SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2012-2.3.1 – Third Implementation Phase of the European High Performance Computing (HPC) service PRACE

PRACE-3IP

PRACE Third Implementation Phase Project

Grant Agreement Number: RI-312763

D7.2.2
Exploitation of HPC Tools and Techniques

Final

Version: 1.1
Author(s): Michael Lysaght, ICHEC
Bjorn Lindi, SIGMA-NTNU
Vit Vondrak, VSB
John Donners, SURFSARA
Marc Tajchman, GENCI-CEA
Date: 20.11.2014

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="6">PRACE Project</td><td colspan="2">Project Ref. №: RI-312763</td></tr>
<tr><td colspan="2">Project Title: PRACE Third Implementation Phase Project</td></tr>
<tr><td colspan="2">Project Web Site: http://www.prace-project.eu</td></tr>
<tr><td colspan="2">Deliverable ID: < 7.2.2></td></tr>
<tr><td colspan="2">Deliverable Nature: <Report ></td></tr>
<tr><td>Deliverable Level:
PU</td><td>Contractual Date of Delivery:
31 / 05 / 2014
Actual Date of Delivery:
31 / 05 / 2014</td></tr>
<tr><td colspan="2">EC Project Officer: Leonardo Flores Añover</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

<table>
<tr><td rowspan="6">Document</td><td colspan="2">Title: Exploitation of HPC Tools and Techniques</td></tr>
<tr><td colspan="2">ID: D7.2.2</td></tr>
<tr><td>Version: <1.1></td><td>Status: Final</td></tr>
<tr><td colspan="2">Available at: http://www.prace-project.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word 2007</td></tr>
<tr><td colspan="2">File(s): D7.2.2.docx</td></tr>
<tr><td>Authorship</td><td>Written by:</td><td>Michael Lysaght, ICHEC
Bjorn Lindi, SIGMA-NTNU
Vit Vondrak, VSB
John Donners, SURFSARA
Marc Tajchman, GENCI-CEA</td></tr>
</table>

	Contributors:	I Bethune, EPCC F Reid, EPCC C Basu, SNIC-LiU S-H Ko, SNIC-LiU C Moulinec, STFC Y Fournier, EDF B Benek Gursoy, ICHEC H Nagel, NTNU A Kwiecien, WCSS M Uchronski, WCSS M Gebarowski, WCSS S Szkoda, WCSS Z Koza, IFT M Tykierko, WCSS P Nolan, ICHEC A McKinstry, ICHEC J-C Meyer, NTNU C Ozturan, Bogazici S Soner, Bogazici A Ronovsky, VSB T Karasek, VSB D Horak, VSB P Petkov, NCSA I Todorov, STFC D Grancharov, NCSA N Ilieva, NCSA E Likova, NCSA L Litov, NCSA S Markov, NCSA A Duran, ITU-UHeM S Celebi, ITU-UHeM S Piskin, ITU-UHeM M Tuncel, ITU-UHeM A Abdel-Rehim, CASTORC G Koutsou, CASTORC C Urbach, University of Bonn G V Demirci, Bilkent A Turk, Bilkent, R Oguz Selvitopi, Bilkent K Akbudak, Bilkent C Aykanat, Bilkent T Ponweiser, JKU P Jovanovic, IPB T Arslan, NTNU J Rodriguez, BSC
	Reviewed by:	David Vincente, BSC Thomas Eickermann, JUELICH
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	11/05/2014	Draft	First draft for PMO review
1.0	24/05/2014	Final version	PMO Review comments addressed
1.1	20/11/2014	After review	Broken references corrected after review prior to publication

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] M. Lysaght, B. Lindi, V. Vondrak, J. Donners, M. Tajchman, PRACE-3IP D7.2.1 A Report on the Survey of HPC Tools and Techniques, pdf: http://www.prace-project.eu/IMG/pdf/d7.2.1.pdf

[2] PRACE Whitepapers can be found at: http://www.prace-project.eu/white-papers

[3] G. Mozdzynski, A PGAS implementation of the ECMWF Integrated Forecasting System (IFS), CRESTA report, pdf: http://www.easc2013.org.uk/sites/default/files/Pdfs/InvitedTalks/EASC-Mozdzynski.pdf

[4] CP2K homepage: http://www.cp2k.org

[5] J. VandeVondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing, J. Hutter, QUICKSTEP: Fast and accurate density functional calculations using a mixed Gaussian and plane waves approach. Comp. Phys. Comm. 167 (2): 103-128 (2005)

[6] Intel Xeon Phi Specifications; http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html

[7] F. Reid and I. Bethune, Optimising CP2K for the Intel Xeon Phi, PRACE White Paper, 2013, pdf: http://www.prace-ri.eu/IMG/pdf/wp140.pdf

[8] Code_Saturne homepage: http://code-saturne.org/cms/

[9] MPI 3.0 standard, pdf: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

[10] ONE_TUBE test case reference:_http://cfd.mace.manchester.ac.uk/cgi-bin/cfddb/prpage.cgi?78&EXP&&database/cases/case78&cas78_head.html&cas78_desc.html&cas78_meth.html&cas78_data.html&cas78_refs.html&cas78_rsol.html&1&1&1&1&1&unknown

[11] DL_POLY homepage: http://www.stfc.ac.uk/CSE/randd/ccg/software/DL_POLY/25526.aspx

[12] C. Kartsaklis, G. Civario, Porting DL POLY on GPGPU with CUDA, presented at NVIDIA GPU Technology Conference (GTC), San Jose, (2010)

[13] The OpenACC homepage: http://www.openacc-standard.org/

[14] I. Bethune, A. Carter, X. Guo, P. Korosoglou, Million Atom KS-DFT with CP2K, PRACE whitepaper, PRACE-1IP, pdf: http://www.prace-ri.eu/IMG/pdf/Million_Atom_KS-DFT_with_CP2K.pdf

[15] I. Bethune, “Improving the scalability of CP2K on multi-core systems, dCSE Project, 2010, pdf: http://www.hector.ac.uk/cse/distributedcse/reports/cp2k02/cp2k02_final_report.pdf

[16] OpenCL Khronos Group homepage, http://www.khronos.org/opencl

[17] U. Borštnik, J. VandeVondele, V. Weber, J. Hutter, Sparse Matrix Multiplication: The Distributed Block-Compressed Sparse Row Library, Parallel Computing, 40(5-6): 47-58 (2014)

[18] The OpenACC 2.0a Specification (Corrected), pdf: http://www.openacc.org/sites/default/files/OpenACC.2.0a_1.pdf

[19] The Khronos OpenCL Registry with OpenCL 2.0 specification, http://www.khronos.org/registry/cl/

[20] Portland Group’s PGI Accelerator compilers: http://www.pgroup.com/resources/accel.htm

[21] Allinea DDT homepage: http://www.allinea.com/products/ddt/

[22] PGDBG Graphical Symbolic Debugger: http://www.pgroup.com/products/pgdbg.htm

[23] F. Reid, I. Bethune, Evaluating CP2K on Exascale Hardware: Intel Xeon Phi", PRACE whitepaper (2014) pdf: http://www.prace-project.eu/IMG/pdf/wp152.pdf
[24] U. Frisch, B. Hasslacher, and Y. Pomeau, Lattice gas automata for the Navier-Stokes equation. Phys. Rev. Lett., 56, 1505 (1986)

[25] G. Kohring, The cellular automata approach to simulating fluid flows in porous media. Physica A, 186, 97–108 (1992)

[26] Portland Group (2010) PGI Fortran & C Accelerator Programming Model new features ver. 1.3.

[27] M. G. B. Johnson, D. P. Playne, and K. A. Hawick, Data-parallelism and GPUs for lattice gas fluid simulations. Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA10), Las Vegas, USA, 12-15 July, pp. 210–216. CSREA. PDP4521 (2010)

[28] EC-Earth homepage: http://www.ec-earth.org/
[29] http://www.ecmwf.int/research/ifsdocs/CY40r1/
[30] G. Madec, NEMO ocean engine, Note du Pole de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 27 ISSN No 1288-1619 (2008) pdf: NEMO_book_3_4.pdf
[31] M Vancoppenolle, S. Bouillon, T. Fichefet, H. Goosse, O. Lecomte, M. A. Morales Maqueda, and G Madec, LIM The Louvain-la-Neuve sea Ice Model. Note du Pole de modélisation, Institut Pierre-Simon Laplace (IPSL), (2012) France, No 31 ISSN No 1288-1619.
[32] Vancoppenolle, M., T. Fichefet, H. Goosse, S. Bouillon, G. Madec, and M.A. Morales Maqueda, Simulating the mass balance and salinity of Arctic and Antarctic sea ice. 1. Model description and validation. Ocean Modelling, 27, 33-53 (2009)
[33] S. Valcke, T Craig, L. Coquart, OASIS3-MCT User Guide, OASIS3-MCT 2.0, Technical Report, TR/CMGC/13/17, CERFACS/CNRS SUC URA No 1875, Toulouse, France (2013)
[34] LULESH webpage: https://codesign.llnl.gov/lulesh.php
[35] ALE3D webpage: https://wci.llnl.gov/codes/ale3d/
[36] I. Karlin, A. Bhatele, J. Keasler, Bradford L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, C. H. Still: *Exploring Traditional and Emerging Parallel Programming Models using a Proxy Application,* proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing
[37] Programming Models at BSC, http://pm.bsc.es/
[38] B. Dunn and R. Yasser, Learning and inference in a nonequilibrium Ising model with hidden nodes, Physical Review E, 2013
[39] J. Bell L Chacon R. Falgout M. Heroux P. Hovland E. Ng C Webster S. Wild J. Dongarra, J Hittinger. Applied mathematics research for exascale computing report (2014) pdf: http://www.netlib.org/utk/people/JackDongarra/PAPERS/doe-exascale-math-report.pdf
[40] OpenFoam homepage: http://www.openfoam.org/
[41] NetGen, Mesh Generator hompage: http://www.hpfem.jku.at/netgen/
[42] Parallel Mesh Multiplication and its Implementation in Code Saturne. A. Ronovsky, P. Kabelikova, V.Vondrak, C. Moulinec, Civil-Comp Proceedings ISSN 1759-3433. (doi:10.4203/ccp.101.11)
[43] Code_Aster homepage: http://www.code-aster.org/
[44] Elmer homepage: http://www.csc.fi/english/pages/elmer
[45] FFLOP homepage: http://industry.it4i.cz/produkty/fllop/
[46] PETSc homepage: http://www.mcs.anl.gov/petsc/
[47] S. Piskin, M. S. Celebi, Analysis of the effects of different pulsatile inlet profiles on the hemodynamical properties of blood flow in patient specific carotid artery with stenosis,

Computers in Biology and Medicine, Volume 43, Issue 6, 1 July 2013, Pages 717-728, ISSN 0010-4825, http://dx.doi.org/10.1016/j.compbiomed.2013.02.014
[48] S. Piskin, M. S. Celebi, "Numerical blood flow simulation with predefined artery movement," Biomedical Engineering and Informatics (BMEI), 2012 5th International Conference, pp.654,658, 16-18 Oct. 2012 doi: 10.1109/BMEI.2012.6513039
[49] H. Turkeri, S. Piskin, and M. S. Celebi, A comparison between non-Newtonian and Newtonian blood viscosity models, Journal of Biomechanics, 44, Supplement 1, 2011
[50] S. Piskin and A. Akkus¸ Biofluid flow applications by open-source software, 17. National Biomedical Engineering Meeting - BIYOMUT 2012, Istanbul, Turkey, October 3-5, 2012
[51] P. Dagna and J. Hertzer, Evaluation of multi-threaded OpenFOAM hybridization for massively parallel architectures, PRACE WP98, Aug. 20, 2013, www.prace-project.eu/IMG/pdf/wp98.pdf
[52] M. Manguoglu, PRACE WP, Sep. 6, 2012, http://www.prace-project.eu/IMG/pdf/A_General_Sparse_Sparse_Linear_System_Solver_and_Its_Application_in_OpenFOAM-2.pdf
[53] M. Culpo, PRACE WP, Sep. 6, 2012, http://www.prace-ri.eu/IMG/pdf/Current_Bottlenecks_in_the_Scalability_of_OpenFOAM_on_Massively_Parallel_Clusters-2.pdf
[54] M. Moyles, P. Nash, and Ivan Girotto, PRACE WP, Sep. 6, 2012, http://www.prace-ri.eu/IMG/pdf/Performance_Analysis_of_Fluid-Structure_Interactions_using_OpenFOAM.pdf
[55] T. Behrens, OpenFOAM's basic solvers for linear systems of equations: Solvers, preconditioners, smoothers, Tech. Rep. DTU, Denmark, Feb. 18, 2009, http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/TimBehrens/tibeh-report-fin.pdf
[56] Xiaoye S. Li and James W. Demmel, SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems, ACM Trans. on Math. Software, Vol. 29, No. 2, June 2003, pp. 110-140.
[57] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, and I. Yamazaki, SuperLU Users' Guide, Tech. Report UCB, Computer Science Division, University of California, Berkeley, CA, 1999, update: 2011
[58] A. Duran, M.S. Celebi, M. Tuncel and B. Akaydın, Design and implementation of new hybrid algorithm and solver on CPU for large sparse linear systems, PRACE-2IP white paper, Libraries, WP 43, July 13, 2012, http://www.prace-ri.eu/IMG/pdf/wp43-newhybridalgorithmfo_lsls.pdf
[59] A. Duran, M.S. Celebi, M. Tuncel, and F. Oztoprak. Structural analysis of large sparse matrices for scalable direct solvers. PRACE-2IP white paper, Scalable algorithms, WP 82, August 20, 2013, http://www.prace-project.eu/IMG/pdf/wp82.pdf
[60] Andreas Frommer, et. al., "Adaptive aggregation based domain decomposition multigrid for the lattice Wilson Dirac operator", arXiv: 1303.1377[hep-lat].
[61] K. Jansen and C. Urbach, "tmLQCD: A Program suite to simulate Wilson Twisted mass Lattice QCD", Comput. Phys. Com. 180(2009)2717-2738 (arXiv: 0905.3331).
[62] tmLQCD homepage: https://github.com/etmc/tmLQCD.
[63] https://confluence.csc.fi/display/HPCproto/HPC+Prototypes.
[64] J Cohen, Graph twiddling in a mapreduce world. Computing in Science & Engineering, 11(4), 29-41 (2009)
[65] Ekanayake, J., & Fox, G. (2010). High performance parallel computing with clouds and cloud technologies. In Cloud Computing (pp. 20-38). Springer Berlin Heidelberg.
[66] Kang, U., Tsourakakis, C. E., & Faloutsos, C. (2009, December). Pegasus: A peta-scale graph mining system implementation and observations. In Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on (pp. 229-238). IEEE.

[67] Tu, T., Rendleman, C. A., Borhani, D. W., Dror, R. O., Gullingsrud, J., Jensen, M. O., and Shaw, D. E. (2008, November). A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories. In High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for (pp. 1-12). IEEE.
[68] Plimpton, S. J., & Devine, K. D. (2011). MapReduce in MPI for large-scale graph algorithms. Parallel Computing, 37(9), 610-632.
[69] Hoefler, T., Lumsdaine, A., & Dongarra, J. (2009). Towards efficient mapreduce using mpi. In Recent Advances in Parallel Virtual Machine and Message Passing Interface (pp. 240-249). Springer Berlin Heidelberg.
[70] Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing, 20(1), 359-392.
[71] Catalyurek, U. V., & Aykanat, C. (1999). Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. Parallel and Distributed Systems, IEEE Transactions on, 10(7), 673-693.
[72] Davis, T. A., & Hu, Y. (2011). The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS), 38(1), 1.
[73] TAU homepage: http://www.cs.uoregon.edu/research/tau/home.php
[74] HPCToolKit homepage: http://hpctoolkit.org/
[75] Orio homepage: http://brnorris03.github.io/Orio/
[76] Moulinec, C., Sunderland, A. G., Kabelikova, P., Ronovsky, A., Vondrak, V., Turk, A., Aykanat, C., and Theodosiou, C., 2012, Optimization of Code_Saturne for Petascale Simulations, PRACE white paper (2012), pdf: http://www.prace-project.eu/IMG/pdf/Optimisation_of_Code_Saturne_for_Petascale_Simulations.pdf
[77] Alya System homepage: http://www.bsc.es/computer-applications/alya-system
[78] PRACE UEABS, Unified European Applications Benchmark Suite, http://www.prace-ri.eu/ueabs?lang=en
[79] Extrae, http://www.bsc.es/computer-sciences/extrae
[80] Paraver, http://www.bsc.es/computer-sciences/performance-tools/paraver
[81] Wautelet, P., Kestener, P., 2012, Parallel IO performance and scalability study on the CURIE supercomputer, PRACE white paper, pdf: http://www.prace-ri.eu/IMG/pdf/Parallel_IO_performance_and_scalability_study_on_the_PRACE_CURIE_supercomputer-2.pdf
[82] J. Bédorf, E. Gaburov, S. Portegies Zwart, A sparse octree gravitational N-body code that runs entirely on the GPU processor, J. Comp. Pys., 231, 2825–2839 (2011)
[83] W. Frings, F. Wolf, V. Petkov, Scalable Massively Parallel I/O to Task-Local File, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 17:1-11 (2009)

List of Acronyms and Abbreviations

AAA	Authorization, Authentication, Accounting.
ACF	Advanced Computing Facility
ADP	Average Dissipated Power
AISBL	Association sans but lucrative (legal form of the PRACE RI)
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)
CAF	Co-Array Fortran
CAL	Compute Abstraction Layer
CCE	Cray Compiler Environment
ccNUMA	cache coherent NUMA
CEA	Commissariat à l'énergie atomique et aux énergies alternatives
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)
CUDA	Compute Unified Device Architecture (NVIDIA)
DARPA	Defense Advanced Research Projects Agency
DDN	DataDirect Networks
DDR	Double Data Rate
DEISA	Distributed European Infrastructure for Supercomputing Applications. EU project by leading national HPC centres.
DFT	Density Functional Theory
DGEMM	Double precision General Matrix Multiply
DIMM	Dual Inline Memory Module
DMA	Direct Memory Access
DNA	DeoxyriboNucleic Acid
DOE	Department of Energy (USA)
DP	Double Precision, usually 64-bit floating point numbers
DRAM	Dynamic Random Access memory
EC	European Commission
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)
ESFRI	European Strategy Forum on Research Infrastructures; created roadmap for pan-European Research Infrastructure.
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
GCS	Gauss Centre for Supercomputing (Germany)
GDDR	Graphic Double Data Rate memory
GÉANT	Collaboration between National Research and Education Networks to build a multi-gigabit pan-European network, managed by DANTE. GÉANT2 is the follow-up as of 2004.
GENCI	Grand Equipement National de Calcul Intensif (France)
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
GS	Gram-Schmidt
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
HET	High Performance Computing in Europe Taskforce. Taskforce by representatives from European HPC community to shape the European HPC Research Infrastructure. Produced the scientific case and valuable groundwork for the PRACE project.
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 program)
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 Exascale Project
IL	Intermediate Language
IMB	Intel MPI Benchmark
I/O	Input/Output
IOR	Interleaved Or Random
IPMI	Intelligent Platform Management Interface
ISC	International Supercomputing Conference; European equivalent to the US based SCxx conference. Held annually in Germany.
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)
LBE	Lattice Boltzmann Equation
LINPACK	Software library for Linear Algebra
LLNL	Laurence Livermore National Laboratory, Livermore, California (USA)
LQCD	Lattice QCD
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
LS	Local Store memory (in a Cell processor)
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
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
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)
MoU	Memorandum of Understanding.
MPI	Message Passing Interface
MPP	Massively Parallel Processing (or Processor)
MPT	Message Passing Toolkit
MRAM	Magnetoresistive RAM
MTAP	Multi-Threaded Array Processor (ClearSpead-Petapath)
mxm	DP matrix-by-matrix multiplication mod2am of the EuroBen kernels

NAS	Network-Attached Storage
NCF	Netherlands Computing Facilities (Netherlands)
NDA	Non-Disclosure Agreement. Typically signed between vendors and customers working together on products prior to their general availability or announcement.
NoC	Network-on-a-Chip
NFS	Network File System
NIC	Network Interface Controller
NUMA	Non-Uniform Memory Access or Architecture
OpenCL	Open Computing Language
OpenGL	Open Graphic Library
Open MP	Open Multi-Processing
OS	Operating System
OSS	Object Storage Server
OST	Object Storage Target
PCIe	Peripheral Component Interconnect express, also PCI-Express
PCI-X	Peripheral Component Interconnect eXtended
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)
PRACE	Partnership for Advanced Computing in Europe PRACE-PP PRACE Preparatory Phase; Project Acronym
PSNC	Poznan Supercomputing and Networking Centre (Poland)
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 Direct Memory Access
RI	Research Infrastructure
RISC	Reduce Instruction Set Computer
RNG	Random Number Generator
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
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
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
SPU	Synergistic Processor Unit (in each SPE)
SSD	Solid State Disk or Drive
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
STRATOS	PRACE advisory group for STRAtegic TechnOlogieS
STT	Spin-Torque-Transfer
SURFsara	Dutch national High Performance Computing & e-Science Support Center
TARA	Traffic Aware Routing Algorithm
TB	Tera (= 240 ~ 1012) Bytes (= 8 bits), also TByte
TCO	Total Cost of Ownership. Includes the costs (personnel, power, cooling, maintenance, ...) in addition to the purchase cost of a system.
TDP	Thermal Design Power
TFlop/s	Tera (= 1012) 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
UFM	Unified Fabric Manager (Voltaire)
UNICORE	Uniform Interface to Computing Resources. Grid software for seamless access to distributed resources.
UPC	Unified Parallel C
UV	Ultra Violet (SGI)
VHDL	VHSIC (Very-High Speed Integrated Circuit) Hardware Description Language

Executive Summary

The objective of PRACE-3IP Work Package 7 (WP7) 'Application Enabling and Support' is to provide applications enabling support for HPC applications codes which are important for European researchers to ensure that these applications can effectively exploit multi-petaflop systems. This applications enabling activity uses the most promising tools, algorithms and standards for optimisation and parallel scaling that have recently been developed through research and experience in PRACE and other projects.

In this deliverable, we report on the exploitation of new HPC tools and algorithms on different codes that are of interest to the European scientific and engineering research community. In this sense, the report here follows on naturally from the T7.2 deliverable D7.2.1, 'A Report on the Survey of HPC Tools and Techniques', which represented the first phase of activity in T7.2. Indeed, much of the exploitation work reported on here, was inspired by the comprehensive and in-depth analysis of state-of-the-art HPC tools and techniques as reported in D7.2.1.

The report on the exploitation of state-of-the-art HPC tools and techniques presented here represents the second phase of activity in T7.2. In this report we summarise how selected state-of-the-art HPC tools and techniques fared on real-world applications during the exploitation phase of T7.2, where we focus on four separate topics that we have identified as being important to enable applications within WP7 on the road to exascale, and which mirror the four topics reported on in the survey of HPC Tools and Techniques in D7.2.1. These are: (1) Programming Models, (2) Scalable Libraries and Algorithms, (3) Debuggers and Profilers and finally, (4) I/O Management Techniques. For a more detailed description of each of the exploitation projects summarised here, we refer the reader to the PRACE-3IP whitepaper associated with each of the 17 projects.

Programming Models

During the second phase of T7.2, we have exploited several different programming models that were reported on in D7.2.1 as having genuine potential on the road to exascale. In this deliverable we provide summary reports on the effectiveness of each of these HPC tools when enabling real applications with future exascale challenges in mind. In particular we have focused on probing new (as well as under-exploited) features in mature programming models, such as the Message Passing Interface (MPI), the new features of which are now starting to confront the challenges of exascale computing.

We have also exploited programming models targeting many-core architectures (where many-core typically implies > 50 cores), which are likely to continue to feature as part of future large-scale systems as we move into the deep petascale era. As pointed out in the first phase of T7.2, the entry of new competitors to the many-core space has increased the relevance of open standards on the road to exascale and we have therefore placed a particular focus on both mature and emerging open standards during the exploitation phase. In terms of more novel approaches to exploiting multi-petascale systems, we have also continued to be inspired by experimental programming models featuring in European exascale projects, which offer experimental task-based models for programming multi-/many-core architectures. We feel that it is worth also noting that, although possibilities for exploiting Partitioned Global Address Space (PGAS) languages on real applications were genuinely explored during the exploitation phase, no real opportunities arose for doing so, possibly reflecting the continuing challenge for exploiting these powerful tools on existing large-scale codes on the road to exascale.

Scalable Libraries and Algorithms

During the exploitation phase we have undertaken six separate enablement projects that have each focused on exploiting scalable libraries and algorithms. In terms of challenges on the road to exascale, global communications, in particular, are known to be a severe barrier when trying to scale across large core counts and many open questions still exist on how, for example, Fast Fourier Transform (FFT) libraries will perform on future exascale systems. With this challenge in mind, we are happy to report on the successful implementation of alternative methods to FFT libraries in a real molecular dynamics application, which has the potential to significantly improve scalability (and functionality) of the code on large node counts for certain problem types. As well as global communications, mesh generation and refinement have also been identified as posing major challenges on the road to exascale and as a result, we have also focused our efforts on both exploiting and improving state-of-the-art mesh tools for enabling Computational Fluid Dynamics (CFD) codes, which we report on here. We also report on the successful implementation of an Alegbraic Multi-Grid (AMG) algorithm within a lattice Quantum Chromo-Dynamics (QCD) code, which has been shown to outperform existing techniques and shows real potential for enabling QCD applications on the road to exascale.

Debuggers and Profilers

In the survey of state-of-the-art HPC tools and techniques as reported in D7.2.1 we found that all of the European exascale projects are concentrating effort on tools for debugging and performance analyses. This is deemed a necessity for efficient use of multi-petascale and future exascale systems: If we are to enable applications on such systems, then we need to have as clear a view as possible of the barriers to achieving performance. At the same time, we noted in D7.2.2, that very little effort had gone into documenting the experience of using such tools on real applications within PRACE to date. Here, we try to rectify this by reporting on how state-of-the art profiling tools fared with respect to real large scale CFD and Computational Structural Mechanics (CSM) codes. We also report on how such tools can potentially be employed in combination with auto-tuning tools, which are becoming of increasing interest on the road to exascale.

I/O Management Techniques

During our surveying in the first phase of T7.2 and as reported on in D7.2.2, we found that users within PRACE have in general not been able to squeeze as much performance from existing parallel file systems as they have from computational hardware, particularly for the case of high-level I/O libraries. With this challenge in mind, we have carried out deeper investigations into extracting performance from file systems using state-of-the-art high-level libraries, work that we are happy to report has improved the I/O performance of an astrophysics application on Tier-0 systems and shows real promise on the road to exascale.

1 Introduction

1.1 The Purpose of the document

The objective of PRACE-3IP Work Package 7 (WP7) 'Application Enabling and Support' is to provide applications enabling support for HPC applications codes which are important for European researchers to ensure that these applications can effectively exploit multi-petaflop systems. This applications enabling activity used the most promising tools, algorithms and standards for optimisation and parallel scaling that have recently been developed through research and experience in PRACE and other projects.

There has been significant research activity undertaken both within PRACE and outside PRACE investigating novel techniques to enable applications on petascale and future exascale systems. Such activities include, for example, PRACE Work Packages [WP6 ('Software Enabling for Petaflop/s Systems') in PRACE-PP, WP7 ('Enabling Petascale Applications: Efficent Use of Tier-0 Systems') and WP9 ('Future Technologies') in PRACE-1IP, WP7 ('Scaling Applications for Tier-0 and Tier-1 Users'), WP8 ('Community Codes'), and WP12 ('Novel Programming Techniques') in PRACE-2IP], other EU-funded projects (European Exascale Software Initiative (EESI), Towards Exaflop applications (TEXT), Collaborative Research into Exascale Systemware, Tools and Applications (CRESTA), Dynamical Exascale Entry Platform (DEEP), Mont-Blanc and international collaborations such as the International Exascale Software Project (IESP).

As stated in the Description of Work (DoW), the objective of this deliverable, D7.2.2, is to report on the exploitation of new HPC tools and algorithms on different codes. In this sense, the report here follows on naturally from the T7.2 deliverable D7.2.1, 'A Report on the Survey of HPC Tools and Techniques' [1] which represented the first phase of activity in T7.2. Indeed, much of the exploitation work reported on here, was inspired by the comprehensive and in-depth analysis of state-of-the-art HPC tools and techniques as provided in D7.2.1.

The report on the exploitation of HPC tools and techniques presented here represents the second phase of activity in T7.2. Here, we report on four separate topics that are important to enabling applications within WP7, and which mirror the four topics reported on in the survey of HPC Tools and Techniques in D7.2.1. These are: (1) Programming Models, (2) Scalable Libraries and Algorithms, (3) Debuggers and Profilers and finally, (4) I/O Management Techniques. In this report we summarise how selected HPC tools and techniques fared on real-world applications during the exploitation phase of T7.2. In this way, we hope to provide, primarily PRACE partners, with information that should hopefully stimulate further interest when considering the tools and techniques for furthering enabling projects. We also hope that the report will be of interest to European HPC users and more generally.

1.2 Organisation of Work

Much of the work during the exploitation phase ("phase 2") of T7.2 followed on naturally from phase 1 and was organized in four subtasks:

- Subtask 7.2.A 'Debuggers and Profilers' (lead by Bjorn Lindi, SIGMA-NTNU)
- Subtask 7.2.B: 'Programming Languages and Standards' (lead by Marc Tajchman, GENCI)
- Subtask 7.2.C: 'Scalable Libraries and Algorithms' (lead byVit Vondrak, VSB)
- Subtask 7.2.D: 'I/O Management Techniques' (lead by John Donners, SURFSARA)

PRACE-3IP Face-to-Face Meeting, Warsaw, Poland

Following the submission of deliverable, D7.2.1 'A Report on the Survey of HPC Tools and Techniques' in M11 (April 2013), a PRACE 3IP WP7 Face-to-Face Meeting was held in Warsaw, Poland in April 2013, where the conclusions of D7.2.1 were analysed in detail and where T7.2 partners were tasked with the following actions:

- Select a promising HPC tool(s)/technique(s) from the Survey
- Identify an application that has potential to scale on large compute systems and which also has relevance to the European scientific and engineering research community
- Produce a Description of Work for the exploitation project
- Produce a working title and abstract for the PRACE-3IP whitepaper associated with the project.

At the Face-to-Face meeting in Warsaw it was agreed that the exploitation projects should be inspired by the conclusions drawn from the analysis of state-of-the-art HPC tools and techniques as reported on in D7.2.1 and that the project should continue to be inspired by work taking place outside PRACE, particularly in the area of exascale research.

During the meeting in Warsaw, a new PRACE 3IP D7.2.2 page was set up on the PRACE wiki in order for monthly project progress reports to be uploaded. The D7.2.2 wiki page also contains an overall workplan and logistical information for the exploitation, including a "phase 2" schedule which was agreed on in Warsaw and which can be seen in Figure 1.

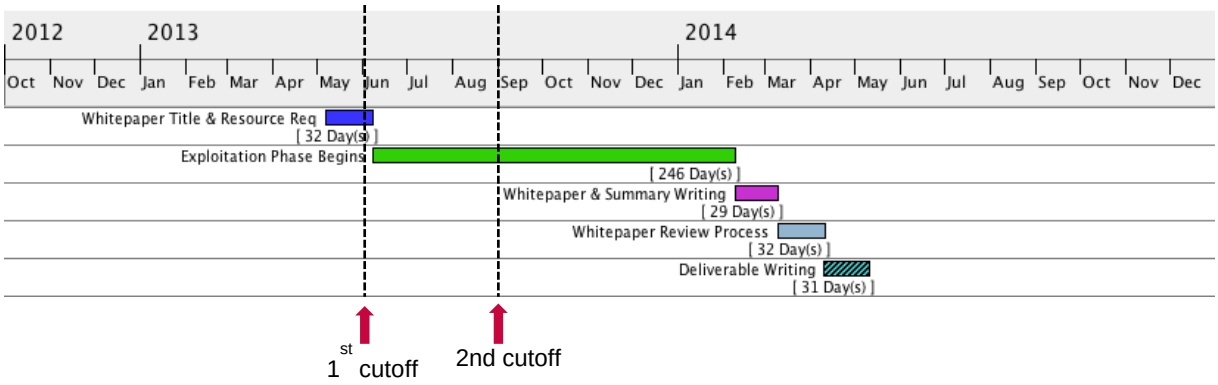

Figure 1: Gantt chart outlying first draft of the PRACE 3IP T7.2 exploitation phase schedule in April 2013. Taken from PRACE 3IP T7.2 Wiki page.

PRACE-3IP Face-to-Face Meeting, Dublin, Ireland

The second Face-to-Face Meeting during "phase 2" of T7.2 was held in Dublin, Ireland in December 2013. During the meeting progress reports were presented for each of the projects and each of the partners was tasked with the following actions:

- Produce final title and abstract for the PRACE-3IP whitepaper associated with the exploitation project.

The final list of 17 whitepaper titles along with author lists were sent to PRACE PMO in February 2014.

All whitepapers were completed and submitted for internal T7.2 review on April 4th 2014 and subsequently for final PRACE-3IP review on April 11th 2014.

1.3 Structure of the Document

The document presents four subsections which are aligned with the four subtasks within T7.2, and which mirror the four sections in deliverable D7.2.1. These are: Programming Models, Debuggers and Profilers, Scalable Libraries and Algorithms and I/O Management Techniques. Within each section, a short introduction is provided which further details the structure of the individual section, which is then followed by a collection of reports summarising the work and findings of each of the T7.2 projects during the exploitation phase.

1.4 Intended Audience

Our objective in preparing this report is to exploit the most promising HPC tools and techniques that may have applicability for petascaling applications within WP7. Targeted primarily at PRACE partners who are involved in enabling applications, it provides an overview of how a selection of state-of-the-art HPC tools and techniques fared when enabling real applications targeting petascale systems/future exascale systems. We also hope that the report here will be of interest to European HPC users and more generally.

2 Programming Models

In this section, we report on eight projects that have each focused on exploiting state-of-the-art programming languages, techniques and standards (hereafter referred to as programming models) in order to enable applications for multi-petaflop/future exascale systems. Each subsection provides a summary of the project along with a reference to the PRACE-3IP whitepaper associated with the project. We recommend that the reader also refers to the associated whitepaper for each project [2], which provides a more detailed report on the projects than is provided here. The list of programming models as well the applications that were enabled can be seen in Table 1

HPC Tool/Technique	*Application*
MPI/OpenMP hybrid	CP2K
MPI 2/3.0 One-sided communications	Code_Saturne
OpenACC/OpenCL	CP2K
OpenACC/CUDA	Cellular Automata Library for CFD
OpenACC/CUDA	DL_POLY_4
OpenMP	Neural Networks application
OmpSs	LULESH
OpenACC	EC-Earth3

Table 1: HPC Tools and Techniques (Programming Models) exploited along with corresponding applications

As highlighted in D7.2.1 [1], currently, one programming model still dominates PRACE application codes more than any other, namely, Single Program Multiple Data (SPMD) message passing using MPI for internode communication, increasingly employed alongside OpenMP for intra-node parallelism. MPI and OpenMP are mature standards and widespread expertise on their use can be found within PRACE. However, both standards are evolving and version 3.0 of MPI and version 4.0 of OpenMP have recently been ratified, with partial

implementations for both now being offered by several tool developers. The reporting on new features of both standards in D7.2.1, as well as on how both programming models have been employed inside and outside PRACE to date has not only inspired us to investigate the opportunities for exploiting new features of the standards, but has also motivated us to further investigate features that already existed in previous versions of the standards and have so far been under-exploited in PRACE to date. An interesting example case of the latter is the exploitation of one-sided communication features offered through MPI 2.0/3.0 as described in detail in the whitepaper 'Enabling Code_Saturne for Multi-petaflop/Exascale with MPI 3.0 One Sided Communications', and which is summarised later in this section.

While combining MPI and OpenMP is still considered to be the hybrid programming method of choice, the recent advent and rapid adoption of many-core coprocessors/accelerators in the design of multi-petaflop systems must increasingly be considered in order to exploit the full potential of the compute hardware space on emerging European multi-petascale/future exascale systems. To date, the challenge of exploiting such heterogeneous systems has typically been met within PRACE by augmenting the MPI/OpenMP hybrid model with an additional third model that targets the Single Instruction Multiple Thread (SIMT) architecture of GPUs thereby forcing the further extraction of hierarchical levels of parallelism in current PRACE applications.

In D7.2.1, it was found that, by far the most popular programming model for programming GPU architectures, both within and outside PRACE, is still NVIDIA's CUDA framework. While concerns are often voiced around the proprietary nature of CUDA, its ease of use and surrounding ecosystem is continually being improved upon and the language itself offers many elegant features for achieving performance on NVIDIA GPUs. There is, however, anecdotal evidence of concerns around the portability, programmability and maintainability of large-scale CUDA-based applications on the road to exascale. One possible solution to these issues is an open standard directive-based approach of which OpenACC represents the strongest offering to date. Interest in OpenACC is growing rapidly both inside and outside PRACE where this has been somewhat reinforced by the recent purchase of the Portland Group by NVIDIA in 2013. (Somewhat less clear at the time of writing is the much talked about roadmap for merging OpenACC and OpenMP into a single open standard). While the standard is still young, and in many cases does not result in the same level of performance as equivalent CUDA implementations, we have been stimulated by recent success stories, as reported on in D7.2.1, (including in DOE exascale projects) to further explore the technology and report here on how it has been exploited to enable four separate applications, namely, DL_POLY, CP2K, EC-Earth and a CFD application during phase 2 of T7.2. A particularly interesting potential discussed in the whitepaper 'An Analysis of State of the Art Tools for Preparing DL_POLY_4 for Exascale', as summarised in this section, is the capability to use OpenACC with CUDA in an interoperable way, possibly having implications for performance and maintainability of large-scale applications on the road to exascale.

The subject of open standards for programming heterogeneous systems has become quite a hot topic now that GPUs are not the only accelerator/many-core offering in town, since the arrival of Intel's Many Integrated Core (MIC) coprocessor (Xeon Phi) to the market in 2013. The Intel Xeon Phi is an x86-based architecture and so familiar open standards such as OpenMP and MPI can be used to program the device. This is currently not the case for GPUs. However, the OpenMP 4.0 standard does now support the targeting of accelerators in general and so, in theory, there is currently nothing to stop compiler support for programming GPUs via OpenMP. While both MPI and OpenMP can be used to program the Intel Xeon Phi, it is still unclear which is the best model for extracting performance and whether different models are better for different problem cases. One noteworthy fact that is emerging from the many Xeon Phi enablement projects in PRACE is that, in many cases, as much programming effort

is needed to obtain performance on the Xeon Phi as is required for programming GPUs. Three separate projects within T7.2 have focused on exploiting OpenMP 4.0 for enabling applications on the Intel Xeon Phi, including CP2K and a QCD application.

With regards to the MPI plus X paradigm for programming heterogeneous systems, we also report here on the exploitation of OmpSs, which is a programming model being developed at Barcelona Supercomputing Center (BSC) and is used in both the Mont-Blanc and DEEP exascale projects and in essence represents an effort to extend the OpenMP model with new directives to support asynchronous parallelism and heterogeneity. However, it can also be understood as new directives extending other accelerator based APIs, like CUDA or OpenCL. With regards to the latter standard, we still find it difficult to gauge whether OpenCL will be more aggressively exploited on the road to exascale. It is a standard that offers a lot in terms of the architectures it can target (including FPGAs as of 2013), but it still lacks a groundswell of support for its ecosystem and as such lags behind CUDA in terms of programmability. Perhaps this will change if Intel puts more support behind the standard as a tool for targeting its Xeon Phi hardware.

Finally, although not exploited during phase 2 of T7.2, we feel it is worth mentioning PGAS languages once again. As mentioned in D7.2.2, investigations into PGAS models have typically been exploratory in nature with no evidence of real applications being enabled with such models to date. (We have, however, found some exceptions to this within several of the exascale projects, e.g., ECMWF's IFS enablement with Co-Array Fortran [3]). Although possibilities for exploiting Partitioned Global Address Space (PGAS) languages on real applications were genuinely explored during the exploitation phase, no real opportunities arose for doing so, possibly reflecting the continuing challenge for exploiting these powerful tools on existing large-scale codes on the road to exascale. However, the benefit of a single language that can be used to efficiently target multi-petascale and future exascale heterogeneous systems in the entirety of their compute hardware space is quite obviously still a welcome prospect.

2.1 Porting CP2K to Intel Xeon Phi with mixed-mode MPI/OpenMP in native mode

WP152: *Evaluating CP2K on Exascale Hardware: Intel Xeon Phi*

Authors: I Bethune (EPCC) and Fiona Reid (EPCC)

Application: CP2K

HPC Tool/Technique: MPI, OpenMP, FFTW

Person Months: 4

CP2K [4] is a popular, open-source program for atomistic simulation. It provides many levels of theory ranging from classical potential models, DFT using a mixed Gaussian and plane waves approach known as QUICKSTEP [5], to hybrid DFT and post-HF methods (MP2, RPA). In addition, many simulation methods are supported including, molecular dynamics, Monte Carlo, path integrals, nudged elastic band and free energy calculations. These features have made CP2K an increasingly popular tool across the fields of materials science, computational chemistry, solid-state physics and biochemistry. It is widely used throughout Europe, including on the current PRACE RI systems, as well as national and local-scale HPC systems.

As a result, it is of interest to evaluate the performance of CP2K on potential future HPC architectures to ensure the continued viability of the code as HPC providers start looking towards Exascale. One such new architecture is the Intel Xeon Phi [6] (Many Integrated Core,

or MIC) platform, which combines the high performance and low power of e.g. GPUs, with the ease of use of standard x86 CPUs.

Intel supports several standard programming models on the Xeon Phi including the widely used MPI, OpenMP and OpenCL, as well as proprietary interfaces like Intel Thread Building Blocks and Cilk+. Since CP2K already has a mixed-mode MPI and OpenMP parallelisation strategy, we ported the code to the Xeon Phi Platform, and investigated the performance and scalability that could be achieved without significant code modification or tuning.

Since this work was carried out in Q2 2013, shortly after the public release of the Xeon Phi, the PRACE cluster EURORA was not available, and we used a smaller cluster `Dommic' at CSCS, consisting of 6 nodes each with a Xeon E5-2670 processor and 2 Xeon Phi 5110P co-processors.

As reported in our PRACE whitepaper, we uncovered and fixed a number of important bugs in the CP2K code, in the Intel Compiler and in Intel's MKL (all of which are now resolved with the Intel Composer XE 2013 SP1 release). We successfully ported CP2K to the Xeon Phi, running in native mode i.e. directly using the Xeon Phi as a many-core computing system, rather than as an accelerator. We evaluated several alternative Fast Fourier Transform libraries (FFTW 3.3.3 compiled for MIC, a pre-release FFTW 3.3.4 with MIC vectorisation support, and Intel MKL). FFTW 3.3.4 and MKL both gave up to a factor of 6 faster performance than the unoptimised FFTW 3.3.3, showing that effective use of the MIC-specific vector instruction set is critical.

When running in mixed-mode MPI/OpenMP mode on the Xeon Phi, the placement of MPI processes and OpenMP threads was found to be very important in achieving maximum performance. A 'balanced' approach where threads are kept close to the parent process while maintaining an even load balance over the Xeon Phi's virtual cores performed best, and this arrangement had to be constructed manually using the MKL_AFFINITY environment variable.

Finally, we carried out benchmarking using the H2O-64 ab-initio molecular dynamics test case. The best performance obtained on the Xeon Phi used 16 MPI processes each with 15 OpenMP threads, generating a total of 240 threads - fully utilising the Xeon Phi. However, using the host CPU with 16 MPI processes was still around 4x faster.

As a result we have shown that while porting an existing code to the Xeon Phi is relatively straightforward, further effort is needed to extract good performance from the MIC architecture. Analysis of individual routines in CP2K gave us a good understanding of what further development was needed, and work to implement these recommendations was undertaken in the PRACE-1IP Extension and is reported in White Paper 140 [7]

2.2 Exploiting MPI 3.0 One-sided Communication for enabling Code_Saturne on Multi-petaflop/Exascale systems

WP153: *Enabling Code_Saturne for Multi-petaflop/Exascale with MPI 3.0 One Sided Communications*

Authors: C Basu (SNIC-LiU), Soon-Heum Ko (SNIC LiU), C Moulinec (STFC) and Y Fournier (EDF)

Application: Code_Saturne

HPC Tool/Technique: MPI 3.0

Person Months: 2

Code_Saturne [8] is a well-known open-source code for solving Navier-Stokes equations in 2D, 2D-axisymmetric and 3D flows. It can handle steady or unsteady, laminar or turbulent, incompressible, compressible or weakly dilatable, isothermal or non-isothermal cases.

Code_Saturne is an MPI parallelized code, where its scalability naturally depends on efficient MPI communications. The newly released MPI 3.0 standard [9] has introduced improved Remote Memory Access (RMA) one-sided communication. The lower overhead associated with one-sided communication as compared to two-sided communication has the potential to increase the performance at peta/exascale by increasing the effective network bandwidth and reducing synchronization overheads. As part of this project, we have investigated the impacts of MPI one-sided communication on the Code_Saturne.

Substantial communication overhead of Code_Saturne comes from a halo exchange routine named `cs_halo_sync_var()`. Halo exchange in this routine is implemented by MPI point-to-point routines `MPI_Isend`, `MPI_Irecv` followed by `MPI_Wait`. In this project we have modified the `cs_halo_sync_var()` routine to replace MPI point-to-point communication with MPI one sided communications. We have implemented two versions of the modified routine using MVAPICH2 implementation of MPI, which has support for both MPI 2.0 and MPI 3.0. The first version (version 1) of our modified routine uses MPI-2.0 one-sided communication routines. For our tests we have taken a test case called “One Tube”. The configuration corresponds to the flow in a staggered bundle of tubes [10]. All our tests were carried out on the Triolith cluster, which is an Intel Sandybridge cluster with an Infiniband network interconnect. Timing results for our modified version 1 routine are shown in Table 2.

MPI processes	Total run time (s)		`cs_halo_sync_var` time (s)	
	original	version 1	original	version 1
64	264.41	264.12	6.6	6.1
128	152.49	152.11	7.3	6.8
256	105.89	105.96	9.2	8.5
512	91.04	91.03	11.3	10.2

Table 2: Scaling results for 'One Tube' test case runs comparing the original two-sided communications vs. the new one-sided communication implementation (version 1)

The modified version 1 of `cs_halo_sync_var()` shows some performance improvement with respect to the original version. The modified cs_halo_sync_var() subroutine is around 10% faster than the original routine for 512 core runs. However weightage of this routine in the overall runtime is around 10 – 12 %. Hence we do not see a proportional reduction in runtime. The version 1 subroutine also has 1-2% less memory overhead compared to the original version. The effects of both timing improvement and less memory consumption are expected to be more prominent when running the code on much larger core/node counts on the road to exascale.

The modified version 2 of `cs_halo_sync_var()` routine uses the MPI 3.0 one-sided communication routines from the MVAPICH2 MPI implementation. We were not able to run this version as the run crashes with MPI 3.0 routines, indicating that the MPI 3.0 implementation employed is still not very robust. However, we conclude from this project, that it seems adopting MPI one-sided communication in Code_Saturne can improve its scalability further on the road to exascale when optimized and a stable implementation of MPI 3.0 becomes available.

2.3 Evaluation of the Effectiveness of OpenACC for enabling DL_POLY_4 on the Road to Exascale

WP156: *An Analysis of State-of-the-art Tools for Preparing DL_POLY_4 for Exascale*

Authors: B B Gursoy (ICHEC) and H Nagel (NTNU)

Application: DL_POLY_4

HPC Tools/Techniques: OpenACC, CUDA

Person Months: 2

DL_POLY is a well-known parallel large-scale molecular dynamics simulations package developed by I.T. Todorov and W. Smith at the STFC Daresbury Laboratory [11]. The DL_POLY_3 package was ported to GPUs, using the CUDA framework by the Irish Centre for High-End Computing (ICHEC) in collaboration with Daresbury Laboratory [12]. Recently, DL_POLY v4.05.1 has been released with a number of modifications in the source code. Although CUDA provides a significant performance improvement, it is hard to maintain the code in order to keep up with the changes of the vanilla MPI code. This has inspired us to consider OpenACC as an alternative tool that is more easily maintainable on the road to exascale compared to the CUDA framework.

OpenACC is a high-level directive-based programming tool for heterogeneous systems [13]. Similar to the execution model of CUDA, it is designed for accelerating compute-intensive loops and regions of C/C++/Fortran code by offloading to the GPUs. As highlighted in the Deliverable D7.2.1 ‘A report on the Survey of HPC Tools and Techniques’, OpenACC has been receiving increasing attention for next generation HPC systems [1]. Inspired by the existing CUDA port, this project is an early evaluation of the effectiveness of OpenACC for enabling DL_POLY_4 on accelerator-based platforms. In particular, it is concerned with investigating the benefits of OpenACC in terms of maintainability, programmability and portability issues that are becoming increasingly challenging as we advance to the exascale era.

All tests were conducted on the Abel supercomputer located at the University of Oslo, Norway. The Abel computing cluster consists of more than 650 Supermicro X9DRT compute nodes each having two Intel E5-2670 Sandy Bridge 2.6 GHz CPUs and 64 GBs of Samsung DDR3 memory. A set of nodes is equipped with NVIDIA K20 GPUs. TEST2 and TEST3 test cases were used from the data sets that come with DL_POLY. The serial version of DL_POLY_4.05.1 was first profiled using the default PGI profiler pgprof to investigate the most time consuming routines. To further detect hotspots in these routines, Allinea MAP v4.2 profiler was used. During profiling, compute intensive loops in `ewald_real_forces()` and `ewald_spme_forces()` have been targeted during this project.

For the `ewald_real_forces()` subroutine, major code refactoring was required to enable OpenACC. We illustrate an incremental approach to increase the performance of the OpenACC port. There was an approximate 13.6x speedup in the execution of the main loop after all the code refactoring relative to the first considered approach. However, the OpenACC

port did not perform better than the serial execution time. For the `spme_forces()` subroutine, adding a set of compiler directives with 'CUDA-inspired' scheduling parameters demonstrated a significant performance increase in the execution of the main loop. In particular, a speedup of ~8.8x was achieved relative to the serial execution time when the 'CUDA-inspired parameters' were specified as gang and vector sizes, representing an ~83% improvement in performance relative to the naive compiler generated version. However, the performance proved not to be as good as the CUDA version. We present our timing results in Table 3. We set the environment variable `PGI_ACC_TIME` to 1 to measure the execution time on the GPU. During this project we have found that one of the limiting factors of OpenACC is that the standard doesn't allow the developer to explicitly utilize the fast GPU shared memory as was exploited in the CUDA port.

Implementation	Runtime
Serial, original code	60.2 sec
OpenACC(default)	39.321 sec
OpenACC(gang(900) vector(8x8))	6.866 sec
CUDA port	2.4271 sec

Table 3: Wall-clock time of the main loop in spme_forces() for different implementations

In Figure 2 we illustrate that the OpenACC port of the main loop in `spme_forces()` scales well when the dimension of the reciprocal lattice vector used within the `spme_forces()` subroutine increases. However, data transfer overhead between the CPU and GPU memory dominates the overall execution time heavily for bigger dimensions. For example, when the reciprocal lattice vector dimension was 106, 26% of the runtime was spent transferring data between the CPU and GPU. With a dimension of 212, data transfers represented 63% of the overall execution time.

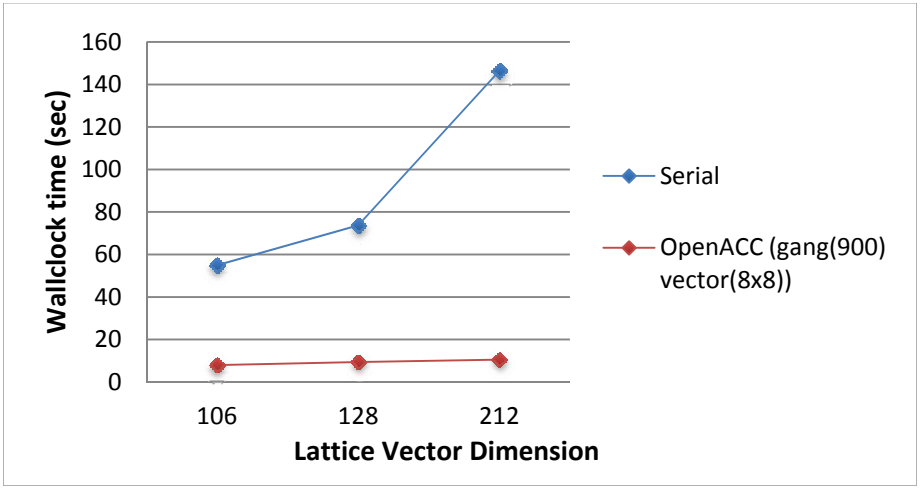

Figure 2: Runtimes of the main loop in `spme_forces()` for different lattice vector dimensions

During this project we have also considered an OpenACC port of the `link_cell_pairs_remove_exclusions()` component along with integrating it with the rest of the CUDA kernels to test the effectiveness of running OpenACC in an interoperable way with CUDA. In this context, the role of the `deviceptr` clause was highlighted in terms of accessing the data that has been already allocated in the GPU memory in order to avoid unnecessary data copies between the CPU and GPU memory. Performance tests showed that

there is an approximate of 2x speedup of the overall execution time due to minimizing data transfers by use of `deviceptr`. Finally, while performance of the OpenACC port was not found to be as high as for the original CUDA implementation, in terms of programmability and maintainability, it is worth pointing out that only four OpenACC directives were added to the original `spme_forces()` subroutine for the OpenACC acceleration while the corresponding CUDA port consists of approximately 500 lines of CUDA code.

2.4 Enabling CP2K for Exascale with OpenACC/OpenCL

WP155: *Enabling the CP2K Application for Exascale Computing with Accelerators using OpenACC and OpenCL*

Authors: A. Kwiecień (WCSS), M. Uchroński (WCSS) and M. Gębarowski (WCSS)

Application: CP2K

HPC Tool/Technique: OpenACC/OpenCL

Person Months: 1.1

CP2K [4] is an open-source application designed for atomistic and molecular simulation of solid state, liquid, molecular and biological systems. CP2K has proven to be a highly scalable code [14], which makes it a good candidate for exploitation on current petascale and future exascale systems. The code is written in Fortran 95, well parallelized with MPI and, in some parts, with hybrid MPI/OpenMP [15] and CUDA.

The main goal of this project was to identify routines in CP2K suitable for enablement on accelerators/coprocessors using OpenCL [16] and OpenACC [13]. OpenCL and OpenACC are both open standards and have been identified by PRACE as important for exascale computing [1] Based on previous research on CP2K [14][17] we have focused our effort on the DBCSR library, which performs sparse matrix multiplications. During this project, we worked on CP2K v2.4 (at the time of writing v2.5 is available for download).

With OpenACC, a developer can annotate C, C++ and Fortran source code to identify the areas to be accelerated using `#pragma` compiler directives and additional functions. The latest version of the standard, OpenACC 2.0a [18], was announced on 31st August, 2013. The OpenCL framework provides an API and a standard language, based on ISO C99 language, to write portable code for multi-core CPUs, GPUs, APUs and other architectures, including Intel Xeon Phi coprocessors. OpenCL kernels are compiled at runtime to target a particular computing device. The latest version of the standard, OpenCL v2.0 [19] was announced on 18th March, 2014. For the testing we used: Supernova (WCSS), and for some additional testing during development also Zeus (Cyfronet) and Fionn (ICHEC) clusters.

The OpenACC directives are supported by only a few commercial compilers. The PGI compiler [20] was our first choice as the license is available on Supernova and Zeus. A number of different issues were identified when building CP2K with the pgfortran compiler, including non-supported Fortran 2008 functions and a segmentation fault. We mainly used PGI version 13.5, but releases v14.1 and v14.3 were also tested, with no improvement in the problematic areas. For debugging purposes we used Allinea DDT [21] and PGDBG [22], but further analysis is required to solve the segmentation fault issue. Slow compilation with PGI has been a disadvantage, limiting to some extent possibilities of exploring different compilation options and target architectures in a given time frame. Due to the issues mentioned we needed to narrow our work to a part of the DBCSR library, to check if using OpenACC in the sparse matrix block multiplication algorithm would give any performance

improvement. For the testing of both OpenCL and OpenACC ports we used an example `dbcsr_example_3.F` delivered with the application source code.

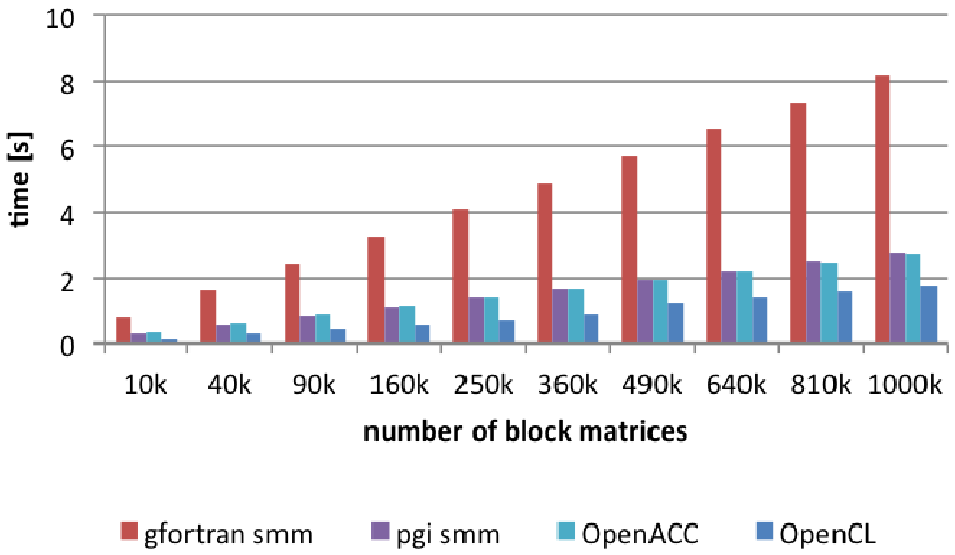

Figure 3: CP2K test results for OpenCL and OpenACC (mm_stack_size = 10000)

We tested several versions of the code including: a serial versions built with GNU `gfortran` and PGI `pgf90` compilers, both linked to an external specialized library for small matrix multiplication; a version with OpenACC enablement; and a version with OpenCL enablement. The results, shown in Figure 3 are obtained for sparse matrices with different numbers of small block matrices (from 10k to 1000k blocks, with the constant block size set to 100x100, giving 10k elements in every block, and `mm_stack_size=10000`). Presented times are mean values gathered from 10 calls to the multiplication function `dbcsr_multiply`. Tests presented were performed on a Supernova node with an NVIDIA GTX 480 GPU. We increased the problem size by changing the dimension of the sparse matrix – adding 100 blocks to each dimension in one step.

If the size of the block matrices was too small (set to 10x10 elements) the OpenCL port performed worse for bigger problems than OpenACC and PGI with SMM. Increasing the size of block matrices to 100x100 elements has given a performance improvement. Also, too small a value of the `mm_stack_size` parameter (e.g. 1000) results in performance issues for accelerator ports (It results in dividing the data into smaller portions and causes many sequential executions of the kernels on the GPU device). In the OpenCL port the computational data is copied to/from GPU before and after each OpenCL kernel execution. The data transfer between CPU and GPU is a well-known performance bottleneck, and setting the parameter to a bigger value (e.g. 10000) resulted in a performance improvement. Changing the parameter for the OpenACC port has no impact on the execution time, because of the placement of the pragmas. We determined that we cannot gain a better performance without significantly refactoring the code. It could be realized by copying the data to the GPU before the multiplication, rather than right at the start of the OpenACC kernel. We have started to investigate this approach, and will continue if time allows.

In conclusion, both OpenACC and OpenCL have shown potential for improving CP2K performance. With OpenCL we have been able to obtain very good results. OpenACC was quite straightforward to introduce but in the tested case, in order to get the most of the GPUs some further modifications of the library would be needed. Nevertheless, we have shown that

even minor code modifications may improve the results and we believe that with proper adjustments even better performance can be achived. It must be noticed that the power of the OpenACC standard strongly depends on the compilers' support. The choice of proprietary compilers seems to be insufficient, especially in a current landscape of scientific applications which often are open-source and developed using open tools and compilers like GCC. This may lead to compatibility issues, as described here for CP2K. Another direction worth considering is investigating the CAPS source-to-source compilers for OpenACC together with GNU or Intel compilers. Combining CAPS and Intel compilers would give another possibility to target the Intel Xeon Phi as one of the accelerators, in addition to the work done with CP2K on this architecture so far [23]. OpenACC and OpenCL once introduced to the code, in connection with the wide compiler support for different targets, are powerful technologies. As an additional conclusion we may state that introducing OpenACC to an existing application is relatively simpler and requires less knowledge and time from the developer than OpenCL. However, it still requires a good understanding of the application, its data and algorithms, and may require refactoring of the original code to gain a performance as expected from the GPU acceleration.

2.5 Enabling the Cellular Automata Library for Exascale with OpenACC

WP154: *Multi-GPGPU Cellular Automata Simulations using OpenACC*

Authors: S Szkoda (WCSS), Z Koza (IFT), M Tykierko (WCSS)

Application: Cellular Automata Library

HPC Tool/Technique: OpenACC, CUDA, GPU Direct

Person Months: 0.5

The Frisch-Hasslacher-Pomeau (FHP) model [24] is a lattice gas cellular automaton designed to simulate fluid flows using the exact, purely Boolean arithmetic, without any round-off error. Here we investigate the problem of its efficient porting to clusters of Fermi-class graphic processing units. To this end two multi-GPU implementations were developed and examined as part of this project: one using NVIDIA's CUDA programming framework as well as GPU Direct technologies and the other one using the OpenACC compiler directives [26] available via PGI with the MPICH2 MPI implementation for MPI communications. For a single Tesla C2090 GPU device both implementations yield up to a 7-fold acceleration over an algorithmically comparable, highly optimized multi-threaded implementation running on a server-class CPU.

We have demonstrated that the weak scaling for the explicit multi-GPU CUDA implementation is almost linear for up to 8 devices (the maximum number of the devices used in the tests), which suggests that the FHP model can be successfully run on much larger clusters and is a prospective candidate for exascale computational fluid dynamics. To enable multi-node calculations with OpenACC an implementation with MPI is currently being developed. The scaling for the OpenACC approach turns out less favorable than the CUDA code due to compiler-related technical issues. We found that the multi-GPU approach can result in considerable benefits for this class of problem, and that GPU programming can be significantly simplified through the use of the OpenACC standard, without a significant loss of performance, providing that the compilers supporting OpenACC improve their handling of the communication between GPUs.

Implementation	GTX480 [GUPS]	M2090 [GUPS]	K20M [GUPS]	8x M2090 [GUPS]
OpenACC	36.8	30.1	N/A	188.4
CUDA	41.1	32.3	57.0	217.1

Table 4: The performance of our implementation of the Cellular Automata Library (FHP) on various CUDA-capable devices in GUPS (lattice node updates per second). Higher is better.

The results for the weak scaling for up to 8 GPUs are shown in Figure 4. The choice between

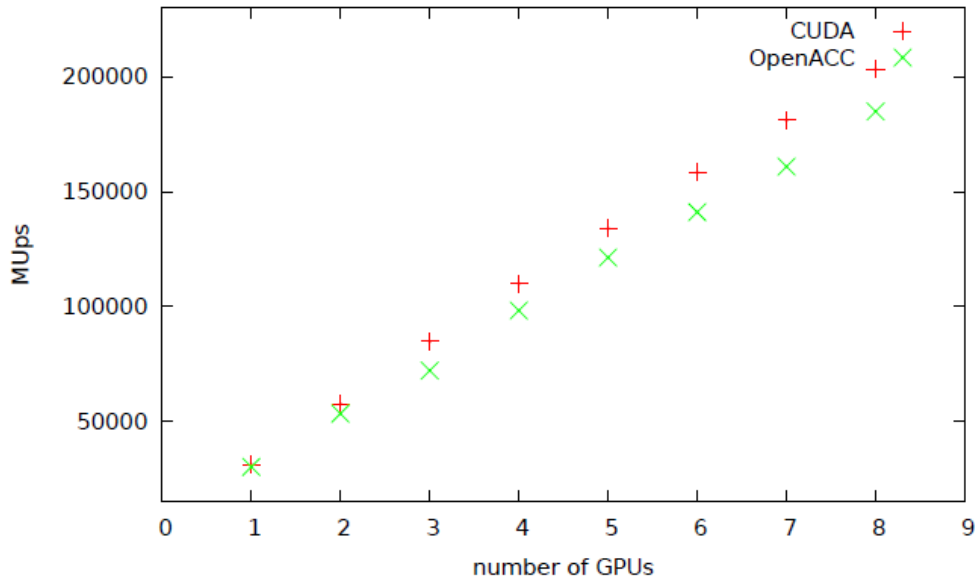

Figure 4: Weak scaling of FHP on a node with 8 NVIDIA M2090 GPUs

technologies depends on both the expected performance and the programming costs. On the one hand NVIDIA CUDA gives us finer control over the parallelization of the application and, assuming a programmer's deep understanding of the underlying accelerator architecture, is expected to outperform any directive-based tools. On the other hand, the results show that by using a small subset of compiler pragmas we can port an application to GPUs in a much shorter period (on the order of a few minutes), without a significant performance loss.

The Portland Group OpenACC implementation is still under development. As soon as it can support a CUDA-aware MPI implementation, the development of efficient Multi-GPU applications will become much easier and it is likely that manual CUDA programming will be used only in the most crucial parts of the code.

While fully recognising that results presented here account for a single node only, the fact that the application scales up within a node so well, shows the potential for effective exploitation of large scale systems when scaling out with MPI, which we hope to report on in the very near future.

2.6 Preparing Coupled Climate Models for Exascale: OpenACC-enabled EC Earth3 Earth System Model

WP166: *Scaling Coupled Climate Models to Exascale: OpenACC-enabled EC-Earth3 Earth System Model*

Authors: P Nolan (ICHEC) and A McKinstry (ICHEC)

Application: EC-Earth3

HPC Tool/Technique: OpenACC

Person Months: 1.5

This project investigates methods to enhance the parallel capabilities of the EC-Earth3 Earth System Model [28] by offloading bottleneck routines to GPUs and Intel Xeon Phi coprocessors. EC-Earth3 component models are IFS [29] for the atmosphere, NEMO [30] for the ocean, and LIM [32] for the sea-ice, coupled through OASIS3-MCT [33]. The sea-ice model, LIM, is a component of NEMO. To gain a full understanding of climate change at a regional scale will require EC-Earth3 to be run at a much higher spatial resolution (T3999 ~5km) than is currently feasible. Although long multi-ensemble climate simulations at this resolution are currently not possible, it is envisaged that the work outlined in this project will provide climate scientists with valuable data for simulations planned for future exascale systems. The goals of this project are to:

1. Highlight bottlenecks of the EC-Earth3 earth system model.
2. Port EC-Earth3 to new hardware "accelerators" such as general-purpose Graphics Processing Units (GPUs) and the Intel Xeon Phi coprocessor.
3. Enhance the parallel capabilities of EC-Earth3 by offloading the corresponding bottleneck routines to GPUs and Intel Xeon Phi coprocessors.
4. Assist in addressing the challenges of porting EC-Earth3 to new generation of HPC systems, which will provide multi-Petaflop performances in the next few years and exaflop performances in 2020.

Goal 1 is fully complete; EC-Earth3 was scale-tested on the Hermit PRACE tier-0 system. Goals 2 and 3 are partially complete; OpenACC directives were successfully used to offload certain bottleneck routines of EC-Earth3 to GPUs and those sections did achieve some encouraging acceleration. A summary of preliminary results is presented here.

Scaling experiments of EC-Earth3 were carried out on Hermit with NEMO ORCA025L46 configuration and IFS resolutions of T799, T1279 and T2047. The simulations ran using 11500 cores and scaled well to ~8000 cores. EC-Earth3 T799 and T1279 resolution runs were profiled, using CrayPat, to highlight bottlenecks at low, medium and high core counts. Results for resolution T799, using 1536 MPI processes, are presented in Figure 5.

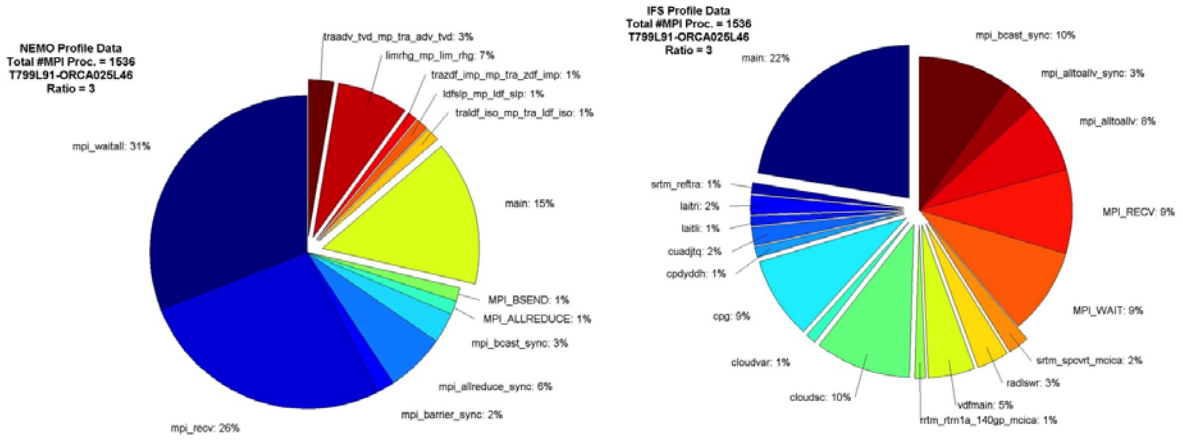

Figure 5: Profiling of EC-Earth3 (T799L91-ORCA025L46). The simulation was run using 1536 CPU cores using a 'total processes per NEMO process ratio' of 3. Routines are presented only if they account for greater than 1% of total run time. Left: NEMO, Right: IFS

OpenACC directives were used to offload bottleneck routines of EC-Earth3 to GPU accelerators. Future work will focus on translating the code to CUDA. EC-Earth3 was compiled and run on the ICHEC system, Stoney, using the PGI compilers v12.8. Each compute node has two 2.8GHz Intel (Nehalem EP) Xeon X5560 quad-core processors. Each node has two NVIDIA Tesla M2090 cards installed, with each card providing 512 GPU cores, 6GB of local GDDR5 memory and a theoretical peak double-precision performance of 665Gflops. The following flags were used to compile EC-Earth3:

```
mpif90 –O1 -acclibs -ta=nvidia,cuda4.2,fastmath,time -Mipa=inline
```

Once EC-Earth3 was compiled, it was relatively straightforward to insert OpenACC directives and to accelerate the bottleneck routines identified.

NEMO was run for one month using the 'ORCA1L46' configuration, which has a horizontal resolution of approximately 1 degree and 46 ocean levels. Simple OpenACC directives were added to the "eosbn2.F90" and "trazdf_imp.F90" routines. These routines were responsible for ~0.6% and 1% respectively of total execution time, when run within EC-Earth3 (T799L91-ORCA025L46) using 1536 MPI processes. These percentages increase when using less MPI processes and when running NEMO in standalone mode. By porting the two aforementioned subroutines to a single M2090 GPU we have achieved a 4% speedup of the overall NEMO code running on a single Nehalem CPU core. Typically, climate simulations are run for ~100 years with an ensemble size of over twenty, so the performance improvement here, although modest, corresponds to a potential acceleration of over 200 days for a multi-decadal ensemble of EC-Earth3 climate simulations.

OpenACC directives were added to the main loop in laitri.F90, and also to some loops in rtm_rtrn1a_140gp.F90. Table 5 shows performance of the routines with very small 50 x 50 x 60 point problem size on a single CPU-core and a single GPU. The GPU achieved a speedup factor of about 3 in both cases. It should be noted that this simulation domain is quite small and work remains to be done to achieve acceleration on higher resolution domains.

#	% Time (self)	Cumulated (sec)	Self (sec)	Self (1-CPU, CPU+GPU)	Total	Routine
1	15.24	108.87	108.87	(33.5,**10.1**)	108.89	LAITRI
2	3.80	206.77	27.18	(12.2,**3.7**)	27.18	RRTM_RTRN1A_140GP

Table 5: Performance of two IFS routines on CPU and GPU. In each case, we compare runtimes using a single CPU (Xeon X5560) core with no GPUs attached to the same CPU chipset with a GPU (NVIDIA Tesla M2090) attached

The results presented show acceleration for two NEMO and two IFS routines. However, a large number of EC-Earth3 routines were unsuccessfully 'accelerated'. This was largely due to the fact that data movement between the host and device over the PCIe bus is relatively slow. We are currently working to minimize this data movement by offloading data only when completely necessary.

2.7 A Hybrid Application of OmpSs

WP150: *A Hybrid use of OmpSs for a shock hydrodynamics proxy application*

Authors: J C Meyer (NTNU)

Application: LULESH

HPC Tool/Technique: OmpSs

Person Months: 1

The LULESH [34] proxy application models the behaviour of the ALE3D [35] multi-physics code with an explicit shock hydrodynamics problem, and is developed in order to evaluate interactions between programming models and architectures, using a representative code significantly less complex than the application it models [36]. As identified in the deliverable D7.2.1 [1], the OmpSs programming model [37] specifically targets programming at the exascale, and this project investigates the effectiveness of its support for development on hybrid architectures.

Simulation of hydrodynamics is of importance to a wide variety of engineering problems. For the sake of simplicity, the proxy application models only the time development of a single, specific blast wave, but it presents computing systems with the data movement and computational characteristics of more general applications, and has been widely used in studies of program and architecture co-design efforts to address exascale challenges.

Taking cognisance of the fact that the main programming environment challenges on the road to exascale are expected to be within nodes rather than across nodes, our focus here is on single node performance, which is where the hardware is most rapidly evolving due to exponentially expanding parallelism, decreasing relative memory bandwidth and less memory per thread.

A defining characteristic of OmpSs is that it internally implements parallelism as a dependency graph of separate computational tasks, either to be explicitly specified by the programmer, or implicitly derived from work-sharing directives. We investigate the performance implications of explicit and implicit task specifications, and find that the implicit approach gives better speed and scalability within a 16-core SMP system for parallelism at the granularity found in LULESH. Parallel speedup curves using implicit task generation for problem sizes from 5^3 through 90^3 element domain sizes are shown in Figure 6.

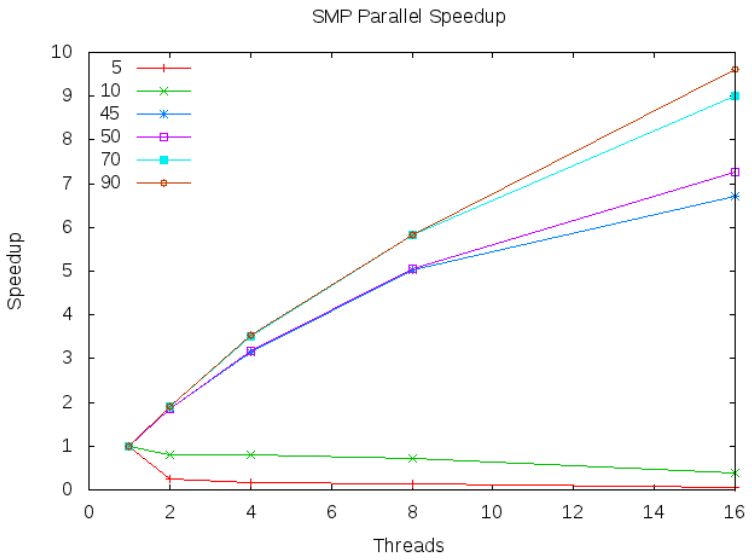

Figure 6: LULESH Speedup on a 16-core SMP system

Further to this, we investigated explicit tasks as a programming construct to offload computational work to graphics accelerators. Full empirical testing proves infeasible in this case, but a model of application behaviour combining observations from NVIDIA K20 accelerators with synthetic benchmarks of data movement requirements is presented. It suggests that the overheads of task migration is out of proportion to the computational intensity of tasks in this application, indicating that a straightforward task-based hybridization may give little benefit, and that a greater restructuring will be necessary if a combined SMP/accelerator implementation is to provide competitive performance using this programming model.

2.8 Application of Accelerator Units to Neural Networks

WP164: *Computational Throughput of Accelerator Units with Application to Neural Networks*

Application: Neural Networks application (ANNIWHD)

HPC Tool/Technique: OpenMP

Authors: J C Meyer (NTNU) and B A Dunn (NTNU)

Person Months: 0.5

This project describes an effort to evaluate a proof-of-concept adaptation of a sample neuroscience application program to utilise accelerator technologies. Accelerator units represent an important architectural development in the progress towards exascale computations, and a combination of an OpenMP programming model with Intel Xeon Phi accelerator units is investigated, both of which were identified as key enabling tools in the deliverable D7.2.1 [1].

A challenging problem in neural computational research is that of estimating structures of neural networks which are hidden from the limited experimental data that can be acquired from *e.g.* neural recordings. The sample application implements an approximate expectation-maximization method to infer the network structure and time varying states of a hidden population [38]. The size of networks that can yield informative results can be made

arbitrarily large, and the long-running computational demand is highly localised, making the application a candidate for exascale computations.

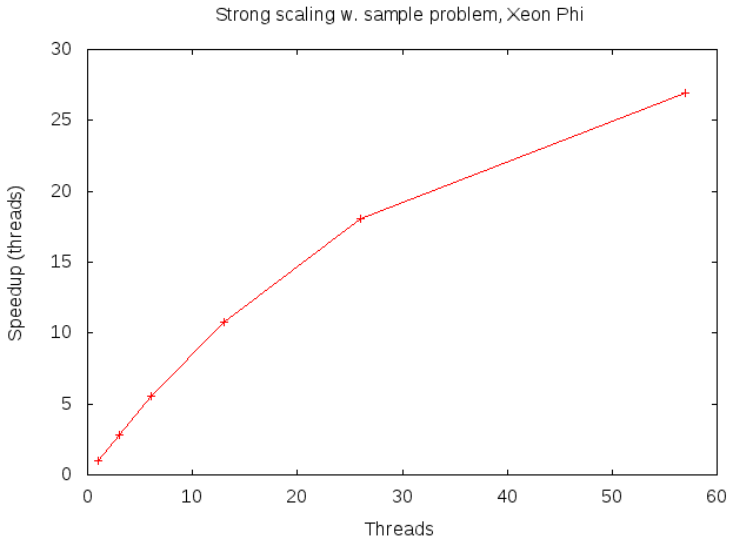

Figure 7: Adapted program speedup relative to single-threaded execution in native mode on Xeon Phi

The sample application's parallelization relies on calling parallelized linear algebra routines from an otherwise sequential implementation, giving insufficient parallel work to effectively improve single-threaded run times. The whitepaper describes a proof-of-concept adaptation to OpenMP, and includes performance figures that show favourable strong scaling results with the relatively small input case already obtaining parallel speedups up to the full utilization of a 57-core Xeon Phi unit (3120A), shown in Figure 7.

3 Scalable Libraries and Algorithms

As a consequence of the move towards large multi-petascale heterogeneous systems, there is an increasing demand for new and improved scalable, efficient, and reliable numerical algorithms and libraries that confront existing and upcoming complexities associated with such systems, including complex memory hierarchies, the overhead of data movement and fault tolerance.

During phase 1 of T7.2 nine areas of focus for tool assessment were identified and reported on in D7.2.1. The list was roughly divided into scalable numerical algorithms/methods (Direct Solvers, Iterative Solvers and FFT Libraries), higher-level libraries and other mesh/graph partitioning tools. The list by no means represented an exhaustive survey of scalable libraries and algorithms, but rather tried to seek a balance between assessing what had been investigated in PRACE to date and what had, at the time, been investigated elsewhere (specifically within exascale projects) with the same set of tools. In carrying out our assessments we drew valuable information from PRACE-2IP WP12 in particular, and from each of the European exascale projects, including the CRESTA project which has a significant co-design focus.

In this section, we report on seven projects that have each focused on exploiting state-of-the-art libraries and algorithms in order to enable applications for multi-petaflop/future exascale systems. Each subsection provides a summary of the project along with a reference to the PRACE-3IP whitepaper associated with the project [2]. We recommend that the reader also

refers to the whitepaper for each project, which provides a more detailed report on the projects than is provided here.

The libraries and algorithms that we have focused our attention on during the exploitation phase along with the applications that were enabled are listed in Table 6

HPC Tool/Technique	*Application*
Poisson Solver	DL_POLY_4
Mesh Generation	OpenFOAM
Mesh Refinement	Code_Saturne
Algebraic Multi-Grid (AMG) Solver	tmLQCD
CFD/CSM Solvers	OpenFOAM, FFLOP, PETSc
MapReduce Algorithm	Sparse Matrix-Vector Kernels

Table 6: HPC Tools and Techniques (Scalable Libraries and Algorithms) exploited along with corresponding applications

In terms of challenges on the road to exascale, global communication, in particular, is known to be a severe barrier when trying to scale across large core counts and many open questions still exist on how FFT libraries will perform on future exascale systems (see reports on FFT libraries in section 4.3 of D7.2.1). With this challenge in mind, we are happy to report on the successful implementation of alternative methods to FFT libraries in DL_POLY_4, which has the potential to significantly improve scalability (and functionality) of the code on large node counts for certain problem types, allowing for scientific investigations using DL_POLY_4 that were heretofore not possible.

As well as alternatives to FFT methods, mesh generation and refinement tools were also identified in D7.2.1 as being of crucial importance to several communities on the road to exascale, including the CFD and CSM domains, where such methods often become severe bottlenecks when scaling to large node counts in multi-physics problems in particular. Indeed, geometry and mesh generation related concerns have also been highlighted in a recent 'Applied Mathematics for Exascale Report' [39] by the DOE's Exascale Mathematics Group. In this section we report on how mesh generation and mesh refinement tools have been exploited to enable Code_Saturne and OpenFOAM, respectively. We also report here on how a novel algorithm based on the Algebraic Multi-Grid (AMG) method was implemented within a QCD application, allowing for much faster convergence than the original algorithm employed. In addition, the algorithm uses a new pre-conditioning method that significantly reduces MPI communications, further indicating potential for enablement on future exascale machines.

An in-depth analysis of open source solvers for CFD and CSM is also summarised here, where tools such as FFLOP and PETSc along with the wider challenges these tools face in solving complex multiscale multiphysics problems on the road to exascale is discussed in detail.

Finally, there are also many issues regarding the reduced reliability of hardware, expected to have an important impact on libraries and algorithms on the road to exascale. This is certainly the case for fault tolerance and resilience. However, these issues are still very new to the PRACE community and in general we have not found many examples of initiatives within PRACE in this area to date, an issue that we feel needs to be rectified quickly if we are to prepare applications for future multi-petascale and exascale machines. This will be a challenging task and one that may require inspiration from areas outside of traditional HPC.

One project reported on here that we feel gives a flavour for the type of non-traditional methods that should be explored more aggressively within PRACE is 'Parallelization of Sparse Matrix Kernels for Large-Scale Scientific Applications using the MapReduce Paradigm'. Besides the proven applicability of the programming model to areas such as graph algorithms and linear algebra operations, it is worth noting that the underlying programming model of the MR-MPI tool, which is exploited here, was first developed at Google with fault tolerance in mind. While the current 'Big Data' era is helping to increase awareness of HPC in the commercial ICT sector, we feel that PRACE should likewise be inspired by the fast pace of advances being made outside of traditional HPC.

3.1 Enabling the Generation of Massive Unstructured Meshes for OpenFOAM using Netgen

WP160: *Generating Massive Unstructured Meshes for OpenFOAM*

Authors: C Ozturan (BOGAZICI) and S Soner (BOGAZICI)

Application: OpenFOAM

HPC Tool/Technique: Netgen, PMSH

Person Months: 4.3

This project has focused on the development of an enablement tool called PMSH for facilitating fast generation of unstructured multi-billion-element tetrahedral meshes (grids) on complex geometries for the open source OpenFOAM computational fluid dynamics (CFD) package [40]. PMSH was developed as a C++ wrapper code around the open source sequential Netgen mesh generator [41]. It is available at https://code.google.com/p/pmsh/. Both OpenFOAM as well the Netgen mesh generator have a wide user base from many areas of engineering and science. OpenFOAM is a popular application that is used by a number of researchers on PRACE Tier-0 and Tier-1 systems. The Netgen mesh generator has also been listed as an important tool for HPC applications in the Deliverable D.7.2.1 'Survey of HPC Tools and Techniques' [1].

OpenFOAM provides a mesh generator called `blockMesh` for simple geometries. The `blockMesh` utility is a multi-block mesh generator that generates hexahedral meshes from a text configuration file. For complex geometries, OpenFOAM also provides a mesh generation utility called `snappyHexMesh`, which generates hexahedral meshes. The `snappyHexMesh` utility works more like a mesh sculptor rather than a generator. It takes an existing mesh such as the one produced by `blockMesh` and chisels out a mesh on a complex geometry that is given in STL format. The `snappyHexMesh` utility has advanced features like the ability to run in parallel and being able to redistribute the mesh so as to perform automatic load balancing. Both utilities, `snappyHexMesh` and `blockMesh`, are not as advanced as other commercial or open source mesh generator packages for producing quality tetrahedral meshes on complex geometries. Since one needs to generate a massive mesh before being able to proceed to the solver phase, the mesh generation phase often forms a bottleneck that will hinder the use of future exascale solvers on complex geometry problems. Therefore, there is a great need in the OpenFOAM community for tools that will enable researchers to generate massive meshes on complex geometries.

In PMSH, parallelization of the mesh generation process is carried out in five main stages: (i) generation of a coarse volume mesh (ii) partitioning of the coarse mesh to get sub-meshes, each of which is processed by a processor (iii) extraction and refinement of coarse surface sub-meshes to produce fine surface sub-meshes (iv) re-meshing of each fine surface sub-mesh to get the final fine volume mesh (v) matching of partition boundary vertices followed by

global vertex numbering. An integer based barycentric coordinate method is developed for matching distributed partition boundary vertices. This method does not have precision related problems of floating point coordinate based vertex matching.

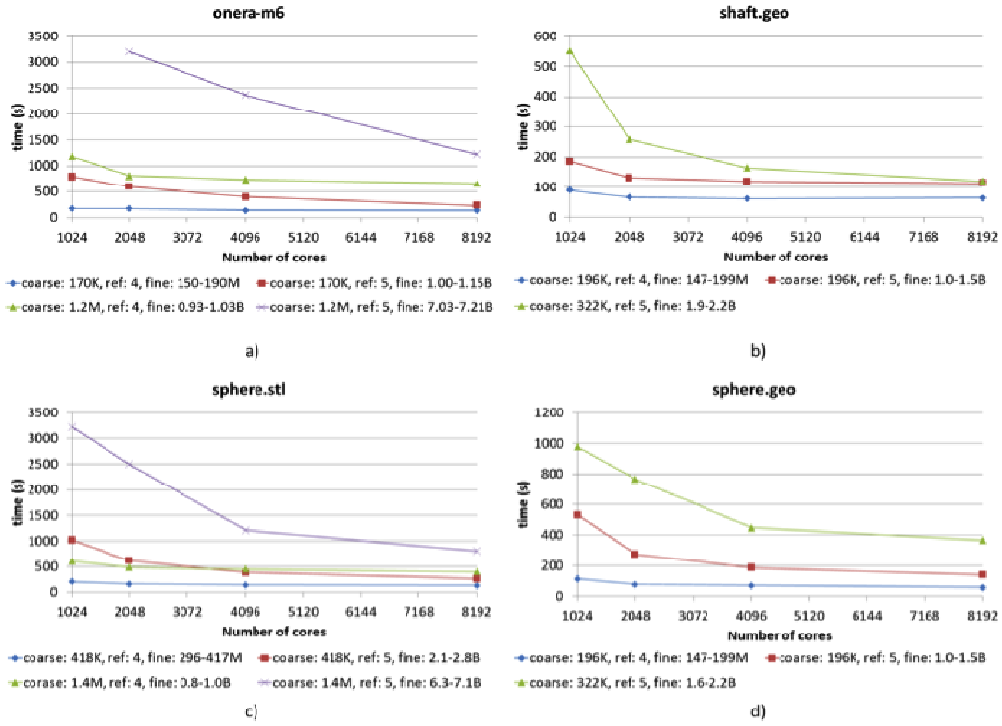

Figure 8: Parallel mesh generation timings for (a) Onera-M6.stl, (b) shaft.geo, (c) sphere.stl and (d) sphere.geo geometries obtained for various ranges of coarse and fine mesh sizes

Tests were performed on NTNU's Vilje system. Vilje is an SGI Altix ICE X system that has 936 nodes available to academic users. Each node has 2 Intel Xeon E5-2670 (Sandy Bridge) processors with a total of 16 cores and 28 GB of memory. Due to 14 TB total memory allocation limitation of submitted jobs, tests using a maximum of 8192 cores were performed. Figure 8 (a), (b), (c) and (d) show timing results of generating meshes for onera-m6,shaft, sphere (stl) and sphere (geo) CAD geometries respectively. As shown in the plots in Figure 8, PMSH enabled us to generate multi-billion element meshes in a scalable way, indicating the potential for our tool within application such as OpenFOAM on mutli-petaflop/future exaflop systems. Scalability is found to be better when the initial coarse mesh is smaller and the surface refinement level is higher. This is because the initial coarse mesh generation is serial and mesh generation from finely refined surface mesh is parallel.

3.2 Enhancing Code_Saturne Capability in the area of Parallel Local Mesh Refinement

WP158: *Parallel Local Mesh Refinement for Code_Saturne*

Authors: A Ronovsky (VSB), T Karasek (VSB) and Vit Vondrak (VSB)

Application: Code_Saturne

HPC Tool/Technique: Parallel Adaptive Mesh Multiplication, ParMetis, PT-Scotch

Person Months: 2.25

In this project, local refinement methodology and its implementation in Code_Saturne [8] are investigated. The local refinement technique can be exploited to create very large meshes with up to 1 billion cells, a capability which is of interest to a wide community on the road to exascale, including within the CFD and CSM communities. The advantage of local refinement is that only regions of interest are refined (e.g., in CFD simulations those regions could be either areas close to boundaries, small geometrical entities or regions with a high gradient of solved quantities). This approach can create meshes, which are not only very large but also provide high accuracy results. The major challenge facing local adaptive refinement is that it breaks load balancing of the original mesh and requires a large amount of global communications.

The numerical experiments of a full hybrid refinement of a nuclear reactor cooler segment are presented. These experiments were carried out on HECToR (CRAY XE6), Blue Joule and Mira (where the latter two machines are IBM Blue Gene/Q machines). Results of an implementation of a multilevel mesh multiplication (global refinement) algorithm for multi-billion cell mesh simulations are presented in our whitepaper. The mesh multiplication algorithm starts from a coarse mesh partitioning using ParMetis or PT-Scotch, two partitioning tools that were identified as having potential on the road to exascale in deliverable D7.2.1, and subdivides each of its cells into a given numbers of subcells, subdivides the boundaries and computes the global indices of the new elements. There is no limitation on the number of refinements that can be computed by the algorithm, aside from the memory available on the machine. Up to four levels of refinement are used in the tests in this project. Meshes containing ~6.6 billion, 13 billion and even 105 billion were generated on the fly using this technique.

The main focus of this project was to develop a strategy for an adaptive mesh refinement technique and implement it in Code_Saturne. An extended data structure necessary for this implementation is presented in detail in our whitepaper, together with a description of the adaptive mesh refinement algorithm.

Performance and scalability tests of the ‘Parallel Adaptive Mesh Multiplication’ (PAMM) algorithm [42] have been carried out on ANSELM. The test case simulates the laminar flow in a lid-driven cavity (cubic box with a horizontal constant velocity imposed at the top wall). The original mesh was created from an unstructured tetrahedral mesh using a full mesh multiplication algorithm.

Two levels of uniform refinement were performed to obtain two meshes, which were subsequently further refined using the local mesh refinement algorithm. After one level of uniform refinement the mesh has 14 million cells (14M), after two levels it consist of 111 million cells (111M). A 14M cell mesh was than locally refined to 16M cells and a 111M mesh to 126M. A non-weighted partitioning was performed by a Morton space-filling curve algorithm. The results of our numerical experiments can be seen in Table 7, Table 8 and Table 9.

No. MPI tasks	Partitioning (s)	Refinement (s)	Avg. time step (s)	Total time (s)
128	45	30	105	1513
256	63	41	85	1231
512	101	55	165	1852
1024	179	71	210	2543

Table 7: Adaptive refinement of a cubic cavity from 14 million cells to 16 million

No. MPI tasks	Partitioning (s)	Refinement (s)	Avg. time step (s)	Total time (s)
128	37	8	701	7386
256	48	9	405	4904
512	73	11	297	3125
1024	177	30	583	6520

Table 8: Full refinement of a cubic cavity from 14 million cells to 111 million

No. MPI tasks	Partitioning (s)	Refinement (s)	Avg. time step (s)	Total time (s)
128	71	66	712	7623
256	82	80	426	5120
512	116	97	321	3542
1024	186	127	605	7035

Table 9: Adaptive refinement of a cubic cavity from 111 million cells to 126 million

The work undertaken as part of this project has contributed to the Code_Saturne project by enhancing the capability of mesh multiplication in order to create full hybrid meshes of up to hundreds of billions of cells as well as adaptive tetrahedral meshes with up to hundreds of million of cells. To use numerical methods using meshes for discretization of computational domain (Finite Element or Finite Volume methods) on future exascale systems large meshes consisting of billions of cells will be necessary. Although the refinement process appears not to scale very well, as can be seen from tables 7 to 9, the refinement process takes only a negligible fraction of total time. With this in mind we conclude that mesh refinement, either full hybrid refinement or local mesh refinement, is a viable solution for the major challenge of generating and storing large meshes on the road to exascale computing, where, in particular, the method of redistributing a relatively coarse mesh to individual processors, where subsequently a much finer mesh is created by one of the refinement methods seems to hold promise in terms of efficiency. Where such refinement will become a genuine bottleneck when trying to scale on large-scale systems a closer look at load balancing will have to be taken.

3.3 Exploiting Open Source Codes for Solving Multi-scale Multi-physics Problems

WP157: *Application of CFD and CSM Open Source Codes for Solving Multiscale Multiphysics Problems*

Authors: D Horak (VSB), T Karasek (VSB) and Vit Vondrak (VSB)

Application: CFD, CSM multiphysics codes

HPC Tool/Technique: FFLOP/PETSc

Person Months: 3

In this project the use of open source Computational Fluid Dynamics (CFD) and Computational Structural Mechanics (CSM) for solving multi-scale multi-physics problems is addressed. Multi-scale and multi-physics problems are a set of problems, which were identified as facing major challenges on the road to exascale systems. In the whitepaper associated with this work an overview of the codes used within PRACE such as OpenFOAM [40], Code_Saturne [8], Code_Aster [43], Elmer [44] together with their limitations for

solving large problems is presented. The main focus of our project was on exploiting the library FLLOP [45] (FETI Light Layer On top of PETSc [46]) library, which is a novel software package developed at IT4Innovations, VSB-Technical University of Ostrava, Czech Republic for the solution of large-scale quadratic programming problems. It is an extension of PETSc, which was identified as a potential HPC tool for exascale systems in deliverable D7.2.1 [1].

During this project, two types of numerical experiments were investigated, where a Fluid Structure Interaction (FSI) simulation is used here to demonstrate the capability of the existing solvers mentioned above. All the experiments were run on HECToR. Since there is no suitable coupler between CFD and CSM codes available and since it was not the intention of this project to develop such a coupler, a simple one-way FSI simulation of a wind turbine was performed. OpenFOAM was used to perform CFD and FLLOP was used for the CSM calculations. Pressures on a blade calculated by OpenFOAM in one particular time-step were through I/O files passed to FLLOP as a loading for the CSM calculations.

Since our main interest is to identify problems and solution methods for future exascale systems we focused on the strong scalability of CFD and CSM solvers. The reason for this is that in real engineering applications the mesh size is usually fixed and the time-to-solution is the main concern. Another typical feature is that CSM meshes are much smaller than CFD meshes. In our case we modelled only one turbine blade for which approx. 10,000,000 of DOFS were needed to capture its physical behaviour. In contrast we needed more than 220,000,000 cells (1,300,000,000 unknowns) on the CFD side to obtain results with requested accuracy. Figure 9 shows the strong scalability of the CSM and CFD solvers for the wind turbine simulation.

From our numerical experiments, almost ideal scalability of the CFD solver up to 1024 cores could be observed. The CSM solver on the other hand shows good scalability only up to 16 cores. This poor scalability can be explained by fact that the CSM mesh is too small and more time is spent on communication instead of computation. Since this is not the first case where we have observed such behaviour we have decided to investigate whether this is really due to a problem size or whether it comes from the nature of the problem i.e., whether all CSM simulations regardless of the problem size would exhibit the same behaviour.

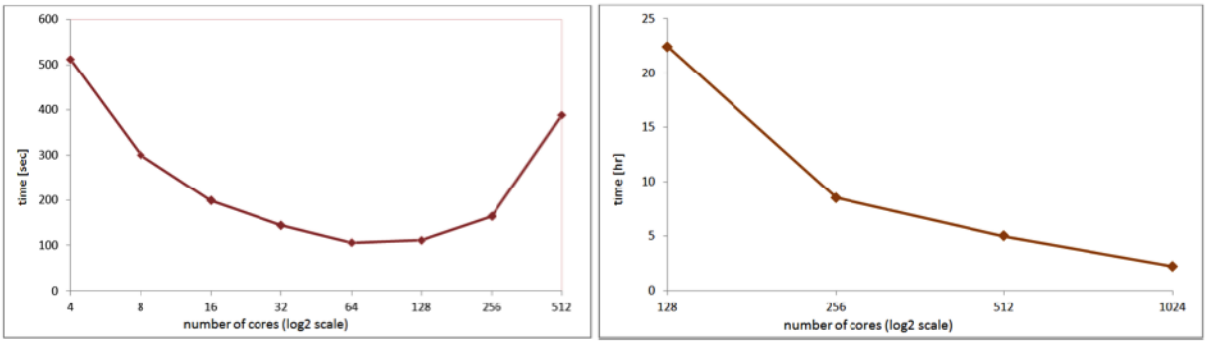

Figure 9: Strong scalability of the CSM (left) and CFD (right) solvers for a wind turbine simulation

To test the scalability of CSM solvers, a numerical model of an elastic cube was used. The weak scalability for 13,824, 8,000 and 4,096 elements per subdomain and the numerical scalability for these configurations are presented in our white paper. To investigate the strong scalability of CSM solvers we selected two different discretizations with 7,077,888 elements (approx. 22,000,000 unknowns) and 32,768,000 elements (approx. 100,000,000 unknowns), respectively. Presented results demonstrate the fact that significant scaling for tens/hundreds cores can be reached if the local problem sizes are sufficiently large.

As part of this project, we have identified two main issues with the solving of multiphysics problems on future exascale systems. The first issue is that there has to be a suitable solver for solving particular problem able to scale up to exascale range. In our work we demonstrated that FLLOP can be such a solver for solving CSM problems. The second issue is that there is an urgent need for a suitable coupler which will couple different solvers used for solving different physical phenomena e.g. FSI. It appears that, currently, no such suitable coupler is available and we pose the question here whether this should be the direction of future development on the road to exascale.

3.4 Enabling DL_POLY_4 for Scalable MD Simulations with Non-Periodic Boundary Conditions: Accounting for Electrostatic Interactions

WP161: *Development of a Grid-Based Electrostatics Model in a Memory Distributed Manner for DL_POLY 4*

Authors: P Petkov (NCSA), I Todorov (STFC), D Grancharov (NCSA), N Ilieva (NCSA), E Likova (NCSA), L Litov (NCSA), S Markov (NCSA)

Application: DL_POLY_4

HPC Tool/Technique: Poisson Solver Library

Person Months: 4

The motivation for this project was to enhance DL_POLY_4 [11] capability in terms of (i) enabling it to treat accurately and efficiently electrostatic interaction under non-periodic boundary conditions, and (ii) preparation for exascale computing involving simulations of very large (~10^7 atoms) systems on extremely large core counts, where the FFT component of SPME and P^3M methods is known to lose its scalable efficiency very quickly and exhibit poor performance due to excessive non-local communications and large memory requirements.

In this project, we demonstrated that the real-space Poisson solver methodology we developed provides a viable alternative to the aforementioned methods, which is especially indispensable in the case of model systems with no periodic boundary conditions. The Poisson solver works in a truly memory distributed manner with very good strong scaling performance for the systems sizes investigated. The decline in performance at large processor counts was due to limits of strong scaling where communication becomes dominant over computation due to increase of small size messages volume and decrease of compute per core.

Our enabling effort exploits a HPC technique based on solving the Poisson equation as an alternative to using FFT techniques, which have been described in detail in deliverable, D7.2.1 ‘A Report on HPC Tools and Techniques’ [1], for molecular systems, which are not subject to periodic boundary conditions. The methodology offers a memory distributed solution of the electrostatic interactions, reducing a nonlocal N^2 problem to a local linear one, subject to communication costs in a manner similar to 3D FFT methodology.

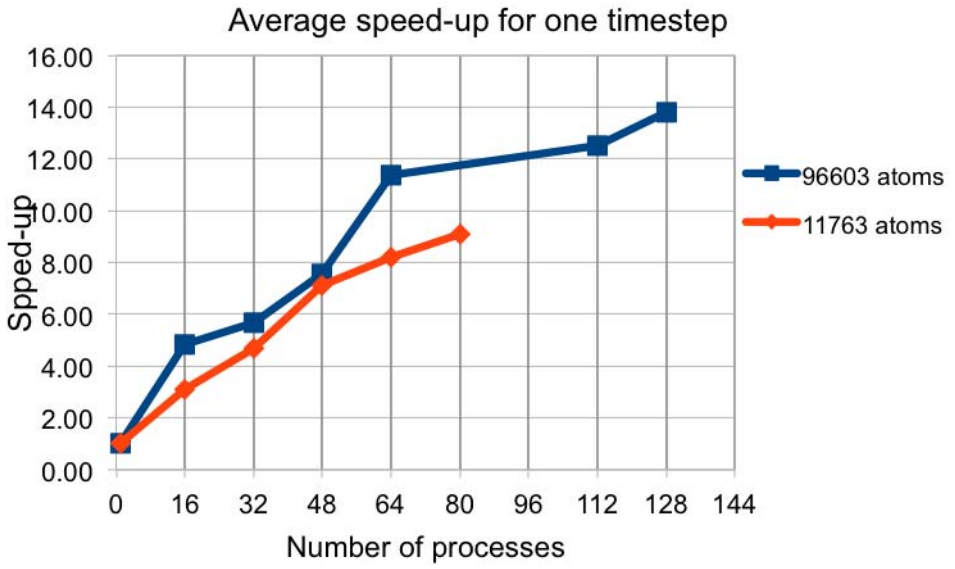

Figure 10: Scaling performance of the Poisson Solver module in DL_POLY_4 (relative to 1 MPI process) vs. number of MPI processes for a single time-step

Due to the long-range nature of the electrostatic interactions, their treatment is of crucial importance in the context of studying the materials' properties at atomistic and molecular level by means of molecular dynamics (MD) simulations. The electrostatics acts on polar or charged molecules, including water, ions, amino acids, nucleic acids, carbohydrates, and lipids, and hence greatly influences their behaviour and properties, including solvation, folding and binding. Thus, it is essential that the electrostatic interactions be evaluated with a high level of accuracy in MD simulations. However, their long-range and many-body character makes their evaluation immensely computationally demanding – in fullness, this is a N^2 problem, N being the number of particles in the model system. A number of techniques have been developed to solve the problem in a more scalable manner, of which the most widely used, especially in computational bio-chemistry and bio-physics, are SPME and P^3M. Both of the latter methods are based on the Ewald summation method and use Fourier transforms as a central operation. The application of these Ewald-based techniques is however limited to model systems with 3D periodic boundary conditions (PBC) in its original formulation. An alternative approach is to use a continuum electrostatics model that allows simulations under general boundary conditions, for example by solving the Poisson equation. Such a technique can be efficiently employed to evaluate accurately the electrostatic contribution to the free energy, for modeling of biomolecular titration states, computation of the electrostatic potential of lipid bilayers, ion-channel characteristics calculations, etc. Moreover, the real-space Poisson solver methodology we developed provides a viable alternative, indispensable in the case of model systems with non-periodic boundary conditions.

We tested the performance and scalability of our DL_POLY Poisson solver module on two model systems with 3921 (~100 Å box edge) and 32201 (~200 Å box edge) TIP3P water molecules, respectively. The test simulations were performed on a Linux cluster with Intel Xeon E5540 at 2.53GHz chips. The reported results were obtained using the *GNU* Fortran90 compiler, *gfortran* version 4.8.1, with O3 level of optimization and the execution times were averaged over 20 timesteps

The models systems intermolecular interactions (van der Waals and Coulomb) were handled with a smaller cutoff (of 5 Å instead of 8-10 Å) than usual in order to warrant a linear scaling of the linked-cells algorithm for construction the Verlet neighbour list (VNL) and have the

Poisson solver routine as the dominant contribution of the force evaluation cycle. This assumption will hold true up to 64 and 1024 MPI counts for the small and the large system respectively. The performance data for the model systems run under the conditions outlined above in Figure 10 shows performance degradation at high core counts, which is due to the increase of the volume of messages of smaller sizes as the processor counts increase. The missing points indicate run failures due to grid-size mismatch on the selected processor counts as processor grids dictated unfavourable grid-cell sizes for the linear 27-stencil approximation of the potential gradients. A direct comparison between FFT and PS is not possible, as FFT relies on periodic boundary conditions, while PS is used when these are not required; this is the key advantage of the proposed method, as it allows going beyond the limitations of FFT, which is crucial for systems not to be handled in the periodic boundary conditions context.

3.5 Enabling OpenFOAM for Bio-medical Flow Simulations

WP162: *Scalability of OpenFOAM for bio-medical flow simulations*

Authors: A Duran (ITU-UHeM), S Celebi (ITU-UHeM), S Piskin (ITU-UHeM) and M Tuncel (ITU-UHem)

Application: OpenFOAM

HPC Tool/Technique: SuperLU_MCDT

Person Months: 5

In this project, we have investigated the challenges facing CFD solvers as applied to bio-medical fluid flow simulations and in particular the OpenFOAM [40] 2.1.1 solver, `icoFoam`, for the large penta-diagonal matrices coming from the simulation of blood flow in arteries with a structured mesh domain. A realistic simulation for the sloshing of blood in heart or vessels in an entire body is a complex problem and can take thousands of wall-clock hours for several main tasks such as pre-processing (meshing), decomposition and solving large linear systems. We generated a structured mesh by using `blockMesh` as a mesh generator tool. To decompose the generated mesh, we employed the `decomposePar` tool. After the decomposition, we used `icoFoam` as a flow simulator/solver tool. For example, the total run time of a simple case is about 1500 hours without preconditioning on one core for one period of cardiac cycle. Therefore, this important problem deserves careful consideration for potential multi-petascale or exascale computing usage.

We started from the relatively small instances for the whole simulation and focused on one time step simulation for solving large linear systems. This version gives important clues for a larger version of the problem. Later, we increase the problem size and time steps to obtain a better picture gradually, in our general strategy. We test the performance of the solver "icoFoam" for its potential multi petascale or exascale computing usage at TGCC Curie (a Tier-0 system) platform at CEA, France. We achieved scaled speed up for large matrices such as 64 million x 64 million matrices up to 16384 cores. In other words, we find that the scalability improves as the problem size increases for this application. This is an important and encouraging result for the problem. On the other hand, we observe software and hardware limitations, which may be solved, when we test for 128 million x 128 million matrices.

Several realistic bio-fluid flow simulations have been carried out [47][48]. The geometries are extracted from real patients or generated using real patient data from CT or MRI scans. Measured flow rate at vessel inlet by ultrasound technique is used to get the velocity profile of the simulation geometry inlet. The parallel runs scale well depending on the preconditioning

and mesh size. Some of the results obtained using OpenFOAM icoFOAM solver is shown in Figure 11.

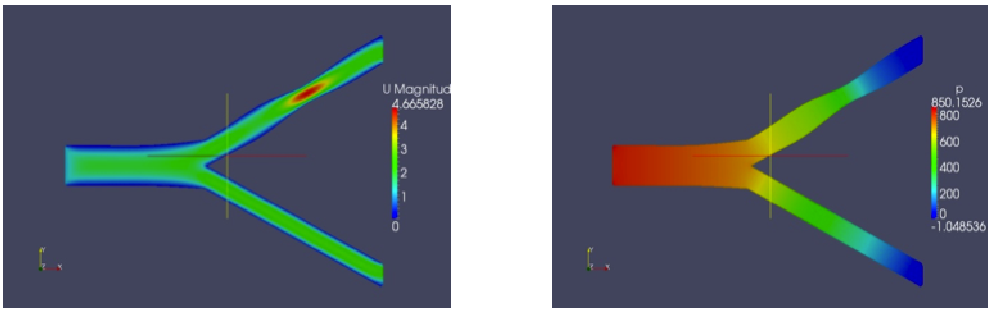

Figure 11: The results obtained using OpenAFOAM icoFOAM solver

We test for several matrices of various mesh sizes, after the decomposition, which takes several hours for each one. Here, we obtain the scalability results for the large sparse systems with three matrices of size 8 million x 8 million up to 1024 cores; 32 million x 32 million up to 8192 cores; and 64 million x 64 million up to 16384 cores. The code has shown almost linear speed-up up to 16384 cores for the largest matrix in our tests. Moreover, we obtain that the scalability becomes better as the problem size increases. Figure 12 illustrates this scaled speed-up for these three large matrices having sizes up to 64 million x 64 million up to 16384 cores.

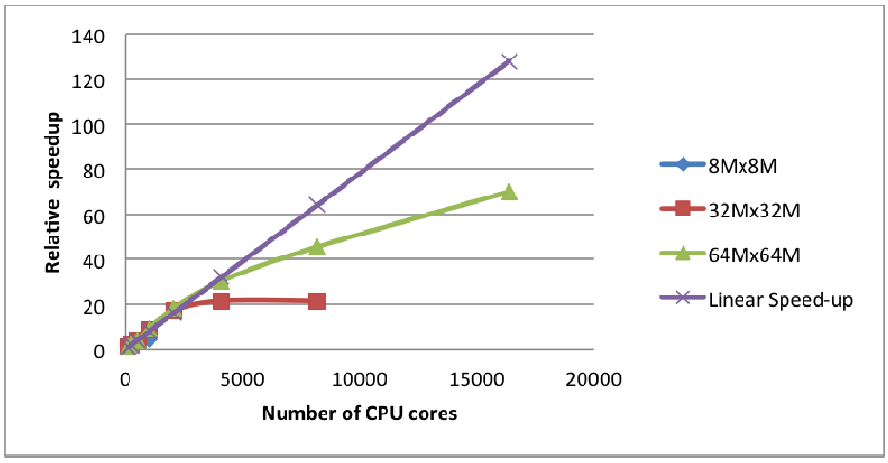

Figure 12: Scalability of icoFOAM solver on CURIE

As part of this project, we have also embedded other direct solvers (kernel class) such as SuperLU_DIST [56][57] and SuperLU_MCDT (Many Core Distributed) [58][59] in addition to default solvers provided by OpenFOAM. We continue the tests of OpenFOAM having the direct solvers to take advantages of them for robustness. In order to show better usability of our many-core enabled direct method compared with iterative methods in blood flow simulations, the coefficient matrices with high condition number in transient flow conditions need to be tested.

It is a well-known fact that during the flow simulations both coefficient matrices and right hand side vector are changing. This change is especially drastic during the severe flow dynamics conditions in simulation. This drastic change, in most cases, shows itself as an ill-conditioned spectral space and high condition numbers. It is important to observe that, for high condition numbers, the time complexity gap between direct and iterative solvers for the solution of linear set of equations will be reduced. For more complex flow conditions the solution time of iterative solvers will increase based on the fixed solution precision. In this

case direct method's cost is still fixed and the potential gap in solution time is expected to be reduced. It is also worth noting that iterative methods work on both coefficient matrix and the right hand side vector changing at each time step but our direct solver works only on the coefficient matrix. This is also a potential advantage for our direct solver in case of large simulation times.

Our many-core aware direct sparse solver has a capability of harvesting the potential benefits of many core distributed systems than any other sparse direct solver especially for unsymmetric matrices. Future exascale systems are expected to be having heterogeneous and many core distributed nodes. We believe that our SuperLU_MCDT software is a strong candidate for these future systems with its strong scalability on many-core distributed servers we tested. Our initial tests on 16K cores show that with the large matrices it scales without observing any significant performance loss. Potential challenges for our many-core aware software is resilience, accelerator support and hyper graph partitioning for both better scalability and sustainability**.**

3.6 Towards the Implementation of an Algebraic Multi-Grid Solver for Lattice QCD on Exascale Hardware

WP163: *Algebraic Multi-Grid Solver for Lattice QCD on Exascale Hardware: Intel Xeon Phi*

Authors: A. Abdel-Rehim (CASTORC), G. Koutsou (CASTORC), C. Urbach (University of Bonn)

Application: tmLQCD

HPC Tool/Technique: AMG Solver

Person Months: 8

In Lattice QCD simulations, 70%-90% of computer time is used to solve a large sparse linear system of equations. Moreover, current and future Lattice simulations are performed at physical light quark masses, large volumes and small lattice spacing. This makes the linear system very ill conditioned. Standard algorithms to solve these systems such as the Conjugate Gradient (CG) algorithm become inefficient in this regime making simulations very slow. As we approach the regime of large-scale simulations on Exascale machines, it is important to use efficient numerical algorithms. Most probably, future Exascale machines will be hybrid machines with multicores on every compute node in addition to one or more many-core accelerators or co-processors such as the Intel Xeon Phi. These efficient algorithms have also to be adapted to the underlying hardware architecture.

We consider here a promising algorithm to solve a large sparse linear systems based on Algebraic Multi-Grid (AMG). The particular implementation considered here is given in [60]. The advantage of this algorithm over standard algorithms such as CG is its efficiency in dealing with low modes of the Lattice Dirac operator. It converges much faster than CG especially on ill conditioned systems such as those that will appear if future Lattice simulations. In addition, this algorithm uses Schwartz Alternating Procedure (SAP) as a preconditioner. In SAP we solve the linear system restricted to local blocks of the lattice (which will be on a single MPI process) in order to find the solution of the global system. This will have the effect of reducing communications since all operations will be done on the local process. This feature indicates suitability for future exascale machines where one would prefer to reduce the communications among MPI processes as much as possible.

We provide an implementation of this solver inside the tmLQCD lattice software suite [61]. This software suite is publically available [62] and is widely used by the European Twisted Mass Collaboration (ETMC). It provides codes for Monte Carlo simulations with Wilson,

Clover and Twisted-Mass fermions. The tmLQCD package uses a hybrid parallelization with OpenMP and MPI making it suitable for hybrid architecture with the Intel Xeon Phi as a co-processor on each compute node together with multicores. The AMG solver is implemented for Twisted-Mass fermions with or without a clover term. The AMG based preconditioner is then used with the Flexible GMRES algorithm (FGMRES).

For the Xeon Phi, the code can be run either in a native mode or an offload mode. However, in the case of a native mode, memory limitations restrict the applicability to rather small lattices. Offload mode, where the large sparse matrix-vector multiplication as well as small linear algebra operations is run on the Xeon Phi while the deflation vector space is stored on the host CPU. In addition to these memory considerations, it is also necessary to vectorize the code to benefit from the wide registers available on the Xeon Phi. These registers allow for a simultaneous operation on 8 double or 16 single precision numbers. Most linear algebra operations appearing in lattice codes such as DAXPY can be easily mapped to these registers offering an improved performance. However, for the multiplication of gluon field matrices which are 3x3 complex unitary matrices times a 3x1 complex colour vector, this mapping is not straightforward. The lattice Dirac operator which provides the large sparse matrix-vector multiplication involves 8 multiplications of these 3x3 matrices times a colour vector for each lattice site. These multiplications however use different matrices and vectors at nearest neighbours of the site under consideration. Mapping these operations to the registers on the Xeon Phi requires a redesign of the data layout on the lattice data. In this work we considered a mapping in which the 3x3 matrices and vectors at more than one site are packed together into a larger structure and then the multiplications are done simultaneously on the matrices on this structure. Two options are considered. The first is to use complex matrices, in which case one pack together the data at 4 sites. The second is to separate real and imaginary parts which simplifies the operations and in this case we pack the data on 8 sites. The advantage of this packing approach is that the local lattice volume for each MPI process is a multiple of 4 or 8 in most lattice simulations. In addition, this approach can be extended to the case when registers become wider as expected to be the trend in the future. In this work we have implemented the vectorization of the 3x3 matrix vector multiplications as a first step towards vectorization of the whole Dirac operator sparse matrix-vector multiply. Further work is needed for the full vectorization which will be the subject of further tuning and optimization. More details and a sample of the vectorization code are given in the related white paper WP163.

The AMG solver based on FGMRES is first tested on a multicore machine. The test is performed on a Cray XC30 system. Each compute node has two sockets and each socket has 12 cores Intel Ivy Bridge processor at 2.4 GHz. TmLQCD is compiled with the Intel compiler and configured with the options:

```
./configure --enable-mpi -enable-sse2 --with-mpidimension=4 --
with-lapack CC=cc F77=ftn
```

We consider two gauge configurations from the ETMC configuration archive with up, down, strange, and charm sea quarks. Parameters of these configurations and their label in the ETMC database are shown in Table 10.

Label	b	L/a	T/a	$a\mu_l$	$a\mu_\sigma$	$a\mu_\delta$	κ
B85.24	1.95	24	48	0.0085	0.135	0.170	0.1612312
D15.48	2.1	48	96	0.0015	0.120	0.1385	0.156361

Table 10: Parameters of the gauge configurations used in testing the AMG Solver.

The full set of parameters used for the solver as given in the input file for the "invert" executable can be found in the PRACE 3IP whitepaper WP163. We solve 20 random right-hand sides and compute the accumulated time to solution for FGMRES. We compare the timing to the CG solver with Even-Odd preconditioning (CG_EO). CG_EO is the most efficient standard solver for tmLQCD with Twisted-Mass fermions. It is to be noted that for the FGMRES (AMG) solver, there is a setup cost for generating the deflation subspace. This setup cost is only a single time cost, however, it has to be taken into account. This is shown in following plots as the time associated with right-hand side number 0. CG_EO doesn't have this setup cost. In Figure 13 we show the results for the smaller and larger lattices involved. For the small lattice, we use 32 MPI processes while for the larger lattice we use 512 MPI processes. The lattice size in D15.48 is the largest lattice available at the moment. In addition, because the code requires the number of lattice blocks for each process to be even and given the size of the blocks used, this limits the number of MPI processes to 32 and 512 respectively. Future simulations will use much larger lattices. As can be seen from the results a gain factor between 2 and 4 in solution time is observed.

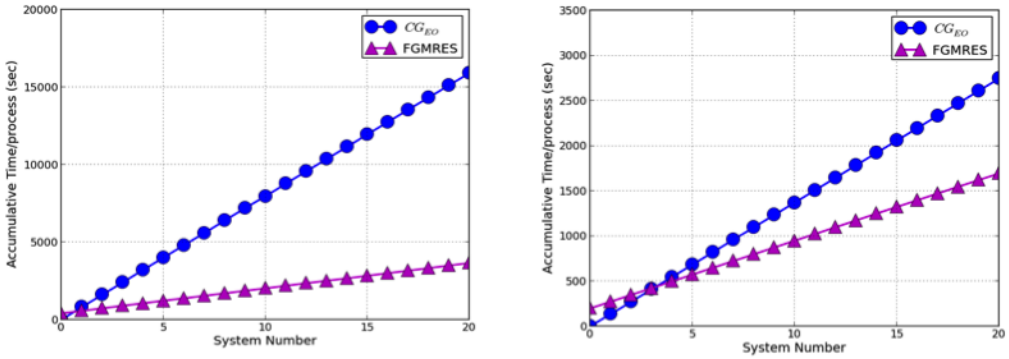

Figure 13: Accumulative time to solution for each MPI process (sec) for solving 20 linear systems. Left: B85.24 lattice on 32 cores. Right: D15.48 lattice on 512 cores.

On the Xeon Phi, we performed tests on the cluster at CSC in Finland. The TmLQCD code is compiled in a native mode using the configure command

```
./configure --enable-mpi --enable-omp --enable-alignment=64 --
enable-gaugecopy --enable-halfspinor --with-mpidimension=4 --
with-lapack CC="mpiicc -mkl -mmic" F77="ifort -mkl -mmic"
CFLAGS="-std=c99"
```

As a first test we tried to run the inverter on the B85.24 ensemble on a single MIC card natively using the same parameters as described before. However, this turned out to take a large amount of memory and the code crashed after performing part of the computation. This highlights the anticipated difficulty of limited memory to run the entire code on the MIC. A redesign of the solver will be necessary to take place if one attempts to use the MIC. Alternatively more MIC cards should be used to overcome this problem. Since we had only limited number of MIC cards available we had to use smaller lattice of size L/a=16, T/a=32 hoping that it will fit in memory. The run is done using openMP with 54 threads. In this case the solver successfully completed and there were no memory issues.

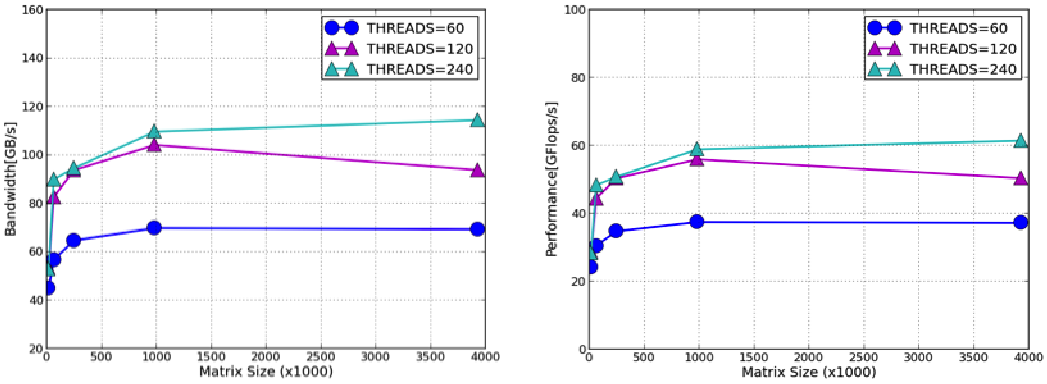

Figure 14: Performance of the vectorized matrix-vector multiplications on a single Xeon Phi card. Left: Bandwidth. Right: FLOPs/s

We also test the performance of the vectorized 3x3 matrix vector multiplications. In Figure 14 we show performance results. In this case we pack 4 complex matrices together of an implementation of the matrix-vector multiplications. The results are encouraging, although they are lower than what we would hope for given the peak performance of about 1 Tflops/s that the MIC can provide.

3.7 Parallelisation of Sparse Matrix Kernels for Large-Scale Scientific Applications using the MapReduce Paradigm

WP159: *Map/Reduce-based Parallelization of Sparse Matrix Kernels for Large-Scale Scientific Applications*

Authors: G V Demirci (BILKENT), A Turk (BILKENT), R Oguz Selvitopi (BILKENT) K Akbudak (BILKENT) and C Aykanat (BILKENT)

Application: Sparse Matrix Alegbra Kernels

HPC Tool/Technique: MR-MPI

Person Months: 5

The scientific computing community has recently tried to exploit the benefits of the MapReduce programming model in several works [64][65][66][67][68][69]. Among such attempts, MR-MPI [64] is a library developed with the aim of enabling usage of the MapReduce paradigm in scientific computing. It is a lightweight C++ library that uses MPI for inter-processor communication. It is a small and portable library that consists of a few thousand lines of code and provides the core functionality of the MapReduce paradigm with a strong focus on performance and scalability. It sacrifices some functionality usually provided by common MapReduce libraries such as fault tolerance and data redundancy. However, these issues generally arise in the context of distributed systems with commodity machines and they are of not of primary concern on Tier-0 systems, but will, however, become so as we advance towards the exascale era.

In this work, we exploit the MR-MPI library to develop a sparse matrix-vector multiplication (SpMV) kernel, which has the potential to be further used in community codes that leverage such kernels during execution. We designate the multiplication of a single column of the coefficient matrix with the corresponding vector entry as a single atomic task. These atomic tasks correspond to mapper tasks in MapReduce paradigm. The columns and the corresponding vector entries are distributed among mapper tasks using the built-in hash

functions of MR-MPI library, which basically perform the partitioning process in a randomized manner. This naïve distribution of columns of the coefficient matrix does not take the data locality into consideration, which is often inherent in coefficient matrices.

MR-MPI allows users to define and use their own hash functions. Usage of part vectors as hash functions allows one to reduce communication requirements of the SpMV operations by minimizing and/or balancing volume of communication. By utilizing user-defined hash functions, we can distribute mapper and reducer tasks and achieve data locality to further reduce communication requirements of parallel SpMV multiplies. In this work, we propose two such techniques to reduce and balance the communication volume: (i) graph-partitioning model and (ii) hypergraph-partitioning model. The former method correctly encapsulates the communication volume incurred in the SpMV operations performed *without* extra convert-and-reduce operations prior to the 'collate' stage while the latter one achieves the same feat *with* extra convert-and-reduce operations prior to 'collate' stage. The partitioning process is a pre-processing stage performed on the matrix used in SpMV operations. The output part vectors obtained at the end of the pre-processing phase are utilized as hash functions in MR-MPI for distributing mapper and reducer tasks among processors.

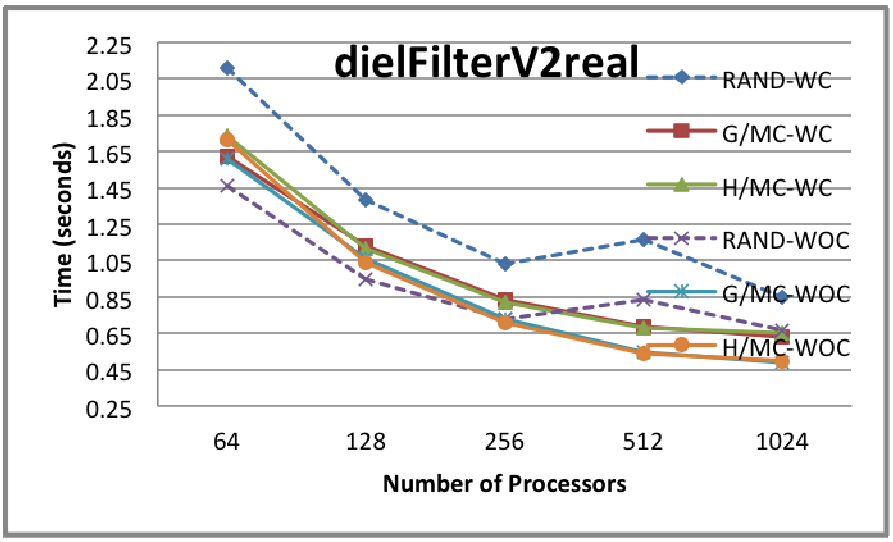

Figure 15: Comparison of mapper/reducer task assignment strategies on SpMV for the dielFilterV2real Matrix

We present our results on *dielFilterV2real* matrix [72] with 48 million nonzeros for number of processors $P = 64, 128, 256, 512, 1024$. Tests were performed on NTNU's Vilje system (an SGI Altix ICE X system). The mapper and reducer task assignments are achieved by using user-defined hash functions as part vectors. We compare three alternatives that are used for the assignment of mapper and reducer tasks in MR-MPI: (i) RAND, (ii) G/MC (graph-partitioning method), and (iii) H/MC (hypergraph-partitioning method). The partitioning of the graph constructed with G/MC and the hypergraph constructed with H/MC are partitioned with MeTiS [70] and PaToH [71] respectively. The obtained results are summarized in Figure 15.

As seen from Figure 15, using a random hash function leads to poor scalability, especially when processor counts increase, which is the case for RAND-WC and RAND-WOC. The results show that to obtain a good scalability performance, various optimization techniques centered on communication requirements of the SpMV are necessary. We believe that the Map/Reduce paradigm has great potential to achieve exascale performance which accommodates several important aspects within itself such as providing a unified interface, the ease of programming and fault tolerance.

4 Debuggers and Profilers

A wide range of state-of-the-art tools for debugging and performance analysis were reported on in D7.2.1, where a particular focus was placed on performance analysis tools which were being exploited and developed in various exascale initiatives throughout Europe and the US. Improving debugging and performance analysis tools is deemed a necessity for efficient use of upcoming exascale architectures: If we are to enable applications on such systems, then we need to have as clear a view as possible of the barriers to achieving performance. PRACE application experts have long been faced with a landscape where the optimal executable cannot be produced by a traditional compiler alone. Different execution strategies needs to be considered, loops need to be unrolled, possibly resulting in statements that can be executed in parallel by several threads, costly memory operations need to be minimized, and caches need to be properly utilised both among traditional multi-core processors as well as accelerators.

As well as using profilers, this complex jungle of optimization strategies suggests the benefits of running auto-tuning tools alongside profilers in order to better understand code behaviour, identify parts of the code suitable for acceleration and improve overall performance. Indeed, there is a view that for exascale systems, auto-tuning will be mandatory, where at such scale, it will be necessary for auto-tuning to be continuous throughout each execution of the application. This requirement is driven by several factors. First, the sheer number of threads (on the order of one billion) will mean that dynamic load balancing will almost certainly be required. Second, the presence of energy limits (either in the form of total consumption or thermal throttling due to heat dissipation) will mean that the performance of cores will be dynamically varying. This will mean that the exact performance of the hardware will not be known until runtime. Finally, it is likely that hardware failures and faults will be the norm and not the exception. This will also contribute to dynamic changes in the hardware available to run applications. Such challenges, have motivated us to go slightly beyond our original plan, which was to focus only on performance analysis tools, and instead also explore the possibility of exploiting auto-tuning tools in PRACE applications, which have so far received little attention in PRACE activities to date.

In this section, we report on two projects that have each focused on exploiting state-of-the-art profilers (and in one case an auto-tuning tool) in order to help enable applications for multi-petaflop/future exascale systems. Each subsection provides a summary of the project along with a reference to the PRACE-3IP whitepaper associated with the project [2]. We recommend that the reader refers to the whitepaper for each project, which provides a more detailed report on the projects than is provided here.

The profilers and auto-tuners that we have focused our attention on during the exploitation phase along with the applications that were focused on are listed in Table 11

HPC Tool/Technique	*Application*
Tau Profiler	Code_Saturne
HPCToolkit	Code_Saturne
Orio auto-tuning tool	Code_Saturne
Extrae profiler	Alya

Table 11: HPC Tools and Techniques (Debuggers and Profilers) exploited along with corresponding applications

Tau is an integrated toolkit for performance instrumentation, measurement and analysis. It provides a flexible, portable and scalable set of technologies for performance evaluation on

extreme-scale HPC systems. We have also investigated Extrae, which is a profiling tool being developed at BSC and has come under particular attention as part of several exascale initiatives in Europe.

As pointed out in D7.2.1, this type of assessment of debugging and profiling tools has rarely been seen in PRACE reports or whitepapers to date. A substantial effort of training on tools for debugging and performance analysis has been carried out within PRACE. However, very little is documented on how successfully these tools have been employed within enabling projects. One of our missions here is to fix this discrepancy in order to understand the full benefits and limits of such tools in extreme cases.

4.1 Profiling Code_Saturne with TAU and auto-tuning kernels with Orio

WP149: *Profiling of Code_Saturne with TAU and autotuning kernels with Orio*

Authors: B Lindi (NTNU), T Ponweiser (JKU), P Jovanovic (IPB), T Arslan (NTNU)

Application: Code_Saturne

HPC Tool/Technique: Tau, Orio, HPCToolkit

Person Months: 12

Profiling is an important and necessary step for enabling software for multi-petaflop/future exascale computing. As increased parallelism is the only way to increase performance, insights to work load distribution and an application's execution profile are prerequisites for exascale enabling. While profiling gives the execution behaviour for an executable, auto-tuning offers methods to let compute resources automatically explore the parameter space governing the run time behaviour in combination with a dataset. In this project we have profiled Code_Saturne [8] and based on the profiling results we have identified functions, or compute kernels suitable for auto-tuning. *Code Saturne* is an open source software package, which can be used for CFD simulations. It solves the Navier-Stokes equations for different types of flow.

The tools we have used for profiling and auto-tuning are TAU [73], HPCToolkit [74] and Orio [75], where each of the tools were identified in deliverable D7.2.1 'A Report on HPC Tools and Techniques' [1] as having potential for helping to enable applications on multi-petaflop/future exascale machines. The TAU Performance System is a portfolio of tools for carrying out performance analysis of parallel programs written in Fortran, C, C++, UPC, Java, Python. TAU is capable of gathering performance information through instrumentation of functions, methods, basic blocks, and statements as well as event-based sampling. HPCToolkit is a suite of tools for performance measurement and analysis, which is based on statistical sampling of timers and hardware performance counters. Orio focuses on thread-level ("intra-node") optimization. It is a tool for auto-tuning of performance critical regions of code - typically at the level of C or Fortran loops. Loop bounds and index increments are formulated in a C-type language with certain restrictions. In a run-time critical region of code, first a computation specification for Orio has to be created. In case that the original application is written in C, this step usually involves only marginal adaptations to the given original code. Secondly, the computation specification has to be annotated in order to instruct Orio which code transformations to apply (e.g. loop unrolling) and which optimization parameters to use for these transformations (e.g. loop unroll factors). All optimization parameters together with their associated allowed value ranges give rise to an (exponentially large) optimization parameter space. Orio generates, compiles, executes and times many different program versions. It reports the configuration, which resulted in the best performance as well as the associated generated code. The project has been carried out on

local workstations or on the Norwegian HPC-cluster “vilje”.”vilje” is a SGI ICE X system consisting of 1404 nodes connected with a Infiniband FDR fabric and a Lustre parallel file system.

We profiled two test cases with different mesh sizes, the T-junction case from the Code Saturne tutorial and the Tube test case from earlier work in PRACE [76]. We present here the results from the Tube Case with a mesh size of 51 million cells profiled with TAU and a mesh size of 200 million cells profiled with HPCToolkit.

Figure 16 shows parts of the profiling result from the Tube Case with 51 million cells executed over 4096 CPU-cores. The case was executed over 1024, 2048 and 4096 CPU cores. In all cases the functions `_mat_vec_p_l_native()` and `_iterative_scalar_gradient()` have the largest exclusive time. As the execution time drops with increasing CPU-core count, the total wall clock time is around 527, 368 and 302 seconds for the different CPU-core counts, respectively, we experience increasing exclusive times for MPI calls indicating that an increasing proportion of the execution time is spent on communication.

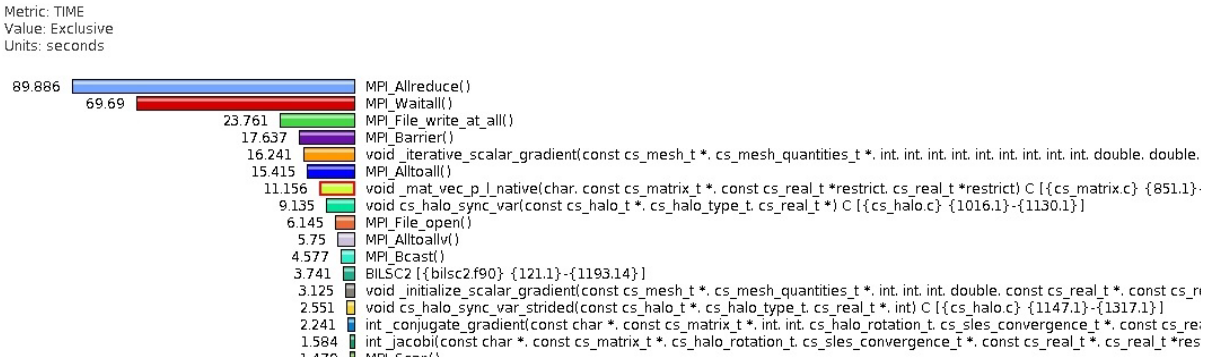

Figure 16: Profiles of functions (averaged on 4096 cores-256 nodes) from Code_Saturne in decreasing order of exclusive time, 51M case.

As a simple metric for identification of performance critical kernel routines, we select the number of processor cycles (PAPI_TOT_CYC) spent within a given routine *R*, excluding those cycles which are spent in routines called by *R*. We call this metric the exclusive time of a routine. The mesh size for the Tube bundle case is 200 million cells and case has been executed across 1024, 2048 and 4096 CPU-cores. In all cases, the routine `_mat_vec_p_l_native()` has a relative high contribution to the overall runtime. The routine `_iterative_scalar_gradient()` seems to be the most critical only for the Tube bundle test case. The explanation for this seems to lie in the fact that `_iterative_scalar_gradient()` is significantly faster for perfectly orthogonal meshes than for more typical meshes with moderate non-orthogonality, like the Tube bundle case.

The above profiling results suggested that our focus be placed on the two kernel routines `_mat_vec_p_l_native()` and `_iterative_scalar_gradient()`. Scalability results can be seen in Figure 17. For the following, we introduce the abbreviations MVN (`_mat_vec_p_l_native()`) and ISG (`_iterative_scalar_gradient()`) for these two routines

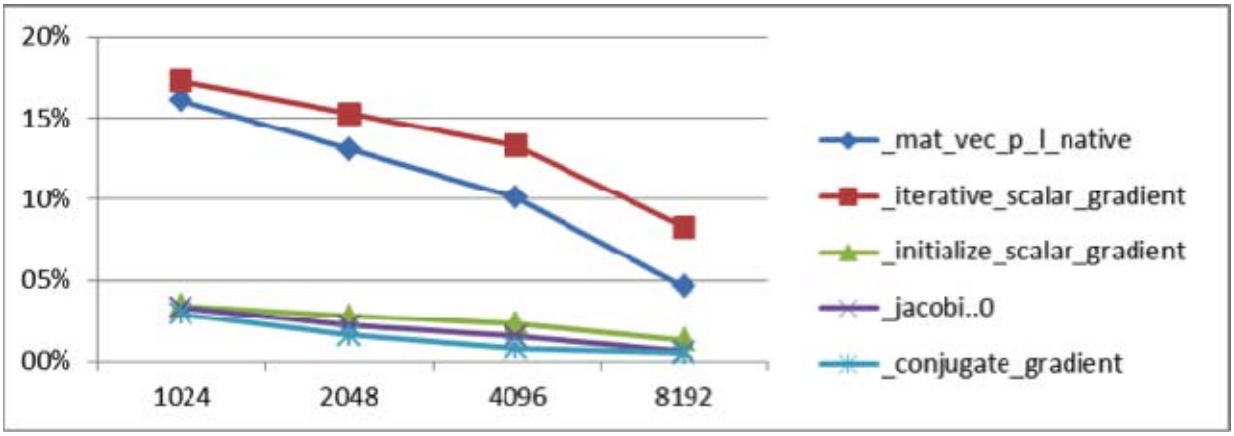

Figure 17: Visual comparison of hotspot routines for the Tube bundle test case

Following Orio's standard workflow, we isolated the two performance critical loops of MVN and ISG in two separate input files for Orio and annotated the code with appropriate auto-tuning instructions. We used the 200 million cells Tube bundle test case with 1024 MPI processes, which we consider to be representative for typical use cases. The generated data files served as input for the Orio auto-tuning process.

In a first step, we applied only loop-unrolling to the selected loops of MVN and ISG. For MVN, the annotated code for Orio is listed in Figure 18. The Orio-generated code - Orio reported 5 as optimal unroll. Indeed the tuned code is measurably faster than the original code. However the speedup is not particularly high: For both routines, MVN and ISG, the total speedup ranges approximately between 3 and 5%, comparing generated and original code with full compiler optimization (-O2 or -O3).

Indeed, a deeper analysis of MVN and ISG shows that both routines suffer from the same fundamental problem: both selected loops iterate exactly once over all internal faces of the process-local part of the mesh. Note that the loop in ISG is a bit more advanced in that the iteration is partitioned into multiple passes, where in each pass independent groups of faces are treated in parallel by multiple threads.[1] However, in our single-threaded testing setup, this degenerates essentially to same type of iteration as in MVN. The total number of iterations for the outer two loops in ISG is exactly one; the innermost loop iterates over all internal local mesh faces for our test case.

For both, MVN and ISG, the indices *ii* and *jj* refer to the two mesh cells which are adjacent to the mesh face of the current iteration. Analysis of the exported runtime data shows that these indices jump very arbitrarily from iteration to iteration. Consequently, the (read and write) access of associated cell data is very inefficient because cache misses occur with very high probability. It seems that things get even worse if this irregular data access pattern is carried out by multiple threads in parallel. Here, in addition to expensive load and store operations from/to main memory, also false sharing effects are observable (threads are frequently invalidating each other's cache lines). We assume that this observation might be a possible explanation for the decreased efficiency when comparing the hybrid (MPI + OpenMP) variant of Code_Saturne with a purely MPI-parallelized version.

Due to the irregular data access pattern in both selected loops, loop vectorization is not applicable. Moreover, loop parallelization has been excluded in the first place, because our initial investigations showed that Code_Saturne performs best when running with single-threaded MPI processes.

[1] In this context we call two faces independent, if they have no adjacent cell in common.

```
/*@ begin PerfTuning (
  def build { ... }
  def performance_counter { ... }
  def performance_params {
        param UF[] = range(1,8);
  }
  def search {
        arg algorithm = 'Exhaustive';
  }
  def input_params { }
  def input_vars
  {
        arg decl_file = 'decl.h';
        arg init_file = 'init.c';
  }
) @*/

  int face_id, ii, jj;
/*@ begin Loop( transform Unroll(ufactor=UF)

  for (face_id = 0; face_id <= n_faces-1; face_id++) {
ii = face_cel_p[2*face_id] -1;
     jj = face_cel_p[2*face_id + 1] -1;
     y[ii] += xa[face_id] * x[jj];
     y[jj] += xa[face_id] * x[ii];
}

) @*/
/*@ end @*/
/*@ end @*/
```

Figure 18: Orio input file for optimization (loop-unrolling) of `_mat_vec_p_l_native`**.**

We come to the conclusion that for a further optimization of MVN and ISG, a smart reorganization of the involved data structures (such as for example the introduction of a thread-level mesh decomposition) as well as appropriate algorithmic reformulations are required. Clearly such fundamental changes to the code lie beyond the capabilities of Orio (and most likely beyond the capabilities of any other fully-automated tool available in the foreseeable future).

4.2 Performance Analysis of Alya on Multi-petaflop Systems using Extrae

WP151: *Performance Analysis of Alya on a Tier-0 Machine using Extrae*

Authors: J Rodriguez (BSC)

Application: Alya

HPC Tool/Technique: Extrae profiler

Person Months: 2

Alya [77] is a computational mechanics code capable of solving different physics. It has been extensively used in MareNostrum III (BSC's Tier-0 machine), and it has been also used as a benchmarking code in PRACE Unified European Applications Benchmark Suite [78]. In this section, Extrae [79] and Paraver [80] will be used to collect and analyze performance data during an Alya simulation in a petaflop environment.

Alya is a code developed at BSC, which solves partial differential equations (PDEs) in non-structured meshes, using finite element methods. Among the problems it solves are: convection-diffusion reactions, incompressible flows, compressible flows, turbulence, bi-phasic flows and free surface, excitable media, acoustics, thermal flow, quantum mechanics (DFT) and solid mechanics (large strain).

The source code is written in Fortran 90/95 and parallelized using MPI and OpenMP. Being a large-scale scientific code, Alya demands substantial I/O processing, which may consume considerable time and can therefore potentially reduce speed-up at exascale.

It is organized in modules that solve different physical problems. These modules are now being used in production, and scalability has been proven on 20,000 cores using meshes with billions of elements. The code has been tested and proven to run efficiently on many Tier-0 machines such as JUQUEEN/FERMI (BlueGene/Q), CURIE (Bull Cluster) and other Intel-based clusters around the world.

In the current section, we will use Alya executions with Test Case B as input (a 27 million mesh representing the respiratory system), which can be obtained from PRACE UEABS. These executions will be instrumented with Extrae, and as a result, we will get a trace file that can be visualized with Paraver.

Trace files obtained with Extrae [79] collect a large amount of data (hardware counters, states, bursts, events), which may lead to a trace file of several gigabytes in size (e.g. the 2048-core execution generated a 66 GB trace file). As it is not possible to manage such an amount of data on an average computer, there are some options to reduce the trace file size to a few gigabytes or even some megabytes in size. Among these options, a trace file can be cut just for a specific time interval, or the trace file can be filtered to show some of the events in the trace.

After a close look on an iterative section of the execution, we can see some load imbalance in the `par_operat()` Fortran routine between the different processes Figure 19. According to the associated histogram produced by Extrae, at this part of the execution, the timings in `par_operat()` calls vary from 106 ms to 292 ms.

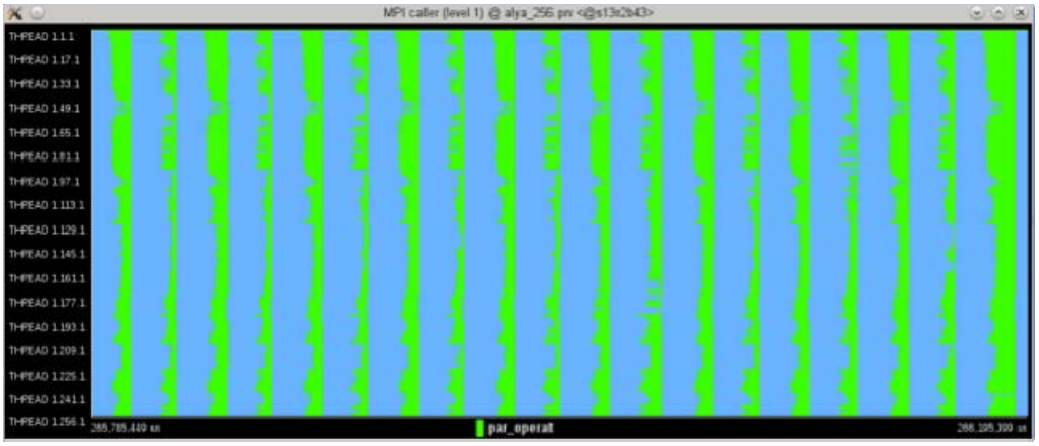

Figure 19: Profile of the iterative part of Alya on 256 processes (zoomed in)

If we look at the source code for subroutine `par_operat()` we can check that depending on the process rank there are different loops and different `MPI_AllReduce()` calls to be executed, which will explain the mentioned behaviour.

As a result of the study, we have verified the powerful options provided by Extrae and Paraver in terms of detail, insight and parallel behaviour.

Regarding the use of Extrae and Paraver, we confirm that they are very valuable tools when getting an in-depth analysis of a code's behaviour. We also conclude that it is a key toolset in the exascale path because of its low level detail of hardware analysis. However, when generating trace files for a very large amount of cores (10,000 cores or above), we may have some intricacy in merging the performance data into one single trace file, or even managing a trace file of several gigabytes.

5 I/O Management Techniques

As pointed out in D7.2.1, the increasing data needs of scientific and engineering applications mean that the problems associated with reading, writing, analysing, storing and sharing large amounts of data are becoming more relevant to a wider user community within PRACE. Indeed, the increase of the I/O performance of HPC systems has not kept pace with compute capabilities. On top of this, users have not been able to squeeze as much performance from parallel file systems as they have from computational hardware. Moreover, on most HPC systems users get exclusive computing resources, whereas the file system is normally shared between all jobs. Many of the applications that have already been run on the PRACE infrastructure, are limited by I/O bottlenecks. Due to the increasing observational datasets from many scientific fields, these problems are not expected to be solved anytime in the near future.

Unfortunately, it is difficult to extract good performance from most current parallel I/O libraries for structured file formats. Libraries like PnetCDF and HDF5 reach only modest I/O bandwidths of 1-2Gb/s or even less in production codes, as shown in a recent PRACE white paper [81]. It is unclear how the performance can be increased significantly, due to the unknown impact of the multiple software and hardware layers between the API that the user sees down to the file system, writing the data to a storage medium.

In this section, we report on a project that has focused on exploiting a state-of-the-art I/O library, namely SIONlib, in order to enable the BONSAI astrophysics application for multi-petaflop/future exascale systems. In D7.2.1, the SIONlib library was identified as a potentially beneficial I/O tool on the road to exacale as it avoids potential performance bottlenecks by explicating the blocked nature of the filesystem. The SIONlib library is well suited for temporary files and situations without an established tool-chain for post-processing that expects certain file formats. Another advantage is that applications already using POSIX I/O calls don't need to be rewritten radically to parallelize their I/O. As noted in the introduction, the International Exascale Software Project roadmap expects check-pointing as a technique likely to continue on exascale systems. The SIONlib library seems ideally suited for this purpose as checkpoints are usually written in one go and resilience of the library is therefore less critical.

As for the other sections, here we provide a summary of the project along with a reference to the PRACE-3IP whitepaper associated with the project. We recommend that the reader also refers to the whitepaper for the project, which provides a more detailed report than is provided here.

5.1 Exploiting the SIONlib library for Fast, Parallel POSIX I/O in the Bonsai Astrophysics Code

WP165: *Using SIONlib for Multi-Petaflop/Exascale POSIX I/O in Bonsai*

Authors: J Donners (SURFsara), J. Bédorf (Leiden University)

Application: Bonsai

HPC Tool/Technique: SIONlib

Person Months: 2

Bonsai [82] is a gravitational N-body tree-code that runs completely on the GPU, which reduces the amount of time spent on communication with the CPU. The Bonsai code has been proven to scale up to nearly the full size of the Titan system at ORNL, which has 18,688 GPU nodes. The report here describes a project to modify the I/O of Bonsai, since it is a remaining

bottleneck: the creation of separate files for each MPI task overloads the Lustre metadata server and causes severe slowdowns. The SIONlib library [83] solves exactly this issue by writing to a shared file. The SIONlib library also circumvents the issues mentioned in the introduction by using the stripe size on the Lustre filesystem as a natural boundary in the file format. The SIONlib I/O performance can be seen as an upper limit for the I/O performance to a shared file.

The use of the SIONlib library on the Lustre filesystem of different PRACE systems was investigated. Several issues had to be resolved, both with the SIONlib library and the Liblustre API, before a satisfactory I/O performance could be achieved. First benchmarks on Cartesius, an Intel cluster with InfiniBand interconnect and Lustre file system with SIONlib 1.4p3 showed a considerable time for creating and opening a SIONlib file that was striped across all OSTs in ANSI mode. Also, the buffering for POSIX mode was incomplete. Both issues will be fixed in the next release of SIONlib. SIONlib reaches about 60% of the performance of a naive approach where each MPI task writes a separate file. However, the naive approach is certainly going to fail at exascale. If not technically, certainly many millions of files cannot be handled in a straightforward manner.

The first experiment tests if the SIONlib library can reach sufficient I/O bandwidth for the Bonsai application. The I/O needs to be fast enough to write all data to disk before the next I/O event of the running application. Each node runs 1 MPI task to perform the I/O.

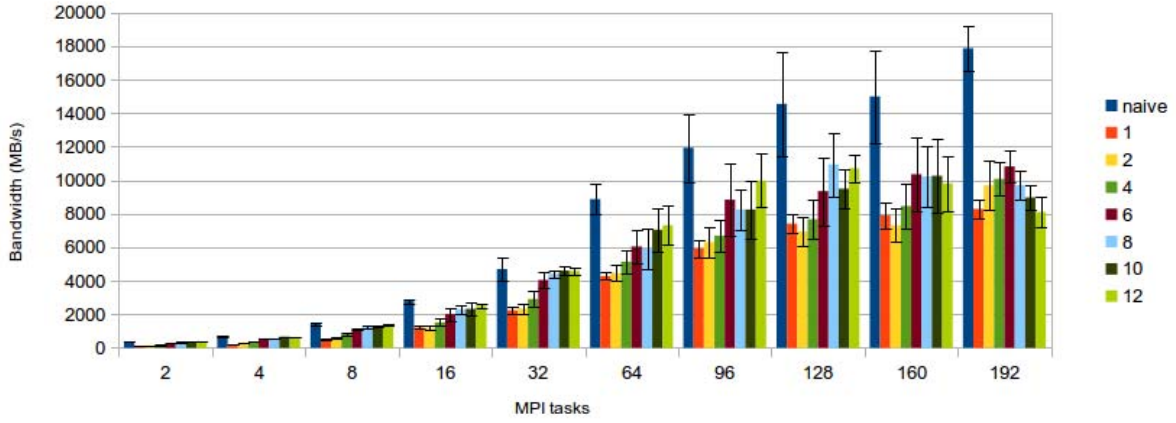

Figure 20: Bandwidth on Cartesius for the naive approach and SIONlib writing to one file using different strip sizes (in MB). All runs with one MPI task per node. Error bars indicate the standard deviation from eight measurements

Figure 20 shows the bandwidth on Cartesius for SIONlib and a separate file per MPI task ('naive approach'). Different lustre stripe sizes were used for SIONlib. It is clear that the naive approach offers the best write bandwidth for these modest MPI sizes. The optimal stripe size for SIONlib is around 6-8MB, with about 60% of the performance of the naive approach.

The second experiment tests if the SIONlib library can scale to over 10,000 MPI tasks writing to a single file. Since there is no supercomputer with over 10,000 nodes and a Lustre file system available within PRACE, these tests have been run with fully occupied nodes. The results show that the I/O performance can also be sustained at high MPI counts, with about 14GB/s for 16K MPI tasks on Hermit.

Reading of the output files is done for two main reasons: to restart the simulation and to analyze the output. This is not a problem, since the read bandwidth is easily exhausted by a handful of nodes. Neither any metadata updates, nor any locking is needed to read the file.

To conclude, the SIONlib library can reach a significant portion of the maximum I/O bandwidth and can sustain this even for files that are shared between more than 10,000 MPI tasks. The SIONlib library exhibits both the performance and the scalability that is needed to be successful at exascale.

Several PRACE systems were used for this project: the PRACE Tier-1 system Cartesius and the PRACE Tier-0 systems Curie and Hermit. Furthermore, some tests were run on the Titan system at ORNL.

6 Summary

This deliverable reports on the 17 projects undertaken as part of the exploitation phase of PRACE-3IP T7.2 'HPC Tools & Techniques', where the projects are aligned with four separate topics that we consider relevant to enable applications on current multi-petascale and future exascale systems, and where inspiration has been taken from the comprehensive survey of state-of-the-art HPC tools and techniques carried out during the first phase of T7.2, which was reported on in D7.2.1. In this section, we summarise our findings separately by topic: programming models, scalable libraries and algorithms, debuggers and profilers, and I/O management techniques. In-depth conclusions for each of the projects reported on in this deliverable can be found in associated whitepapers and so here we list only what we think are the most salient findings made during the exploitation phase.

Programming Models

- By exploiting MPI 2.0/3.0 one-sided communications, we have successfully enabled a subroutine in the CFD code, Code_Saturne, to improve its scalability on multi-petaflop and future exascale systems. For the small test cases investigated during the exploitation phase of T7.2, we found that MPI one-sided performance to be slightly better than two-sided point-to-point MPI communications. However, we expect that the impact of the work should be more greatly felt when employing Code_Saturne on larger problem types and higher node counts on the road to exascale. More significant performance improvements are expected when efficient and robust MPI 3.0 implementations become available soon. We have also demonstrated how MPI one-sided communications implemented during the exploitation phase are expected to improve memory usage, which is an important factor as MPI's internal memory usage is expected to become very large on future exascale machines.
- By exploiting MPI and OpenMP as well as more novel programming models such as OpenACC and OpenCL, we have started preparing the materials science code, CP2K, for exascale. During the early stages of the exploitation phase, we focused on enabling CP2K on the new Intel Xeon Phi platform for the first time. The Xeon Phi's MIC architecture is expected to be a relevant architecture on the road to exascale and many development projects around Europe are currently investing effort in enabling codes for future large-scale Xeon Phi-based clusters, which have the benefit of being exploitable with mature open standards such as MPI and OpenMP. However, needless to say, GPGPU architectures are still very much in the game as a component architecture on future exascale systems, and as part of CP2K-focused enablement effort, we have also exploited OpenACC and OpenCL to enable the DBCSR library in CP2K on a potentially wider range of computing resources, to bring the overall application closer to exascale. In particular, by exploiting OpenACC we have been able to obtain impressive performance relative to the corresponding lower-level CUDA implementation of the library without many modifications to the source code, demonstrating the potential of the relatively young standard on the road to exascale.

- By exploiting OpenACC, we have enabled a novel Cellular Automata Library, which employs the Frisch-Hasslacher-Pomeau (FHP) model to simulate fluid flows using exact, purely Boolean arithmetic, without any round-off error. We investigated the problem of its efficient porting to GPGPU-based clusters. As part of the project we demonstrated that the almost linear weak scaling for both multi-GPU implementations within a node suggests that the FHP model can be successfully run on much larger clusters and is a prospective candidate for exascale computational fluid dynamics. We also showed that the strong scaling for the OpenACC approach faced technical issues with communication between GPUs. We expect that, as soon as more robust PGI compilers become available, the development of efficient multi-GPU applications in general will become much easier.
- By exploiting OmpSs, we have explored the task-driven approach to targeting both CPU and GPGPU devices in a single code, namely the shock hydrodynamics proxy application, LULESH, which features heavily as part of many exascale-focused projects in the US. As pointed out in D7.2.1, task-based parallelism may be key to obtaining performance on future heterogeneous platforms, but has been under-investigated in PRACE to date. However, for this particular application/model pair we found that the frequent synchronization of small tasks exposes little slack to take advantage of. Indeed, we have found that LULESH works best when task-spawning overhead is at a minimum, and so semi-static load balancing will probably offer a better way to exploit hybrid execution over dynamic schemes as a follow-up approach.
- By exploiting OpenACC, we have investigated the possibilities for enabling codes within the EC-Earth3 suite for future exascale systems. To gain a full understanding of climate change at a regional scale will require EC-Earth3 to be run at a much higher spatial resolution (T3999 ~5km) than is currently feasible. Although multi-decadal, multi-ensemble global climate simulations at this resolution are currently not possible, it is envisaged that the work outlined in this deliverable will provide climate scientists with valuable data for simulations planned for future exascale systems.
- We have investigated the effectiveness of OpenACC in enabling DL_POLY_4 and how the PGI implementation of the standard compares relative to the original CUDA port of DL_POLY. Although, there is still some way to go until OpenACC proves itself in terms of performance (e.g., OpenACC 2.0 still doesn't allow the developer to explicitly utilize the fast shared memory on NVIDIA GPUs), the standard is still in its infancy and the features it strives to offer, namely programmability, portability and maintainability make it a promising tool for enabling large-scale complex applications, such as DL_POLY_4, on the road to exascale.
- Finally, by exploiting OpenMP 4.0, we enabled a Neural Networks application on the Intel Xeon Phi for the first time, where very good scaling performance was achieved. The application area is new and very promising, and due to its almost embarrassingly parallel nature, it holds real promise in exploiting many-core technology on the road to exascale.

Scalable Libraries and Algorithms

- By implementing and exploiting a new distributed memory Poisson Solver library in DL_POLY_4, we have provided an alternative to the existing FFT library in DL_POLY_4, which for many problem cases results can potentially result in a significant improvement in scalability on large node counts due to reduced global communication. Global communication, in particular, is known to be a severe barrier when trying to scale across large core counts (see reports on FFT libraries in section 4.3 of D7.2.1) and many open questions still exist on how FFT libraries will scale on

future exascale systems. It is expected that when DL_POLY_4 allows for the use of OpenMP within its MPI DD framework, the scalability will be further improved.

- By implementing and exploiting a new mesh generation tool, PMSH, we have enabled massive mesh generation in OpenFOAM for the first time. In particular, meshes with 7 Billion elements have been generated in approximatley 20 minutes using a scalable mesh generator built on top of Netgen. We strongly expect that the tool will allow for exascale-enabled solvers to be used on complex geometry problems in the near future. Our future work will focus on advanced schemes for projecting vertices on geometric boundaries. As well as mesh generation, we have also demonstrated that mesh refinement as enabled during the exploitation phase in Code_Saturne, is also a possible option to create huge meshes (billions of cells) for future CDF and CSM exascale challenges.
- As part of the exploitation phase, we have also provided an in-depth analysis of open source solvers for CFD and CSM, where tools such as FFLOP and PETSc along with the wider challenges these tools face in solving complex multiscale multiphysics problems on the road to exascale have been discussed in detail in the PRACE-3IP whitepaper, WP157.
- By exploiting a new Algebraic Multi-Grid solver algorithm we have started preparing the lattice QCD code tmLQCD for the exascale era and have demonstrated that the AMG technique is faster than standard CG algorithm used in tmLQCD. The algorithm is expected to play an important role as we approach large-scale simulations with physical quark masses and large volumes. These large-scale simulations will require tens of thousands of cores or more. We have also enabled the tmLQCD code on the Intel Xeon Phi where more work is now needed to redesign key algorithms because of memory limitations. Overall, our study provides a first feasibility step towards using Multi-Grid solvers in QCD applications on future exscale machines.
- Finally, we have looked outside of traditional HPC to investigate the opportunities for exploiting paradigms that have been developed for different workloads, namely the MapReduce paradigm. Although there are open questions about what the MapReduce paradigm can offer in terms of performance, we are happy to report that its exploitation is already leading to stimulating conversations centred round fault tolerance as PRACE prepares for the exascale era.

Debuggers and Profilers

- We have exploited the state-of-the-art profiler TAU to provide a detailed analysis of the performance challenges facing Code_Saturne on the road to exascale and have also used the tool in combination with a static auto-tuning tool, namely Orio, to investigate the potential benefits of auto-tuning on the road to exascale. The tools and techniques show potential in guiding the application developer to better performance in a heterogeneous run-time environment. As these tools mature, they will probably prove indispensable in dealing with the complexity that a heterogonous compute environment represents.
- During the exploitation phase, we have also analysed the performance of a Tier-0 code (Alya) using Extrae and Paraver for collecting and viewing the performance data. We have verified the powerful options provided by Extrae and Paraver: detail, insight and parallel behaviour and have identified real issues that tool developers will need to confront at extreme scale. In particular, we have highlighted that when generating trace files for a very large amount of cores (10,000 cores or above), we may have some difficulties in merging the performance data into one single trace file, or even managing a trace file of several gigabytes.

I/O Management Techniques

- We have exploited SIONlib to prepare the BONSAI astrophysics application for future multi-petascale/exascale systems and have demonstrated that the library reaches a write performance of about 15-20GB/s on several PRACE systems: Cartesius, Curie and Hermit. SIONlib combines a simple API with a file format that is designed to give maximum performance on parallel filesystems and we have demonstrated that the library scales on up to at least 16K cores writing to the same file. I/O is already a huge bottleneck on today's petascale systems and is expected to become a much bigger challenge on the road to exascale.