PThreads, and MPI exist that already proved to scale well up to 2000 processors on a wide variety of architectures including GPUs, the Cell BE, SGI Altix4700, and IBM BlueGene/L.

The performance of RAxML's computational kernels has steadily been optimized over the last years. Though the main development focus has been on x86-based architectures, most optimizations are very generic and thus other architectures also benefit from it. For example, the computation intensive loops have been simplified and partially rewritten in order to facilitate compiler-based loop vectorization. The MPI version of RAxML uses a master/worker concept and as such makes frequent use of collective communication calls. Wherever possible, the most specific MPI call has been used, so that the application requires no knowledge about the communication infrastructure (network topology, bandwidth, latency, etc.) but leaves all decisions to the MPI runtime environment.

6.2.4 DL-POLY

DL_POLY [35] is a general purpose serial and parallel molecular dynamics simulation package developed at Daresbury Laboratory by W. Smith, T.R. Forester and I.T. Todorov. The Molecular Simulation Group (now part of the Computational Chemistry Group, MSG) developed the original package at Daresbury Laboratory under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5). The Natural Environment Research Council through the eMinerals project also supported later developments. The package is the property of the Central Laboratory of the Research Councils.

Two versions of DL_POLY are available, DL_POLY2, which has been parallelized using the Replicated Data strategy and is useful for simulations of up to 30,000 atoms on 100 processors and DL_POLY3, which uses Domain Decomposition to achieve parallelism and is suitable for simulations of order 1 million atoms on 8-1024 processors.