SEVENTH FRAMEWORK PROGRAMME

Research Infrastructures

INFRA-2010-2.3.1 – First Implementation Phase of the European High Performance Computing (HPC) service PRACE

PRACE-1IP

PRACE First Implementation Project

Grant Agreement Number: RI-261557

D7.5

HPC Programming Techniques

Final

Version: 1.0

Author(s): Cevdet Aykanat (Bilkent),
Antun Balaz (IPB),
Iris Christadler (LRZ),
Ivan Girotto (ICHEC),
Jose Gracia (HLRS),
Vladimir Slavnic (IPB),
Andy Sunderland (STFC),
Ata Türk (Bilkent)

Date: 30.04.2012

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-261557**	
	Project Title: PRACE First Implementation Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: < **D7.5**>	
	Deliverable Nature: <DOC_TYPE: Report / Other>	
	Deliverable Level: PU *	**Contractual Date of Delivery:** 30/April/2012
		Actual Date of Delivery: 30/April/2012
	EC Project Officer: Thomas Reibe	

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

Document Control Sheet

Document	**Title: <HPC Programming Techniques>**	
	ID: <D7.5>	
	Version: <1.0>	**Status:** Final
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2003	
	File(s): D7.5.docx	
Authorship	**Written by:**	Cevdet Aykanat (Bilkent), Antun Balaz (IPB), Iris Christadler (LRZ), Ivan Girotto (ICHEC), Jose Gracia (HLRS), Vladimir Slavnic (IPB), Andy Sunderland (STFC), Ata Türk (Bilkent)
	Reviewed by:	Claudio Gheller (CSCS) Florian Berberich (JUELICH)
	Approved by:	MB/TB

Contributors:

Aristotle Univ.: Paschalis Korosoglou, Christos Theodosiou

Bilkent: Seher Acer, Kadir Akbudak, Mustafa Korkmaz, Gündüz Vehbi Demirci

BSC: Judit Gimenez, Xavier Aguilar

CaSToRC: Giannis Koutsou, Alexei Strelchenko

CINECA: Fabio Affinito, Carlo Cavazzoni, Massimiliano Culpo, Andrew Emerson, Giovanni Erbacci, Marzia Rivi**,** Ivan Spisso

CSC: Jussi Enkovaara, Martti Louhivuori, Peter Råback

EPCC: David Henty

FMI, Finland: Sebastian von Alfthan

GRNET: Konstantinos Nikas

ICCS: Georgios Goumas, Vasileios Karakasis, Nectarios Koziris

ICHEC: Michael Lysaght, Peter Nash, Filippo Spiga, Rob Farber

IDRIS: Jean-Michel Dupays, Pierre-François Lavallée, Dimitri Lecas, Philippe Wautelet

IPB: Milos Nikolic, Josip Jakic, Petar Jovanovic, Aleksandar Jovic, Dusan Stankovic

JKU: Frederic-Gerald Morcos, Martin Polak, Volker Strumpen

KTH: Lilit Axner, Bogdan Frecus, Berk Hess, Erik Lindahl, Zilvinas Rinkevicius, Michael Schliephake, Olav Vahtras

LiU: Chandan Basu

LRZ: Volker Weinberg

NCSA: Krassimir Georgiev, D. Grancharov, N. Ilieva, N. Kosturski, E. Lilkova, I. Lirkov, L. Litov**,** Svetozar Margenov, P. Petkov, Y. Vutov

PSNC-WCNS: Marcin Gebarowski, Agnieszka Kwiecien, Mariusz Uchronski

SARA: John Donners, Marcin Zielinski

STFC: Thomas Keal, Stephen Pickles, Ilian Todorov

UiO: Trygve Helgaker, Maria Francesca Iozzi, Thomas Kjærgaard, Simen Reine, Ole Widar Saastad

Umeå Univ.: Mikael Rännar

UYBHM: Murat Manguoglu

VSB: Z. Dostal, V. Hapla, D. Horak, Petr Kovar, Tereza Kovarova, T. Kozubek, Dalibor Lukas, V. Vondrak, Jan Zapletal

Document Status Sheet

Version	Date	Status	Comments
0.1	07/March/2012	Draft	Formatting issues clarified
0.2	12/March/2012	Draft	Draft versions of chapters
0.3	19/March/2012	Draft	Review of chapter
0.4	7/April/2012	Draft	Final versions of chapters
0.5	11/April/2012	Draft	Version for internal review
0.6	17/April/2012	Draft	Review by Claudio Gheller
0.7	20/April/2012	Draft	Review by Florian Berberich
0.9	23/April/2012	Draft	Incorporation of review comments, updates by subtask leaders
1.0	25/April/2012	Final version	Version for approval by MB/TB

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, HPC, Programming Techniques

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] http://www.prace-project.eu

[2] PRACE Preparatory Access, http://www.prace-ri.eu/IMG/pdf/prace_preparatory_access_terms_of_reference_01032012.pdf

[3] Inauguration of CURIE, the second Tier-0 system, PRACE press release, http://www.prace-ri.eu/CURIE-Grand-Opening-on-March-1st-2012?lang=en

[4] More capacity to the PRACE Research Infrastructure: a Cray manufactured XE6 system called "HERMIT" at HLRS in use in autumn 2011, PRACE press release, http://www.prace-ri.eu/More-capacity-to-the-PRACE

[5] The Cilk Project, http://supertech.csail.mit.edu/cilk/

[6] Alan D. Simpson, Mark Bull, Jon Hill, "Identification and Categorisation of Applications and Initial Benchmarks Suite", PRACE Deliverable D6.1, http://www.prace-ri.eu/IMG/pdf/Identification_and_Categorisatio_of_Applications_and_Initial_Benchmark_Suite_final.pdf

[7] Phillip Colella, "The Landscape of Parallel Computing Research: A View From Berkeley", http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

[8] ScalaLife project, http://www.scalalife.eu/

[9] http://developer.nvidia.com/cuda-tools-ecosystem

[10] http://www.tacc.utexas.edu/news/press-releases/2011/stampede

[11] http://www.cray.com/Products/XK6/Specifications.aspx

[12] http://www.olcf.ornl.gov/titan/

[13] http://www.ncsa.illinois.edu/BlueWaters/

[14] http://www.green500.org/

[15] http://lattice.github.com/quda

[16] Clark, M. A. and Babich, R. and Barros, K. and Brower, R. C. and Rebbi, C, "Solving Lattice QCD systems of equations using mixed precision solvers on GPUs", Comput. Phys. Commun., V 181, 2010, p.1571

[17] https://github.com/lattice/quda/tree/quda-dwf-mpi

[18] P. Giannozzi and et al., "QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials", Journal of Physics: Condensed Matter, 21(39), 2009.

[19] http://qe-forge.org

[20] A. Dal Corso, "A pseudopotential plane waves program (pwscf) and some case studies", in Lecture Notes in Chemistry, Vol. 67, C. Pisani editor, Springer Verlag, Berlin (1996).

[21] F. Spiga and I. Girotto, "phiGEMM: a CPU-GPU library for porting Quantum ESPRESSO on hybrid systems", Proceeding of 20th Euromicro International Conference on Parallel, Distributed and Network-Based Computing -- Special Session on GPU Computing and Hybrid Computing, IEEE Computer Society, ISBN 978-0-7695-4633-9, pp. 368-375 (2012)

[22] M. Fatica, "Accelerating linpack with cuda on heterogenous clusters", In Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2009.

[23] http://speedit.vratis.com/

[24] http://code.google.com/p/cusp-library/

[25] http://www.symscape.com/gpu-openfoam

[26] http://www.symscape.com/files/articles/openfoam21x-windows/v1-mingw-openfoam-2-1-x.patch.gz

[27] Sean Rul, Hans Vandierendonck, Joris D'Haene and Koen De Bosschere, "An experimental study on performance portability of OpenCL kernels", Application Accelerators in High Performance Computing, 2010 Symposium, Papers (2010)

[28] Purnomo, Budirijanto and Rubin, Norman and Houston, Michael, "ATI Stream Profiler: a tool to optimize an OpenCL kernel on ATI Radeon GPUs", ACM SIGGRAPH 2010 Posters

[29] Kazuhiko Komatsu, Katsuto Sato, Yusuke Arai, Kentaro Koyama, H Takizawa, H Kobayashi, "Evaluating Performance and Portability of OpenCL Programs", Science And Technology, p. 781-784, 2010

[30] http://www.elsevierdirect.com/companions/9780123884268/MK_CUDA_Examples.zip

[31] Manuel Hasert, Harald Klimach, and Sabine Roller, "CAF versus MPI − applicability of coarray fortran to a flow solver", Published in: EuroMPI'11 Proceedings of the 18th European MPI Users' Group conference on recent advances in the message passing interface. Springer-Verlag Berlin, Heidelberg, 2011

List of Acronyms and Abbreviations

ACA	Adaptive Cross Approximation
ArBB	Array Building Blocks, programming language developed by Intel
BEM	Boundary Element Method
BETI	Boundary Element Tearing and Interconnecting
BLAS	Basic Linear Algebra Subprograms
CAF	Co-Array Fortran
ccNUMA	cache coherent NUMA
CEA	Commissariat à l'Energie Atomique (represented by GENCI, France)
CFD	Computational Fluid Dynamics
CG	Conjugant Gradient
CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CINES	Centre Informatique National de l'Enseignement Supérieur (represented in PRACE by GENCI, France)
CPU	Central Processing Unit
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CSR	Compressed Sparse Row, sparse matrix storage format
CSX	Compressed Sparse eXtended, sparse matrix storage format
CUDA	Compute Unified Device Architecture (NVIDIA)
CURIE	PRACE Tier-0 system, operated by GENCI, France
DBCSR	Distributed, Block, Compressed, Sparse, Row Library
DIC	Diagonal-based Incomplete Cholesky
DECI	Distributed European Computing Infrastructure
DEISA	Distributed European Infrastructure for Supercomputing Applications EU project by leading national HPC centres.
DFT	Density Functional Theory
DGEMM	Double precision General Matrix Multiply
DoW	Description of Work for PRACE-1IP
DP	Double Precision, usually 64-bit floating point numbers
EC	European Commission
EPCC	Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
ETHZ	Eidgenössische Technische Hochschule Zuerich, (Switzerland)
FETI	Finite Element Tiering and Interconnecting
FFT	Fast Fourier Transform
FMM	Fast Multipole Method

FP	Floating-Point
FPU	Floating-Point Unit
FTE	Full-Time Equivalent
GB	Giga (= 230 ~ 109) 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)
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 =109 periods or clock cycles per second
GigE	Gigabit Ethernet, also GbE
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPL	GNU Public Licence
GPU	Graphic Processing Unit
GTC	GPU Technology Conference
HERMIT	PRACE Tier-0 system, operated by HLRS, Germany
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
IDRIS	Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
IPB	Institute of Physics Belgrade (Serbia)
JUGENE	PRACE Tier-0 system, operated by JUELICH, Germany
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
LAPACK	Linear Algebra PACKage, software library for numerical linear algebra
LINPACK	Software library for Linear Algebra
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MIC	Many Integrated Cores, accelerator architecture from Intel
MPI	Message Passing Interface
MxM	Matrix-by-Matrix multiplication
NUMA	Non-Uniform Memory Access or Architecture

OpenCL	Open Computing Language
Open MP	Open Multi-Processing
OSKI	Optimized Sparse Kernel Interface
PCG	Preconditioned Conjugant Gradient
PCIe	Peripheral Component Interconnect express, also PCI-Express
PCI-X	Peripheral Component Interconnect eXtended
PGAS	Partitioned Global Address Space
PLX	PRACE Tier-1 system (with GPUs), operated by CINECA, Italy
PM	Person Month
PME	The Particle Mesh Ewald method used in GROMACS
POSIX	Portable OS Interface for Unix
PRACE	Partnership for Advanced Computing in Europe;
PRACE-1IP	PRACE First Implementation Project: project acronym
PRACE-2IP	PRACE Second Implementation Project; successor project
PRACE-PP	PRACE Preparatory Phase, predecessor project
PSNC	Poznan Supercomputing and Networking Centre (Poland)
QCD	Quantum Chromodynamics
QMM	Quantum Mechanical Model
RNG	Random Number Generator
SARA	Stichting Academisch Rekencentrum Amsterdam (Netherlands)
SLURM	Simple Linux Utility for Resource Management
SMP	Symmetric MultiProcessing
SMT	Simultaneous multithreading
SNIC	Swedish National Infrastructure for Computing (Sweden)
SP	Single Precision, usually 32-bit floating point numbers
SpMxV	Sparse Matrix-Vector-Multiplication
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
TB	Tera (= 2^{40} ~ 10^{12}) Bytes (= 8 bits), also TByte
TFlop/s	Tera (= 10^{12}) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
UPC	Unified Parallel C
WP	PRACE-1IP Work Package

Executive Summary

Work Package 7 “Enabling Petascale Applications: Efficient Use of Tier-0 Systems” (WP7) ensures the effective exploitation of the PRACE Tier-0 systems by increasing scalability and performance of applications. Codes are either successful applicants for the preparatory access Type C calls (Task 7.1 “Applications Enabling for Capability Science”) or they are part of an established collaboration between PRACE and application communities (Task 7.2 “Applications Enabling with Communities”). While the focus in WP7 is on enabling applications for Tier-0 systems, the tasks should also benefit application performance on Tier-1 systems.

Task 7.5 is called “Programming Techniques for High Performance Applications”. This task worked with users to implement new programming techniques, paradigms and algorithms for Tier-1 and Tier-0 systems, which have the potential to facilitate significant improvements in their applications performance. The task worked in close collaboration with Task 7.1 and Task 7.2 to ensure that the research communities benefit. Task 7.5 worked on five different areas to increase performance and scalability of user codes:

- Scalable algorithms
- Scalable libraries
- Multi-core/many core systems
- Accelerators
- Novel HPC languages

Overall, more than 30 projects were carried out by 25 different partners and third parties. They ranged from the introduction of new algorithms for sparse matrix operations to the assessment of new languages like StarSs, Chapel, Cilk and ArBB; from the comparison of mathematical libraries to the hybridization of important user codes to test the mixed OpenMP and MPI programming model. Several projects were dedicated to porting applications to GPUs, one outcome of this activity is a freely available Quantum Espresso CUDA port.

Task 7.5 covered a plethora of different approaches and codes. The following deliverable is a summary of all projects performed within Task 7.5. It consists of chapters for each of the five topics, containing high-level summaries of all projects. In most of the summaries, links to PRACE white papers or scientific publications are given for those readers that are interested in more detailed information. Each chapter contains a short introduction highlighting the different projects and a summary with conclusions and outlook.

1 Introduction

As stated in the Description of Work (DoW), the aim of Task 7.5 "Programming Techniques for HPC Applications" is to implement new programming techniques, paradigms and algorithms for Tier-0 systems which might have the potential to significantly improve application performance. The task cooperates with Task 7.1 "Application Enabling for Capability Science" and especially Task 7.2 "Application Enabling with Communities" to ensure that research communities and users benefit. Task 7.5 has nearly 300 PMs distributed across 18 countries and 26 partners or third parties, used for more than 30 individual projects. The deliverable gives an overview of these projects and tries to identify best-practices from the gained experience and results. The deliverable itself is quite concise in order to allow people to easily identify the projects that are of particular interest for them and to encourage further reading in the accompanying white papers or the referred publications.

1.1 Application enabling crucial for Petascale era

PRACE-1IP work on applications is all bundled in WP7 "Enabling Petascale Applications: Efficient Use of Tier-0 Systems", by far the biggest PRACE-1IP WP (it consists of ~950 person months (PMs), which equals 40% of the project's staff budget). These numbers illustrate how crucial application enabling has become in the Petascale era. Efficient use of Tier-0 and Tier-1 systems requires that most application codes are able to scale to several thousand cores. But the reality is that currently, still many widely used application codes scale typically to less than 4k cores, sometimes only to several hundred cores.

So it is obvious that in order to fully utilize today's Multi-Petascale systems and in order to proceed to Exascale computing we can't rely on incremental improvements but should consider not only programming paradigms and mathematical libraries but also the widely used numerical algorithms. Task 7.5 is the task where advanced –potentially disruptive- ideas could be developed, improved and tested against real production codes. Figure 1 illustrates how the different tasks of WP7 interact with each other and the user communities.

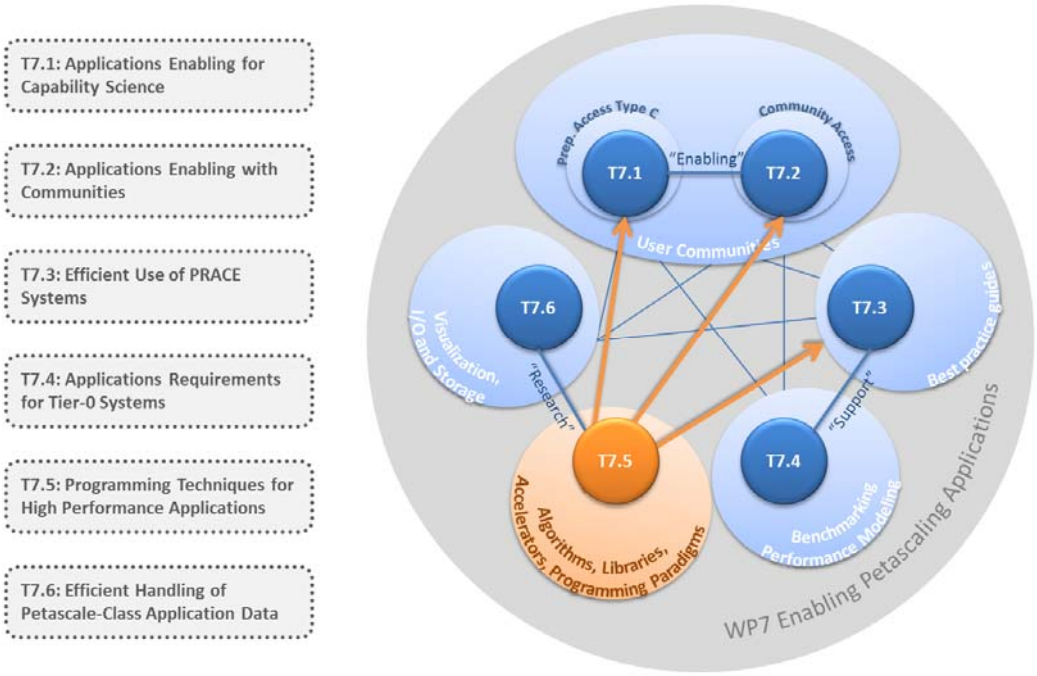


Figure 1: Overview of different tasks in WP7

Deliverable D7.5 is intended to be a "best practice guide on new HPC programming techniques and paradigms" (see DoW). It provides a kaleidoscopic vision of the 30+ individual projects carried on in the task and summarizes how the results could be transferred to other codes or communities; if they are widely applicable or under which circumstances it is worthwhile testing them on scientific applications. Where possible, different approaches are compared against each other to ensure that the readers can decide which of them are useful for their scientific domain/application code and to enable them to estimate implementation effort and potential benefits of new approaches.

More than 400 PMs are allocated in Task 7.5, 18 of the 20 countries participating in PRACE-1IP are engaged in Task 7.5. Effort is split amongst 26 third parties. While this would give an average value of 16 PMs per partner, the mean is 9 PMs. Only 6 partners have more than 0.5 full-time equivalents (FTE) allocated to Task 7.5: LRZ (15 PMs, task leader), ICHEC (17 PMs, subtask leader), Bilkent (17 PMs, subtask leader), IPB (21 PMs, subtask leader), JKU (24 PMs), VSB (37 PMs) and NCSA (42 PMs). More details on the effort available can be found in Table 1. More than 30 projects have been accomplished; several of those projects are collaborations among different partners. In effect, that means that most of the projects have been carried out with 4-10 PMs while a few projects were really big (e.g. from JKU, VSB and NCSA).

Country	Partner	Funded PMs	Unfunded PMs
Germany	LRZ	15	
	HLRS	6	
France	CNRS	11	
UK	EPCC	3	
	STFC	8	
Spain	BSC	4	
Finland	CSC	5	
Netherlands	SARA	12	
	CIT-RUG	4	
Austria	JKU	18	6
Sweden	KTH	1	4
	LiU	2	1
	UmU	4	1
Italy	CINECA	10	2
Poland	PSNC	4	
Greece	GRNET	2	2
	AUTH	1	
	ICCS	4	2
Portugal	UC-LCA	9	

Country	Partner	Funded PMs	Unfunded PMs
Ireland	ICHEC	17	
Turkey	UYBHM-ITU	6	6
	Bilkent	17	17
Cyprus	CASTORC	6	
Bulgaria	NCSA	21	21
Czech Republic	VSB	37	
Serbia	IPB	21	
	Sum	240	36

Table 1: Overview of efforts per partner

1.2 Organization of work

In order to structure work in WP7, the first goal for all task leaders was to define work plans, including a proposal for several subtasks. Figure 2 gives a schematic overview of the subtasks in Task 7.5; it also illustrates how major topics like new algorithms, scalable libraries, new programming paradigms, GPGPUs, multi-core programming and PGAS languages were distributed across subtasks (ST).

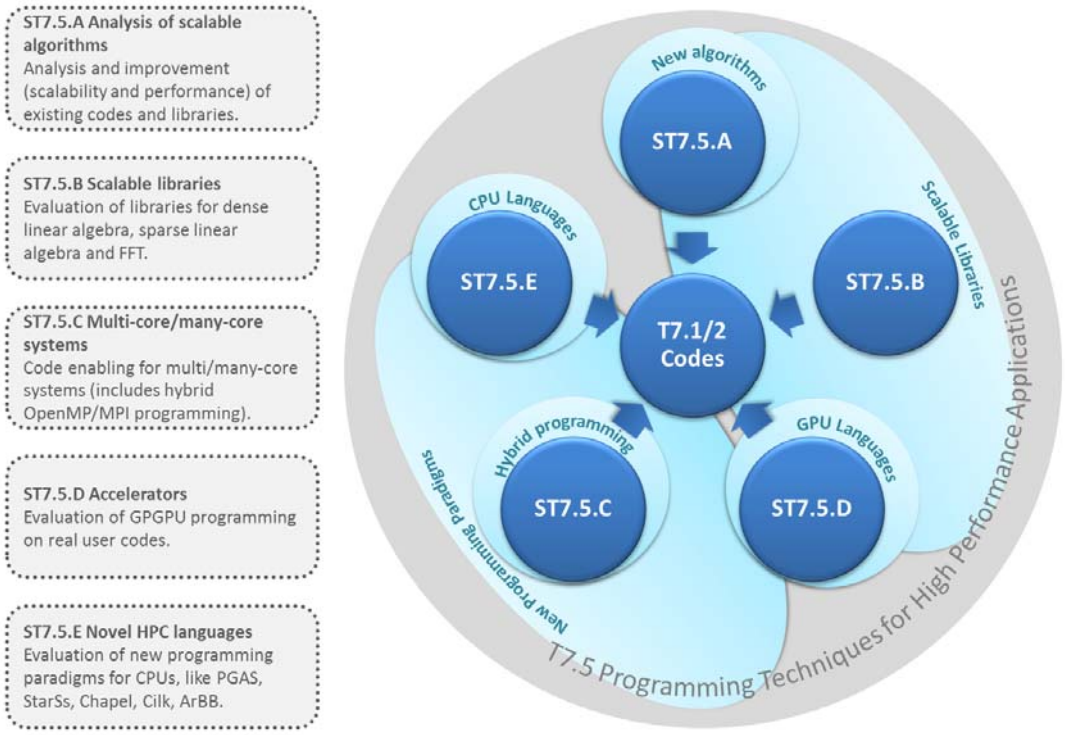


Figure 2: Schematic overview of subtasks in Task 7.5

During the PRACE-1IP kick-off meeting in September 2010 at LRZ, subtask leaders have been identified. The subtask leaders and co-leaders are:

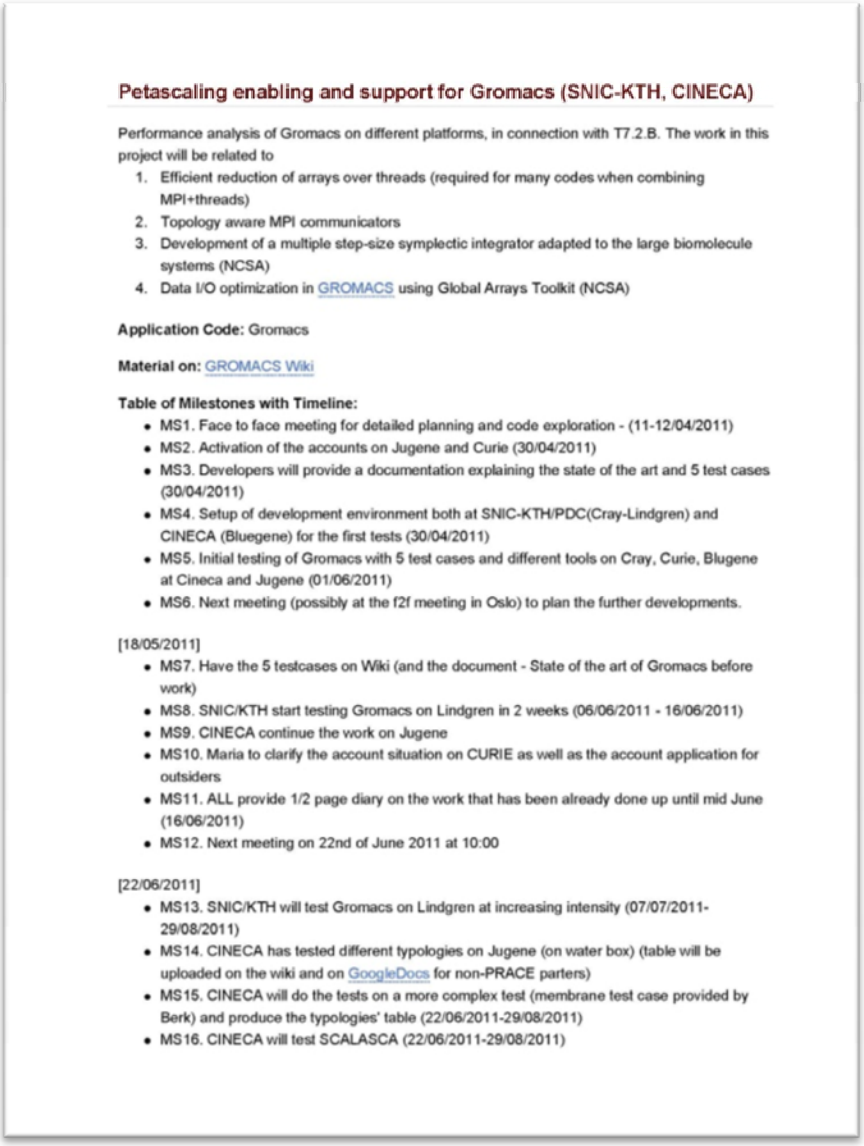
- Subtask 7.5.A: Cevdet Aykanat and Ata Türk (Bilkent)
- Subtask 7.5.B: Andy Sunderland and Stephen Pickles (STFC)
- Subtask 7.5.C: Antun Balaz and Vladimir Slavnic (IPB)
- Subtask 7.5.D: Ivan Girotto (ICHEC)
- Subtask 7.5.E: Jose Gracia (HLRS)

One task of WP7 is the support of preparatory access type C proposals under Task 7.1 "Applications Enabling for Capability Science". The PRACE call for preparatory access proposals is run by the PRACE AISBL and is a contiguous call, with cut-off dates every three months. Type C is meant for code development and optimization projects that require support of PRACE experts to improve the application code [2]. Since it could not be a priori known which projects will be submitted and granted preparatory access it was necessary to get an overview of HPC knowledge available within WP7. A survey was carried out to inquiry knowledge of HPC applications, research domains, high end systems and programming languages. The resulting matrix was internally referred to as "list of experts". Concerning the support of Task 7.1 applications or Task 7.2 communities, it was agreed that both a bottom-up and a top-down procedure will be used. That means that either Task 7.5 members interested in providing enabling/optimization work could volunteer to take over Task 7.1/2 duties (bottom-up), or the Task 7.1/2 leaders ("principle investigators") could contact the Task 7.5 subtask leaders to identify with the help of the "list of experts" individuals that would be suitable for the task (top-down). Except from one or two projects, where not enough knowledge and/or manpower were available within Task 7.5, all projects could be assigned; two-thirds of the Task 7.5 projects are actually supporting either Task 7.1 or Task 7.2 codes.

1.3 Overview of Task 7.5 projects

During the ramp up phase, in which people were waiting for first preparatory access projects, and for the final selection of Task 7.2 codes, most Task 7.5 partners started with so-called "up-front research projects" to test, develop and improve their ideas. Results of this phase were than incorporated in widely used scientific applications. To allow an easy overview of all projects, to ensure that progress of all projects could be easily monitored and to allow lively collaborations, dedicated wiki pages have been set up and augmented with regular status reports. Figure 3 shows an example screen shot of one of the wiki pages.

The following Mind-map (Figure 4) lays out the different projects within Task 7.5. It is an overview of all projects that have been started; most of them provided summary reports on their work for this deliverable. Some of the titles have been changed during the course of the project.



Petascaling enabling and support for Gromacs (SNIC-KTH, CINECA)

Performance analysis of Gromacs on different platforms, in connection with T7.2.B. The work in this project will be related to

1. Efficient reduction of arrays over threads (required for many codes when combining MPI+threads)
2. Topology aware MPI communicators
3. Development of a multiple step-size symplectic integrator adapted to the large biomolecule systems (NCSA)
4. Data I/O optimization in GROMACS using Global Arrays Toolkit (NCSA)

Application Code: Gromacs

Material on: GROMACS Wiki

Table of Milestones with Timeline:

- MS1. Face to face meeting for detailed planning and code exploration - (11-12/04/2011)
- MS2. Activation of the accounts on Jugene and Curie (30/04/2011)
- MS3. Developers will provide a documentation explaining the state of the art and 5 test cases (30/04/2011)
- MS4. Setup of development environment both at SNIC-KTH/PDC(Cray-Lindgren) and CINECA (Bluegene) for the first tests (30/04/2011)
- MS5. Initial testing of Gromacs with 5 test cases and different tools on Cray, Curie, Blugene at Cineca and Jugene (01/06/2011)
- MS6. Next meeting (possibly at the f2f meeting in Oslo) to plan the further developments.

[18/05/2011]

- MS7. Have the 5 testcases on Wiki (and the document - State of the art of Gromacs before work)
- MS8. SNIC/KTH start testing Gromacs on Lindgren in 2 weeks (06/06/2011 - 16/06/2011)
- MS9. CINECA continue the work on Jugene
- MS10. Maria to clarify the account situation on CURIE as well as the account application for outsiders
- MS11. ALL provide 1/2 page diary on the work that has been already done up until mid June (16/06/2011)
- MS12. Next meeting on 22nd of June 2011 at 10:00

[22/06/2011]

- MS13. SNIC/KTH will test Gromacs on Lindgren at increasing intensity (07/07/2011-29/08/2011)
- MS14. CINECA has tested different typologies on Jugene (on water box) (table will be uploaded on the wiki and on GoogleDocs for non-PRACE parters)
- MS15. CINECA will do the tests on a more complex test (membrane test case provided by Berk) and produce the typologies' table (22/06/2011-29/08/2011)
- MS16. CINECA will test SCALASCA (22/06/2011-29/08/2011)

Figure 3: Example of a wiki page

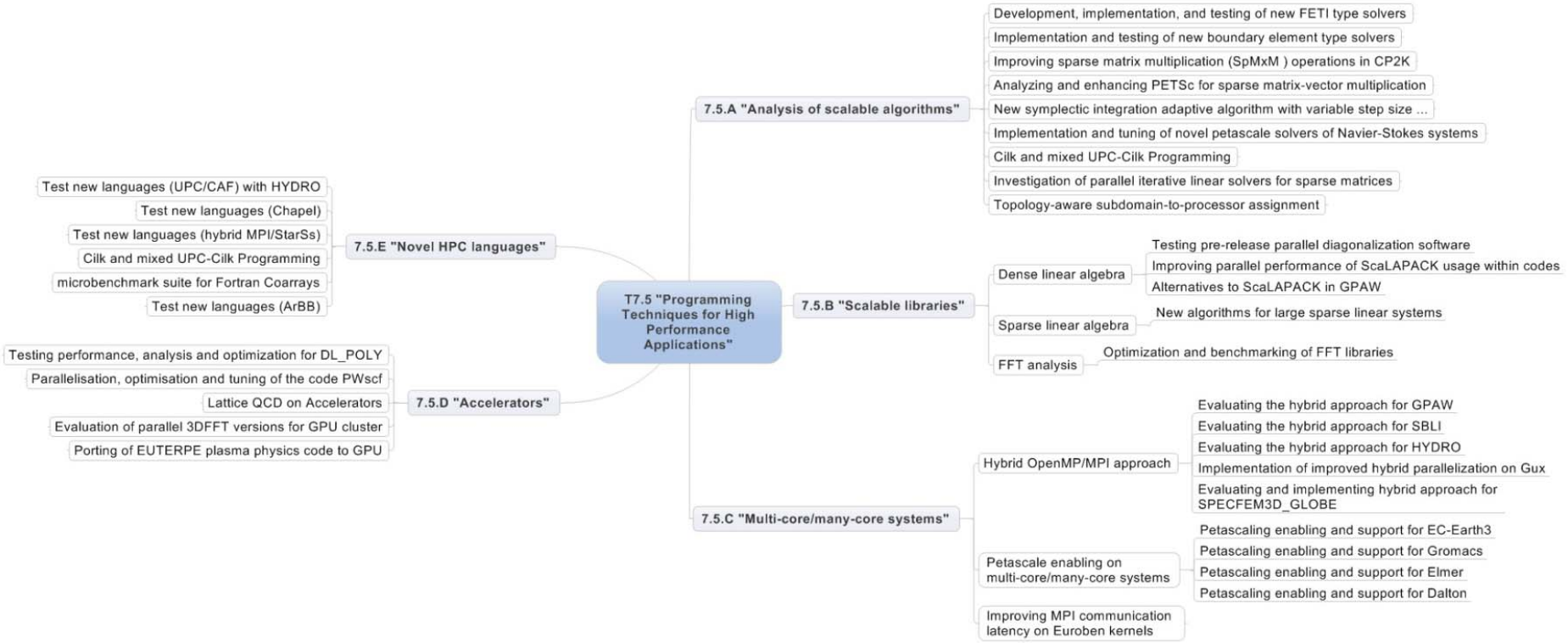


Figure 4: Mind-map of Task 7.5 projects

Many Task 7.5 projects delivered support to Task 7.1 and Task 7.2 codes. Effectively, three Task 7.1 projects were supported (ELMER, CP2K and GUX) and 10 out of 11 Task 7.2 codes received support from Task 7.5. Figure 5 gives an overview of Task 7.2 Community Codes. More information on the selection process and the codes (including references) can be found in the PRACE-1IP deliverables D7.2.1 "Interim report on collaboration with communities" and D7.2.2 "Final report on collaboration with communities". Except from the Plasma Physics code EUTERPE, all projects received support from Task 7.5. It turned out to be easier to support Task 7.2 community codes, which were projects with a longer duration, than the typical Task 7.1 three months project coming in through preparatory access. The first Task 7.2 codes have been chosen end of 2010, which gave the possibility to work more than one year on those applications - a common and necessary time frame for major algorithmic changes. Another reason for the relatively low number of supported Task 7.1 projects is probably the unexpected low number of Type C preparatory access proposals.

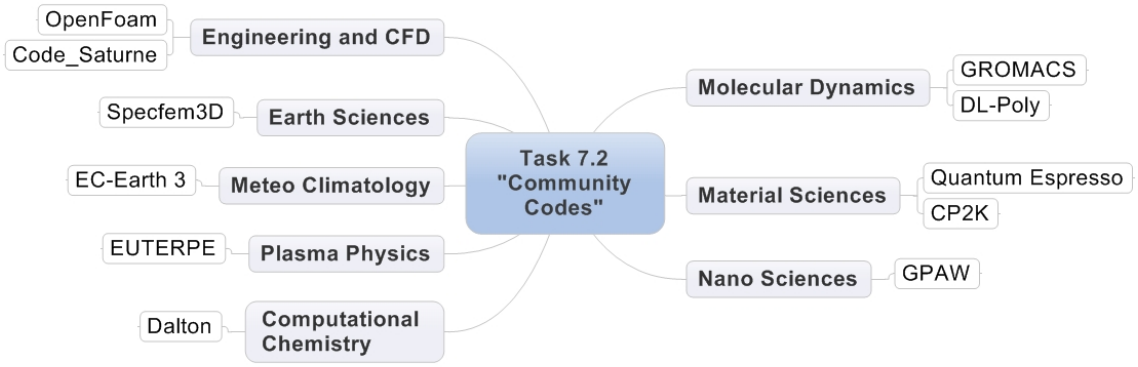


Figure 5: Mind-map of Task 7.2 Community Codes

1.4 Tier-0 Resources

By the time PRACE-1IP started only JUGENE, the first PRACE Tier-0 systems was available. CURIE was installed in two successive phases since the end of 2010 and is fully operational since March 2012 [3]. During this period, CURIE was made gradually accessible for research purposes through the PRACE regular and preparatory access calls. First accounts for Task 7.5 partners were granted in May 2011. Later, the system was upgraded and hybrid nodes with a total of 288 Nvidia M2090 "Fermi" GPUs were added. HERMIT, the third PRACE Tier-0 system became available only in autumn 2011 [4]. This was relatively late for PRACE-1IP and only a few Task 7.5 projects used HERMIT.

The total CPU resources available on Tier-0 for Task 7.5 were relatively modest: 1.300.000 core-hours on JUGENE and 380.000 core-hours on CURIE. Therefore, many of the projects used other machines, either local ones or Tier-1 systems and only ported their codes to Tier-0 later. While the second Tier-0 system CURIE now being equipped with GPUs, most development was done on the GPU cluster PLX in CINECA who generously offered in-kind CPU hours for testing. The fact that GPUs became available only later in the project is probably the reason for the relatively small amount of projects in the accelerator subtask. The PRACE-2IP work package WP8 "Community Code Scaling" with nearly 500 PMs has a focus on GPU enabling for Tier-0 systems.

1.5 Structure and Highlights

The deliverable is organized as follows: Results from each subtask are given in the following chapters. Each chapter contains an introduction and a conclusion section together with short summaries of all major projects within the respective subtask. For smaller projects, the full set of results is given in the summary. For medium to bigger projects, a high-level summary is given together with a pointer to either a scientific publication or a white paper. Overall, Task 7.5 produced 13 white papers and contributed to 8 Task 7.2 white papers. By the time this deliverable will be published the white papers will be available on the PRACE RI website [1].

Since this report and the accompanying white papers should serve as a best practice guide for (Tier-1 and) Tier-0 users, we tried to make the deliverable as concise as possible to allow the reader to easily identify the projects that are of interest. If the one-page summary aroused interest in the reader, the reader is referred to the white papers and publications. Last but not least, interested readers are encouraged to get in contact with the principle investigators if they have further questions.

Subtask 7.5.A (Chapter 2) assessed the impact that new algorithms can make on existing applications. This subtask looked at cache-oblivious algorithms as well as hybrid programming or effects of system topology on performance. Several projects focused on sparse matrix operations commonly used in many codes.

Subtask 7.5.B (Chapter 3) assessed, benchmarked and compared scalable libraries. The subtask focused on dense linear algebra, sparse linear algebra and FFT. Projects ranged from testing pre-released parallel diagonalization software, over benchmarking alternatives to ScaLAPACK, to a thorough assessment of available FFT libraries.

Subtask 7.5.C (Chapter 4) worked in two directions to improve performance on multi-core CPUs. Part of the projects focused on petascaling community codes, the others evaluated the hybrid programming model for multi-Petascale systems: MPI mixed with OpenMP. While some projects assessed performance of an existing hybrid implementation or on the improvement of the existing version, others developed a hybrid version from scratch.

Subtask 7.5.D (Chapter 5) investigated the use of performance gains achievable with the help of GPUs for codes like OpenFoam, Quantum Espresso, DL_POLY and QUDA. CUDA was the main programming environment; only the project on DL_POLY investigated the maturity and portability of OpenCL for a real HPC application. Subtask 7.5.D extended the QCD library QUDA by two additional fermion actions and ported PWscf to allow Quantum Espresso users to leverage the performance of GPUs. The latter project will be presented at GTC, the leading GPU Technology conference.

Subtask 7.5.E (Chapter 6) examined several different programming models for CPUs. Basic benchmark kernels like the Euroben kernels (mod2am/MxM, mod2as/SpMV and mod2f/FFT), a matrix transposition, a Jacobi and a Conjugent Gradient solver were ported or benchmarked with Chapel, Cilk and ArBB. Hydro, a widely used CFD benchmark in PRACE, was ported to UPC and Cilk (see also Subtask 7.5.A). A Lattice Boltzmann production code was very successfully ported to StarSs+MPI.

The deliverable is summarized by a conclusion and outlook in Chapter 7.

2 Scalable Algorithms

To enable petascaling of applications, parallelization schemes and algorithms that have been actively used in small-to-medium scale systems should be re-evaluated and possibly development of more scalable algorithms should be investigated. This chapter discusses support actions performed towards petascaling of current community application codes as well as diverse techniques, such as cache-oblivious algorithms, hybridization and topology-aware domain decomposition, which should be considered to enable petascaling of applications.

The first five sections are efforts towards identifying and resolving the performance bottlenecks in the petascaling of current community application codes such as ELMER, SPECFEM3D, CP2K, GROMACS, NAMD and OpenFOAM. Section 2.1 describes the efforts towards integrating an efficient massively parallel implementation of a Finite Element Tiering and Interconnecting (FETI) method into ELMER for the numerical solution of large linear systems arising in linearized engineering problems. Section 2.2 proposes to replace the Boundary Element Method (BEM) employed in SPECFEM3D for acoustic simulations with a new approach named Adaptive Cross Approximation BEM (ACA-BEM) and describes a new approach for the parallel implementation of ACA-BEM. Section 2.3 proposes to replace the Canon-based parallel sparse-matrix-matrix multiplication (SpMxM) scheme, which was reported to be the bottleneck in CP2K application, with a novel parallel SpMxM algorithm that localizes most of the multiplications so that the volume of communication during parallel SpMxM operation is reduced. Section 2.4 analyses the workload and communication distribution issues for GROMACS and NAMD on a BlueGene/P cluster, with the profiling tool SCALASCA and the GROMACS built-in tool g_tune_pme. Section 2.5 describes efficient algorithms for time-stepping, mesh refinement and parallel mapping for the solution of incompressible Navier-Stokes equations with the goal of providing these solutions as a library to be used in ELMER and OpenFOAM.

Irregular computations commonly encountered in today's scientific computing applications cause poor usage of CPU caches in today's deep memory hierarchy technology. Iterated computations with the same memory access patterns existent in these irregular computations can be exploited to achieve very high performance gains via restructuring and/or reordering of computations for respecting temporal and spatial localities properly. Section 2.6 proposes three row/column reordering algorithms for cache-oblivious sparse-matrix-vector multiply (SpMxV) operations. The SpMxV implementations based on these reordering algorithms are embedded into OSKI (BeBOP Optimized Sparse Kernel Interface Library) and an analysis of the trade-off between the reordering overhead and the performance gain is provided. Section 2.7, in an effort to improve the performance of HYDRO, proposes a restructuring of the two-dimensional 11-point stencil computations involved in HYDRO, which is a CFD code developed at IDRIS/CEA, to produce a cache oblivious program minimizing the number of data transfers between the levels of a memory hierarchy.

Hybrid architectures, in which nodes are connected through a network and each node consists of multiple cores, are commonly being used in petascale computing systems. The hybrid parallel computation model combines MPI and OpenMP for such architectures. Although pure MPI programming model can also be used for hybrid architectures, the combination of MPI and OpenMP is expected to yield better performance, since the data in the distributed and shared memories are both handled by the corresponding specialized frameworks by exploiting the strengths of both programming paradigms: the high scalability and rich functionality for process control in MPI, and the low communication overhead for small messages and fine-grain parallelism in OpenMP. Section 2.8 investigates hybridization efforts

for parallel SpMxV operations in two distinct frameworks and Section 2.9 gives guidelines for when to utilize hybrid schemes for a better performance in parallel SpMxV.

Petascale computing systems such as PRACE Tier-0 systems are built of hundreds of thousands of cores with large network diameters (between 20 to 60 hops for Blue Gene/P and XT5). It is a general belief that if virtual cut-through and wormhole routing is utilized, the message latency is independent of the distance in absence of blocking. However, even though this belief is valid for small-to-medium scale systems, recent studies indicate otherwise on petascale systems are reported and it is important that the PRACE community is aware of such studies. The aim of Section 2.10 is to guide application developers in identifying scenarios where the topology of the underlying system may affect performance and scalability.

2.1 Massively parallel implementation of FETI methods

Supported by: T. Kozubek (VSB), V. Vondrak (VSB), Z. Dostal (VSB), D. Horak (VSB), V. Hapla (VSB)

Whitepaper: T. Kozubeka, V. Vondrak, P. Raback, J. Ruokolainen, "*Relevant ingredients of the massively parallel implementation of FETI methods*", PRACE technical white paper

FETI (Finite Element Tearing and Interconnecting) type domain decomposition methods are powerful tools for constructing numerically and parallel scalable solvers. A FETI variant called Total FETI (TFETI) was developed at the Department of Applied Mathematics, VSB-Technical University of Ostrava, Czech Republic. The key idea is to apply the TFETI method on the numerical solution of the large linear systems arising in linearized engineering problems. Using this approach, a body is partitioned into non-overlapping subdomains, an elliptic problem with Neumann boundary conditions is defined for each subdomain, and intersubdomain field continuity is enforced via Lagrange multipliers. The Lagrange multipliers are evaluated by solving a relatively well-conditioned dual problem of a small size that may be efficiently solved by a suitable variant of the conjugate gradient algorithm. In TFETI, even the Dirichlet boundary conditions are enforced by Lagrange multipliers, which make the method very flexible.

The goal of this project was to develop and test an efficient massively parallel implementation of TFETI and integrate it into the ELMER multiphysical simulation software. The parallel implementation involves the effective regularization of singular subdomain stiffness matrices which improves conditioning and enables actions of their generalized inverses by using any standard method for matrix inversion, e.g. Cholesky decomposition. Another relevant ingredient is the effective parallel implementation of the coarse problem solution which is defined by an orthogonal projection onto the kernel of the large and rectangular matrix G arising from the existence condition in the TFETI algorithm. The projection itself consists in application of the projector $P = I - G^T(GG^T)^{-1}G$ on a vector. This operation strongly depends on whether and how G should be distributed and how action of $(GG^T)^{-1}$ should be implemented. The VSB team tested different strategies only for distribution of G into vertical blocks because the opposite distribution leads to enormous increase of communication cost. Tested strategies were the following: (i) iteratively using CG method performed by the master process, (ii) directly using Cholesky factorization performed by the master process, (iii) parallel application of explicitly computed inverse of GG^T using Cholesky factorization. The most promising seems to be strategy (iii).

The performance of the TFETI implementation in PETSc is demonstrated on a 2D elastostatic problem of the steel traverse. In Figure 6, the scalability behaviour is illustrated by

decomposing the computational domain into identical boxes and discretizing each box by 64800 triangles. The machine used for testing is the Tier-1 system Hector at EPCC.

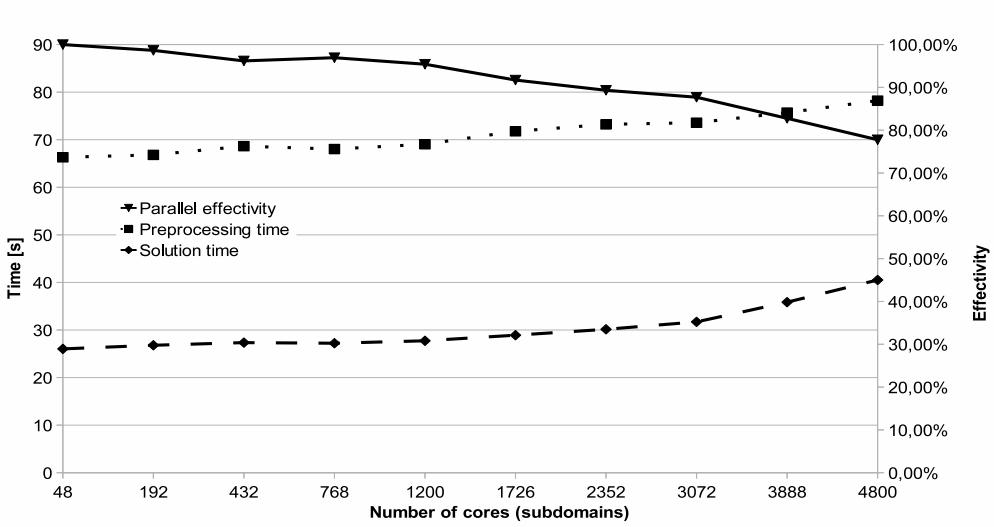


Figure 6: Scalability results for 2D linear elasticity benchmarks using the PETSc implementation (48 cores = 3 millions of unknowns and 4800 cores = 315 millions of unknowns).

On the other hand the performance of the TFETI implementation in ELMER multi-physical simulation software is demonstrated on an elastic three-dimensional problem defined on a box made of steel. The scalability behaviour is shown in Figure 7 by decomposing the computational domain into identical boxes and discretizing each box by 8000 bricks. The displayed weak-scaling computational times are achieved on the French Tier-0 system CURIE.

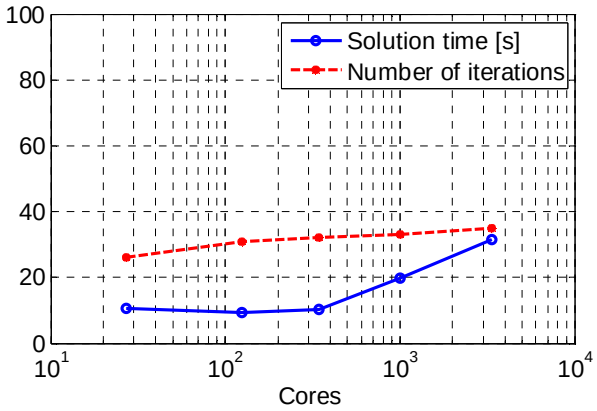


Figure 7: Weak-scalability results for 3D linear elasticity benchmark using the FETI implementation in ELMER.

In both examples, the times to solution are expected to remain in the same range since the number of unknowns and cores (subdomains) increase by the same factor. For large decompositions the above-mentioned coarse problem solution starts to dominate. Therefore the VSB team plans to replace the standard FETI method by its hybrid version, which is expected to eliminate this drawback. But even this standard FETI implementation in ELMER has demonstrated scalability for problems that were previously impossible to solve in parallel due to severe convergence problems. The current results look very promising.

The TFETI algorithm has been successfully applied also on nonlinear contact problems of engineering mechanics. In Figure 8, the computed total displacement with 1.6 million of unknowns of the yielding clamp connection of steel arched supports is depicted together with

the used domain decomposition. This type of construction is used to support the mining openings.

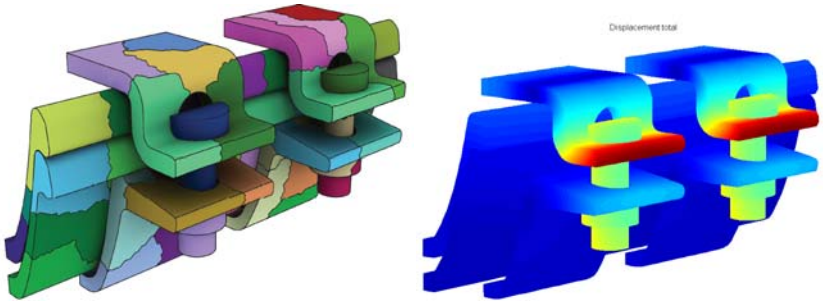


Figure 8: Domain decomposition (left) and computed total displacement (right).

2.2 Implementation and testing of new boundary element type solvers for SPECFEM3D

Supported by: Dalibor Lukas (VSB), Petr Kovar (VSB), Tereza Kovarova (VSB), Jan Zapletal (VSB)

Whitepaper: Dalibor Lukas, Petr Kovar, Tereza Kovarova, Jan Zapletal, "A Parallel Fast BEM on Distributed Memory Systems for the Helmholtz Equation as an Extension of SPECFEM3D"

SPECFEM3D is a parallel software for performing seismic simulations, e.g. earthquake simulations of the globe. The acoustical simulation relies on a Fourier transform of the seismic elastodynamic data, resulting from SPECFEM3D_GLOBE, which are then postprocessed by a sequence of solutions to Helmholtz equations, in the exterior of the globe. For the acoustic simulations the Boundary Element Method (BEM), which reduces computation to the sphere, has been employed. BEM is a powerful tool for constructing efficient scalable solvers for the numerical solution of elliptic boundary value problems. The main benefit of the application of BEM, as compared to the more popular FEM, is that the formulation of the problem is reduced to the boundary of the underlying domain which yields a significant dimension reduction. In particular, BEM is desirable, e.g., when dealing with large or unbounded domains or shape optimization problems. However, since BEM requires the explicit knowledge of a fundamental solution of a given partial differential operator, it is applicable only to the problems involving materials with rather simple properties. The method can be significantly improved by other fast techniques such as Adaptive Cross Approximation (ACA), Fast Multipole Method (FMM), and Boundary Element Tearing and Interconnecting (BETI), which accelerate the evaluation of the matrices and the consequent matrix–vector multiplication and lead to asymptotically nearly linear space and time complexities.

In the ACA-BEM approach the triangulation of the boundary into n triangles is hierarchically decomposed into clusters as depicted in Figure 9. The pairs of clusters are related to submatrices, which are classified as near or far-field, depending on a relation between the cluster sizes and their distance. While the near-field submatrices are kept fully-populated, the far-field submatrices are approximated by a low-rank format.

The method ACA subsequently chooses pivots in the residuum of the actual approximation of the far-field matrix, and then updates the approximation by a rank-1 matrix. It is a product of the pivoted row and column of the residual matrix. It can be viewed as an interpolation of the original far-field matrix to the pivot rows and columns. The approximation error is the spectral norm of the Schur complement. ACA in a combination with hierarchical matrices reduces the quadratic complexity to $O(n \log(n))$ time.

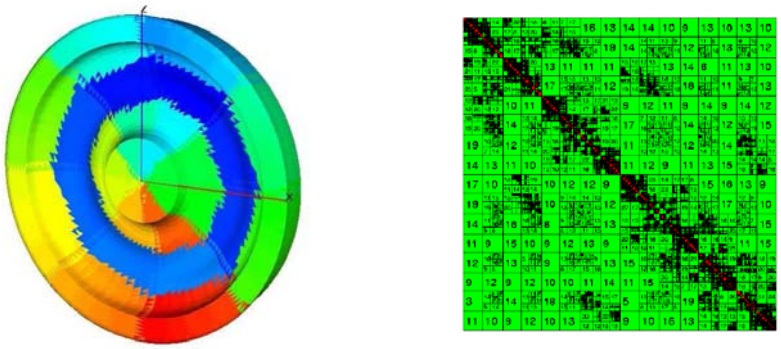


Figure 9: Hierarchical clustering of the geometry (left), related hierarchical matrix (right)

In this project, a new approach to the parallel implementation of ACA-BEM is proposed, which relies on subdivision of the boundary mesh into K pieces and a subsequent assignment of related blocks of the matrix to K concurrent processes by means of cyclic decomposition of complete graphs. The proposed approach respectively achieves $n/\mathrm{sqrt}(K)$ and $2n/\mathrm{sqrt}(K)$ triangles/process for diagonal and off-diagonal blocks of the hierarchical matrix. Moreover, the diagonal blocks are more expensive to assemble and store than those related to far-field off-diagonal blocks.

In the parallel implementation realized in this project, processors first concurrently read the numbers of boundary mesh parts from the prepared decomposition of the complete graph. Then they load the related parts from disk and start assembling of the related blocks of the system matrix, each compressed by means of hierarchical matrices and ACA. After the assembly, the processes are synchronized via waiting for a signal from master to contribute to the matrix-vector multiplication within GMRES iteration method for the solution of a linear system of equations.

Figure 10 demonstrates the parallel scalability of the matrix assembling (left) and the related memory requirement (right) per core of the proposed method for the single-layer matrix of the Laplace operator on a cube. In the legend, the level of discretization n and the corresponding matrix compression rate c are indicated. The compression rates do not depend on numbers of employed cores. The computed results support theoretical computational complexity of matrix assembling $O(n \log(n)/K)$ as well as expected memory complexity $O(n \log(n)/K + n/\mathrm{sqrt}(K))$. Note that a comparable FEM discretization of the largest problem would lead almost to a billion of volume unknowns.

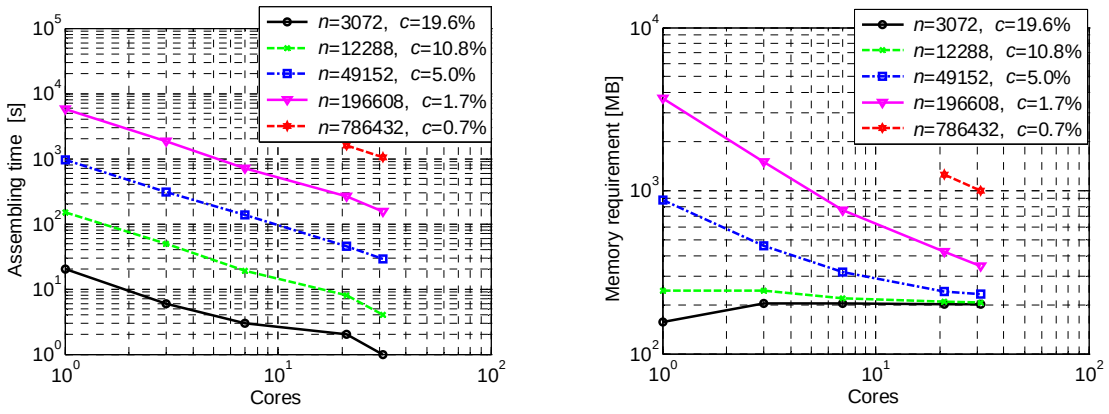


Figure 10: Parallel scalability of the matrix assembling (left) and the corresponding memory requirement (right) per core.

2.3 Improving sparse matrix multiplication (SpMxM) operations in CP2K

Supported by: Cevdet Aykanat (Bilkent), Kadir Akbudak (Bilkent)

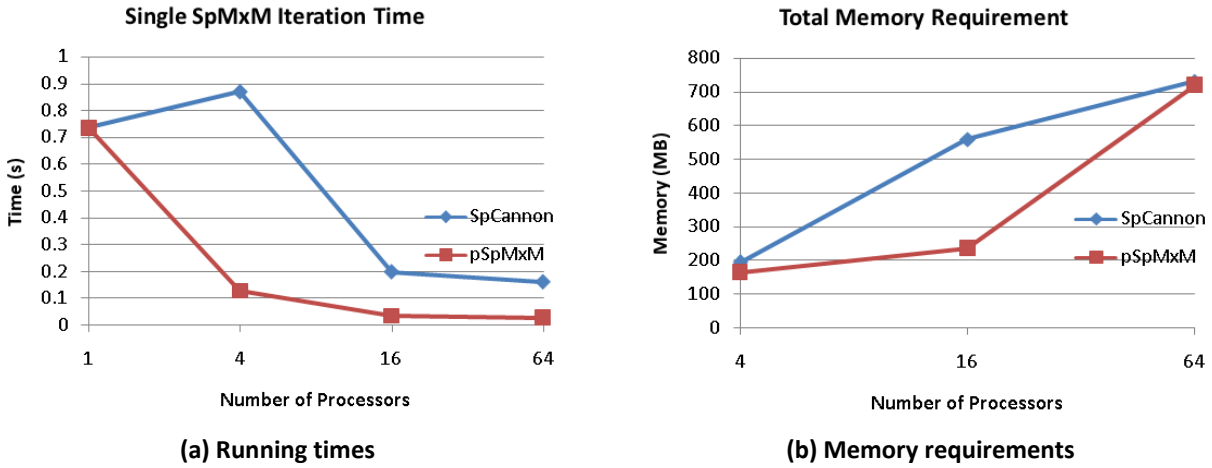
Collaborators: Ata Türk (Bilkent)

Multiplication of two sparse matrices (SpMxM) forms the computational core of many scientific applications. It is used in many application areas such as molecular dynamics, computational fluid dynamics, and climate simulation. CP2K is a state-of-the-art tool used in atomistic and molecular simulations of solid state, liquid, molecular and biological systems and it employs a new linear-scaling self-consistent field (SCF) method that performs SpMxM operations on the sparse density matrix. This method uses the parallel SpMxM library DBCSR. For large systems (e.g. 1,000,000 atoms), the dominant bottleneck in SCF computations is the SpMxM operation, which is currently implemented such that it utilizes Cannon's algorithm. Cannon's algorithm is a distributed algorithm for dense matrix multiplication operation on two-dimensional processor meshes. In this algorithm, the total memory requirement remains constant for dense matrix multiplication. This algorithm can also be used for sparse matrix multiplication. However, Cannon does not exploit the sparsity of matrices and performs extensive communication. Furthermore, it is not guaranteed to have uniform distribution of nonzeros, which may cause imbalance in computational loads of processors.

In this project a novel pre-processing step and a parallel SpMxM scheme is proposed and implemented such that it localizes most of the multiplications so that the volume of communication during parallel SpMxM operation is reduced. Since sparsity patterns of the two matrices multiplied in SCF iterations do not change for approximately 10 to 20 iterations and the same sparse matrices are multiplied to compute the sign of a matrix using Newton-Schulz iterations, the preprocessing step is expected to be amortized by the speed up in the SpMxM operation. Communication-overhead minimization can increase the scalability of CP2K.

For evaluation of the proposed parallel SpMxM code (pSpMxM), a school book implementation of Cannon's algorithm for sparse case (SpCannon) is implemented. Note that in this SpCannon implementation, total memory requirement increases with increasing number of processors due to the sparse storage of the matrices. For example, in *compressed storage by rows* (*CSR*) scheme, when number of processors (*P*) is quadrupled, size of *IROW* array is halved whereas sizes of *ICOL* and *NZ* arrays drop to one fourth of the previous sizes, in the case of uniform distribution of nonzeros.

Extensive comparisons of the pSpMxM and SpCannon methods on sign matrices dumped from Newton-Schulz iterations in CP2K are conducted. The CP2K benchmark named H2O-32-se-ls-7, which was provided by Iain Bethune from EPCC, was used. The H2O-32-se-ls-7 input file describes a periodically repeating box of 32 water molecules, where this box is repeated NREP times in each dimension. We obtained the sign matrix for NREP=9 and this matrix is filtered for a cut-off value of 10^{-6}. The obtained sign matrix was randomly permuted before being tested in SpCannon to obtain uniform distribution of nonzeros. The two codes were run on the CURIE supercomputing cluster (French Tier-0 PRACE system) for *P=4, 16,* and *64*processors. The parallel execution times and total memory requirements are presented in the following figures. As seen in Figure 11(a), pSpMxM runs at least five times faster than SpCannon. As seen in Figure 11(b), for up to 64 processors, the memory requirement of pSpMxM is lower than SpCannon. However, especially for higher number of processors, the memory requirement of pSpMxM is expected to be larger than SpCannon.



(a) Running times (b) Memory requirements

Figure 11: Running times and memory requirements of the SpCannon and pSpMxM methods.

2.4 Analysis of the symplectic integration algorithms for biomolecular simulations

Supported by: Leandar Litov (NCSA), D. Grancharov (NCSA), E. Lilkova (NCSA), N. Ilieva (NCSA), P. Petkov (NCSA)

Whitepaper: D. Grancharov, E. Lilkova, N. Ilieva, P. Petkov and L. Litova, "Analysis of symplectic integration algorithms with variable step size for petascale biomolecular simulations", PRACE technical white paper

This project studies the scalability and the work-load increase and distribution among the computing cores in the widely used MD simulation packages GROMACS (version 4.5.3) and NAMD (CVS from 19.02.2011), on the example of three test systems with increasing size ($5x10^5$,~10^6 and~$2.2x10^6$atoms respectively), with the profiling tool SCALASCA and also by means of the GROMACS built-in tool g_tune_pme. The stability and scalability of the existing integration algorithms with variable time-step implemented in NAMD and GROMACS are also analysed in order to identify the sources of the instabilities (different kinds of resonances). Based on these analyses, a symplectic time reversible integration algorithm specially designed for Petascale biomolecular MD simulations is under development.

These investigations were performed at the IBM BlueGene/P supercomputer of Bulgarian National Center for Supercomputing Applications (8192 computing cores) and on 32 computing cores of a local Linux cluster at the Faculty of Physics of the St. Kl. Ohridski University of Sofia. The main results of these investigations are summarized as follows:

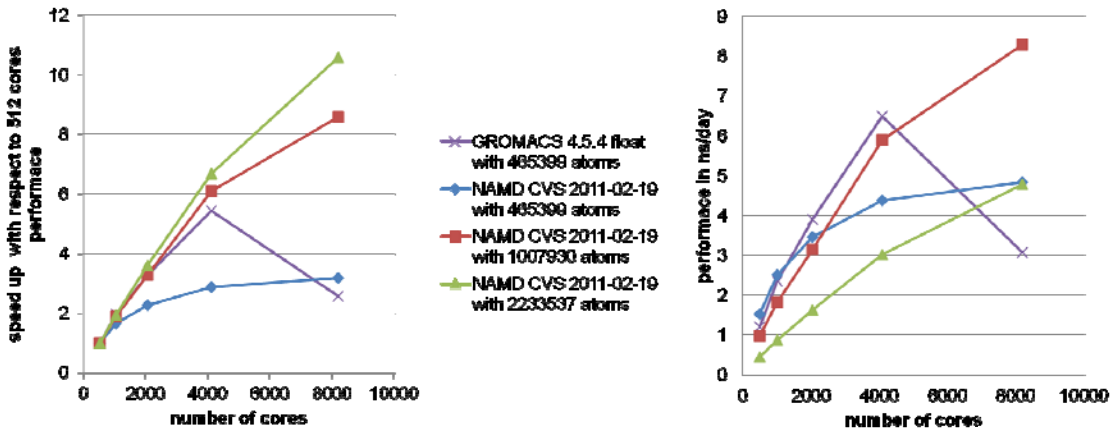


Figure 12: Speed-up (left) and performance (right) of GROMACS 4.5.4 and NAMD CVS 2011-02-19.

As seen in Figure 12, the scaling of NAMD improves with the increase of the system size, though with a slower growth beyond 4096 computing cores. For GROMACS, this number of cores appears to be critical, as it scales well only up to that point, but with higher overall performance than NAMD (see Figure 12 (right))

As seen in Table 2, up to 2048 cores, the three domain decomposition modes of GROMACS – interleave, pp_pme and Cartesian – have similar performance, with slight prevalence of the default mode – interleave. However, on 4096 cores this mode has the lowest performance, the other two performing better, with a negligible difference between them.

ddorder mode computing cores	Interleave [*ns/day*]	pp_pme [*ns/day*]	Cartesian [*ns/day*]
512	6.672	6.592	6.600
1024	12.122	11.905	11.973
2048	20.856	20.627	20.426
4096	27.994	31.306	31.544

Table 2: GROMACS performance in the three dd-order modes

Analysis of short simulations (2 ps) with fraction of pme cores 1/2, 3/8, 1/4, 1/8 and 1/16 of a test system of 460000 atoms on 512 to 4096 computing cores on the Bulgarian supercomputer with GROMACS 4.5.4 shows that the performance increases with the reduction of the number of pme cores up to 4096 cores where saturation is observed (Figure 14). Nevertheless, with the reduction of the pme only cores scalability drops, because of the increase in communication (Figure 14);

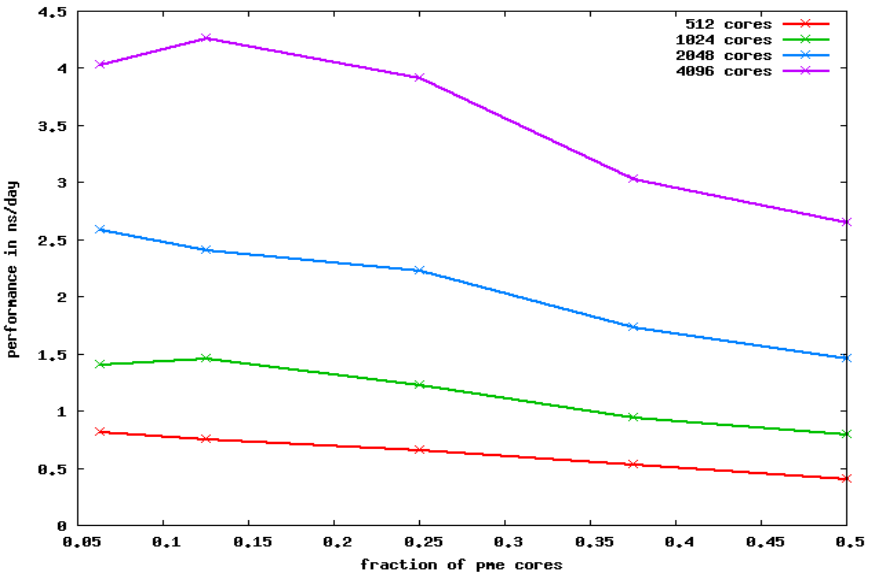


Figure 13: Performance of the Gromacs integrator as function of the number of pme only cores (shown as fraction from the total amount of cores).

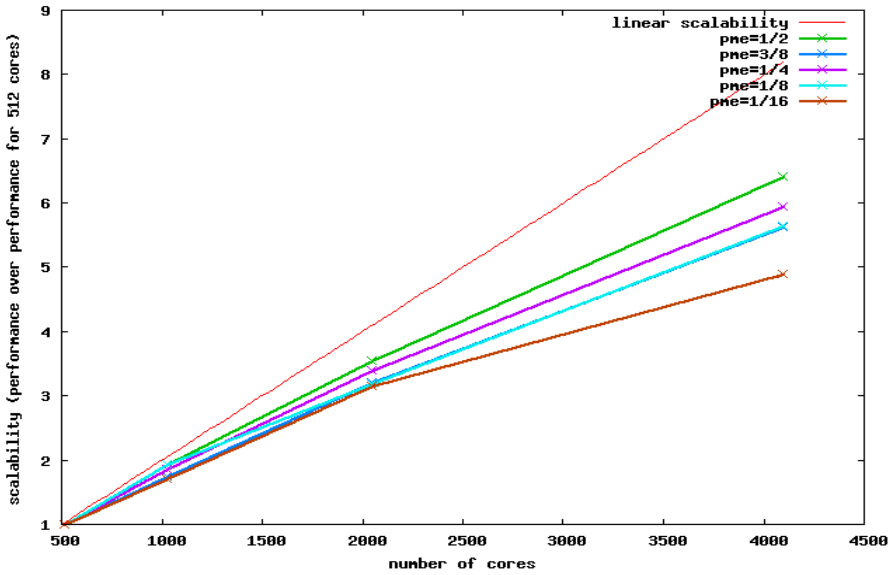


Figure 14: The scalability of the Gromacs integrator for different number of pme only cores (shown as a fraction from the total amount of cores).

GROMACS appears to be less suitable for implementation of variable step-size algorithms than NAMD, where an essential improvement of the performance is already achieved that way. Simulations with a system containing approx. 35000 atoms demonstrate that it is possible to speed up the calculations with 47,8 % using the VerletI/r-RESPA multiple timestep algorithm when calculating the short range electrostatic interactions on every timestep and the long range electrostatic interactions on every 6 timesteps (column 5 on Figure 15) with perfect energy conservation as shown on Figure 16. An even greater speed up might be achieved with the paremeters from the sixth column in Figure 15, but at the price of unstable simulation after 34 ns with conserved total energy though.

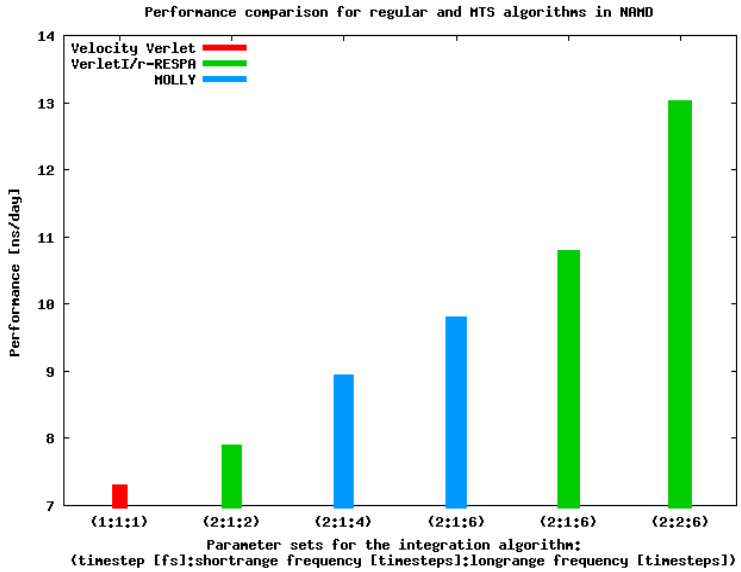


Figure 15: NAMD performance for different parameter sets (timestep [fs]: frequency of calculation of short range forces [number of timesteps]:frequency of calculation of long range forces [number of timesteps]) for different integration algorithms

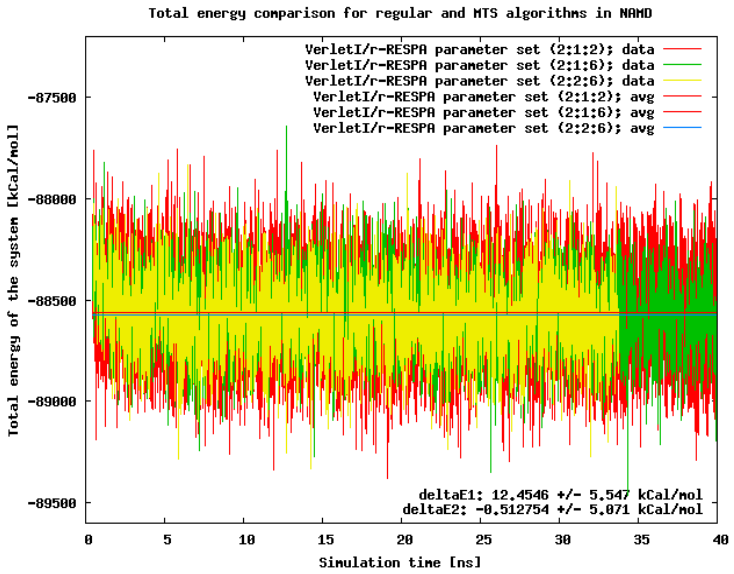


Figure 16: VerletI/r-RESPA integration algorithm shows perfect energy conservation for different parameter sets. Parameter set (2:2:6) leads to unstable simulation after 34 ns.

2.5 Parallel Solvers for Incompressible Navies-Stokes Equations

Supported by: Svetozar Margenov (NCSA), Krassimir Georgiev (NCSA), N. Kosturski (NCSA), I. Lirkov (NCSA), Y. Vutov (NCSA)

Whitepaper: K. Georgiev, N. Kosturski, I. Lirkov, S. Margenov, Y. Vutov, "Parallel Solvers for Incompressible Navies-Stokes Equations and Scalable Tools for FEM Applications", PRACE technical whitepaper

This project is focused on: (*a*) construction and analysis of novel scalable algorithms to enhance scientific applications based on mesh methods (mainly on finite element method (FEM) technology), and (*b*) optimization of a new class of algorithms on many core systems.

The activities were motivated by advanced large-scale simulations of turbulent flows in the atmosphere and in the ocean, simulation of multiphase flows in order to extract average statistics, solving subgrid problems as part of homogenization procedures. The computer model is based on implementation of a new class of parallel numerical methods and algorithms for time dependent problems. It only requires solution of tridiagonal linear systems and therefore it is computationally efficient, with optimal computational complexity of the same order as that of an explicit scheme, and yet, unconditionally stable. The scheme is particularly convenient for parallel implementation.

A novel contribution of this work is to avoid the matrix transposition which is usually used in alternating the directions in time stepping algorithms. New scalable algorithms and software for FEM simulations were implemented. This implementation considers voxel and unstructured meshes, stationary and time dependent problems, linear and nonlinear models. The focus of the implementation was towards development of scalable mesh methods and tuning of the related software tools attuned to the IBM Blue Gene/P architecture but other massively parallel computers and MPI clusters were taken into account too. Efficient algorithms for time stepping, mesh refinement and parallel mappings are implemented. The computational models address discrete problems in the range of $O(10^9)$ degrees of freedom in space. The related time stepping techniques and iterative solvers are targeted to meet the Tier-1 and (further) Tier-0 requirements.

The final goal of this project is to provide portable tools for integration in commonly accepted codes like Elmer and OpenFOAM. The developed software is organized as a computer library for the use of researchers dealing with solution of incompressible Navier-Stokes equations. Scalability results of the developed software on a 4096-core IBM Blue Gene/P system and 256-core Dell PowerEdge cluster are displayed in Table 3, where n_x, n_y and n_z are the numbers of the grid nodes correspondingly to *Ox*, *Oy* and *Oz* directions, the CPU times are measured in seconds by using the MPI function MPI_Wtime, the speed-up is defined by $S_p=T_1/T_p$, where T_p is the time for the execution of the algorithm using p processors (cores); the efficiency on p processors is defined as $E_p=S_p/p$.

n_x	n_y	n_z	Number of cores											
			2	4	8	16	32	64	128	256	512	1024	2048	4096
Dell PowerEdge cluster														
120	120	120	1.007	1.001	0.703	0.691	0.936	1.025	1.082	0.849				
120	120	240	0.975	0.759	0.484	0.483	0.709	0.939	0.902	0.827				
120	240	240	0.997	0.753	0.457	0.446	0.520	0.757	0.886	0.853				
240	240	240	0.967	0.757	0.430	0.477	0.545	0.795	0.766	0.888				
240	240	480	0.970	0.734	0.413	0.453	0.493	0.552	0.543	0.695				
240	480	480	1.002	0.715	0.424	0.441	0.485	0.520	0.500	0.552				
IBM Blue Gene/P computer														
120	120	120	1.055	1.096	1.143	0.881	1.117	1.092	1.014	0.900	0.851	0.581	0.420	0.267
120	120	240	1.014	1.064	1.094	1.154	0.861	1.108	1.038	0.922	0.910	0.631	0.439	0.340
120	240	240	1.008	1.015	1.052	1.171	1.151	0.874	1.061	0.977	0.943	0.672	0.557	0.435

Table 3: The parallel efficiency (Ep) achieved on an IBM Blue Gene/P computer and a Dell PowerEdge cluster for 3D Stokes problem.

The presented numerical tests on IBM Blue Gene/P (up to 1024 nodes (4096 cores)) confirm that the solvers meet the Tier-1 requirements. The analysis of the output results show that for the problems with big dimensions the efficiency achieved on IBM Blue Gene/p computer is much larger than this achieved on Dell PowerEdge cluster. The superlinear speed-up which can be seen on Blue Gene/P up to 128 cores is due to the better use of the cache memories of the processors (L3 cache is 8MB for IBM Blue Gene/P computer). The decreased efficiency which can be seen in some of runs shows that the use of the standard MPI communication subroutines is not very efficient between cores in one and the same node.

The developed scalable tools for FEM applications are incorporated and tested within the Elmer environment, which was for the first time successfully ported to the IBM Blue Gene/P architecture. We solved a stationary heat conduction equation on a cylindrical rod, composed of two materials. We applied *Dirichlet* boundary conditions on both ends of the rod. The mesh was generated with the automatic 3d tetrahedral mesh generator *NetGen*, then partitioned using the MPI-based parallel library *ParMETIS* and finally transformed to the Elmer mesh format. Tests with varying thermal conductivity jump between the two materials were performed. The advantages of the parallel AMG (Algebraic MultiGrid) compared to the more commonly used parallel incomplete factorization (ILU) are well expressed in the case of coefficient jumps and unstructured grids (see Table 4 for coefficient jump of 100). A set of comparative analysis for parallel scalability of Elmer on Blue Gene/P and a MPI cluster are also performed to illustrate how the problems with porting of Elmer are solved including the new developed tools for parallel mesh refinement.

P	*N*	Iterations ILU	Time in sec. ILU	Iterations BoomerAMG	Time in sec. BoomerAMG
128	1 678 336	569	58.8	20	38.1
128	13 426 688	999	563	22	321
256	1 678 336	624	42.2	20	30.2
256	13 426 688	519	287.2	23	243
512	1 678 336	886	43.6	20	34.2
512	13 426 688	2949	512	24	183
1024	1 678 336	889	65.8	20	45
1024	13 426 688	698	172.4	22	188
1024	107 413 504	728	1956	20	1461

Table 4: Number of iterations and CPU time for stationary heat conduction equation on a cylindrical rod with coefficient jump of 100 (P – number of processor nodes, N – number of unknowns)

2.6 Analyzing and enhancing OSKI for sparse matrix-vector multiplication

Supported by: Cevdet Aykanat (Bilkent), Kadir Akbudak (Bilkent)

Collaborators: Ata Türk (Bilkent)

Whitepaper link: Kadir Akbudak, Enver Kayaaslan, Cevdet Aykanat, "Analyzing and enhancing OSKI for sparse matrix-vector multiplication", PRACE technical white paper, http://arxiv.org/abs/1203.2739

Sparse matrix-vector multiplication (SpMxV) is an important kernel operation in iterative linear solvers used for the solution of large, sparse, linear systems of equations. In these iterative solvers, the SpMxV operation $y=Ax$ is repeatedly performed with the same large, irregularly sparse matrix A. Irregular access pattern during these repeated SpMxV operations causes poor usage of CPU caches in today's deep memory hierarchy technology. However, SpMxV operation has a potential to exhibit very high performance gains if temporal and spatial localities are respected and exploited properly. Here, temporal locality refers to the reuse of data words (e.g., x-vector entries) within relatively small time durations, whereas spatial locality refers to the use of data words (e.g., matrix nonzeros) within relatively close storage locations (e.g., in the same lines).

This work investigates two distinct frameworks for the SpMxV operation: single-SpMxV and multiple-SpMxV frameworks. In single-SpMxV, the y-vector results are computed by performing a single SpMxV operation $y = Ax$. In multiple-SpMxV, $y=Ax$ operation is computed as a sequence of multiple input- and output-dependent SpMxV operations, $y = y + A^k x$ for $k = 1,\ldots,K$, where $A = A^1+\ldots+A^K$. For single-SpMxV, two cache-size-aware row/column reordering methods based on top-down 1D and 2D partitioning of a given sparse matrix are proposed and implemented. The 1D-partitioning-based method (sHP_{CN}) relies on transforming a sparse matrix into a single-bordered block-diagonal form by utilizing the column-net hypergraph model. The 2D-partitioning-based method (sHP_{eRCN}) relies on transforming a sparse matrix into a doubly-bordered block-diagonal form by utilizing the row-column-net hypergraph model. The objectives in the transformations based on partitioning the respective hypergraph models are shown to correspond to minimizing the upper bounds on the number of cache misses. In the 1D method, the column-net hypergraph model correctly encapsulates the minimization of the respective upper bound. For the 2D method, the row-column-net hypergraph model is enhanced to encapsulate the minimization of the respective upper bound on the number of cache misses. The primary objective in both methods is to maximize the exploitation of the temporal locality due to the access of x-vector entries,

whereas exploitation of the spatial locality due to the access of *x*-vector entries is a secondary objective.

The multiple-SpMxV framework depends on splitting a given matrix into a sum of multiple nonzero-disjoint matrices so that the SpMxV operation is computed as a sequence of multiple SpMxV operations. For an effective matrix splitting, a cache-size-aware top-down approach (mHP_{RCN}) based on 2D partitioning is proposed and implemented by utilizing the row-column-net hypergraph model. An upper bound on the number of cache misses is provided based on this matrix-splitting, and it is shown that the objective in the hypergraph-partitioning (HP) based matrix partitioning exactly corresponds to minimizing this upper bound. The primary objective in this method is to maximize the exploitation of the temporal locality due to the access of both *x*-vector and *y*-vector entries.

The performances of the proposed methods are tested against three state-of-the-art methods: sBFS, sRCM and sHP_{RN}, all of which belong to the single-SpMxV framework, on 17 large sparse matrices. The experiments are carried out by performing actual runs through using OSKI (BeBOP Optimized Sparse Kernel Interface Library). Note that the small letter "s" in method names is used to indicate the single-SpMxV framework, whereas "m" is used to indicate the multiple-SpMxV framework.

Table 5 displays the average performance comparison of the existing and proposed methods for the test matrices. In the table, the second column shows the normalized OSKI running times for original matrices. The other columns show the normalized running times obtained through the reordering methods. Each normalized value is calculated by dividing the OSKI time of the respective method by untuned OSKI running time for original matrices. As seen in the first two columns of the table, optimizations provided through the OSKI package do not improve the performance of the SpMxV operation performed on the original matrices. This experimental finding can be attributed to the irregularly sparse nature of the test matrices.

	Normalized with respect to OSKI running times (without tuning) on matrices with original order							
	Original Order		Existing Methods			Proposed Methods		
			Single SpMxV					Mult. SpMxVs
Matrix Categories	Not Tuned	OSKI Tuned	sBFS	sRCM Modified	sHP_{RN} (1D)	sHP_{CN} (1D)	sHP_{eRCN} (2D)	mHP_{RCN} (2D)
Symmetric	1.00	1.01	0.97	1.23	0.94	0.94	0.86	**0.84**
Nonsymmetric	1.00	1.03	1.00	1.14	0.98	0.93	0.87	**0.84**
Rectangular	1.00	1.20	0.93	1.07	0.90	0.82	0.85	**0.78**
Overall	1.00	1.08	0.96	1.15	0.94	0.89	0.86	**0.82**

Table 5: OSKI running times

As seen in Table 5, on the average, the 2D methods sHP_{eRCN} and mHP_{RCN} perform better than the 1D methods sHP_{RN} and sHP_{CN}, where mHP_{RCN} (adopting the multiple-SpMxV framework) is the clear winner. On the overall average, sHP_{CN} , sHP_{eRCN} and mHP_{RCN} achieve significant speedup by reducing the SpMxV times by 11%, 14% and 18%, respectively, compared to the unordered matrices; thus confirming the success of the proposed methods.

Table 6 displays the normalized preprocessing overhead as well as the amortization values of the ordering methods. An amortization value denotes the average number of SpMxV operations required to amortize the preprocessing overhead.

	Existing Methods				Proposed Methods					
	Single SpMxV								Multiple SpMxVs	
	sBFS		sHP_{RN} (1D)		sHP_{CN} (1D)		sHP_{eRCN} (2D)		mHP_{RCN} (2D)	
Matrix Categories	Over-head	Amor-tization	Over-head	Amor-tization	Over-head	Amor-tization	Over-head	Amor-tization	Over-head	Amor-tization
Symmetric	17	465	194	3135	190	1716	514	3732	920	5097
Nonsymmetric	26	700	314	5078	304	2740	664	4822	1198	6640
Rectangular	23	621	383	6197	254	2292	620	4503	1240	6870
Overall	22	587	286	4620	245	2209	596	4327	1110	6149

Table 6: Average normalized preprocessing overhead and average number of SpMxV operations required to amortize the reordering overhead

As seen in Table 6, the top-down HP methods amortize for larger number of SpMxV computations compared to the bottom-up sBFS method. For example, the use of sHP_{CN} instead of sBFS amortizes after 276% more SpMxV computations on the overall average. As also seen in the table, 2D methods amortize for larger number of SpMxV computations compared to the 1D methods. For example, the use of mHP_{RCN} instead of sHP_{CN} amortizes after 178% more SpMxV computations.

2.7 Multithreaded cache oblivious HYDRO

Supported by: Volker Strumpen (JKU)

Whitepaper: V. Strumpen, "Multithreaded Cache Oblivious HYDRO", PRACE technical white paper

HYDRO is a CFD code developed at IDRIS/CEA. The program is based on an ADI method for solving 2-dimensional flow problems. The original code structure is optimized for vector processing. Each time iteration sweeps through all rows and then through all columns of the alternating directions. HYDRO utilizes a 2-dimensional stencil computation with an 11-point stencil. In an effort to improve the performance of HYDRO, the code is restructured to produce a cache oblivious program. This restructuring minimizes the number of data transfers between the levels of a memory hierarchy and saves a factor of $O(Z^{0.5})$ in the number of cache misses compared to the original iterative program, where Z is the capacity of the higher-level cache. The results given in Table 7 confirm the expectation that HYDRO benefits from the proposed cache-oblivious restructuring.

input data set	original	cache oblivious
small	10.2	9.1
bench	55.6	51.1
input	632.7	553.1
sbench	978.5	606.0

Table 7: Comparison of original HDYRO and cache oblivious HYDRO on an Intel 2.93GHz Xeon X5570 (Nehalem) processor(execution times in secs for four different input datasets)

A multithreaded version of HYDRO written in Cilk [5] is also implemented in this project. This program retains the cache oblivious property. Figure 17 and Figure 18 display the speedup curves of the multithreaded cache oblivious HYDRO program compared to the original HYDRO code on two machines: an SGI Altix 4700 with Intel IA-64 1.6GHz processors and an SGI Ultraviolet with Intel Xeon E7-8837, 2.67GHz 8-core Nehalem processors. The sbench input data set is executed with up to 20 cores. All measurements are taken under regular production conditions while sharing the machines with other users. SGI UV is found to outperform the older SGI Altix by a factor of about 3.5.

The superlinear speedups in these figures can be attributed to the cache oblivious structure of the multithreaded Cilk program. The sublinear outliers of the speedups on the SGI Altix occur because the measurements were performed during regular production conditions. However, the trend is clear: the Itanium processors benefit only marginally from the cache oblivious algorithm. The problem with input data set sbench has sufficiently much parallelism to scale linearly up to 20 cores.

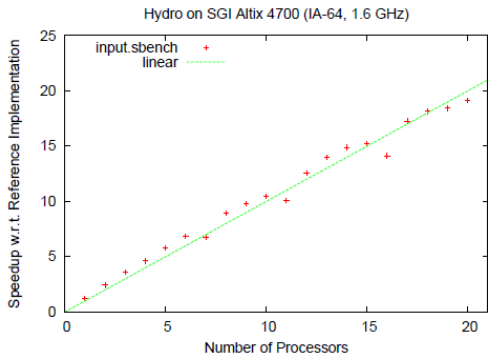


Figure 17: Speedup curves of multithreaded cache oblivious HYDRO on SGI Altix

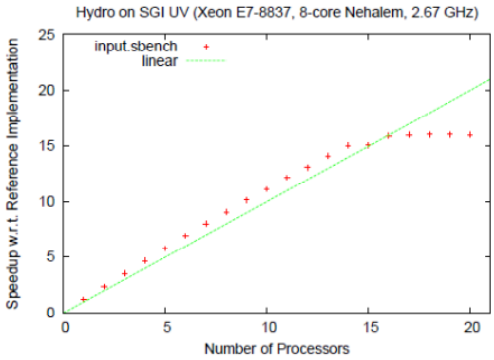


Figure 18: Speedup curves of multithreaded cache oblivious HYDRO on SGI UV

2.8 Investigation of parallel iterative linear solvers for sparse matrices

Supported by: Marzia Rivi (CINECA), Massimiliano Culpo (CINECA)

Whitepaper: Massimiliano Culpo, "Current bottlenecks in the scalability of OpenFOAM on massively parallel clusters", PRACE technical white paper

OpenFOAM is an object-oriented toolkit for continuum mechanics written in C++ and released under the GNU-GPL license. More than being a ready-to-use software it can be thought as a framework for Computational Fluid Dynamics (CFD) programmers to build their own code, as it provides them with the abstraction sufficient to think of a problem in terms of the underlying mathematical model.

The scaling behaviour of different OpenFOAM versions is analysed on two benchmark problems: Lid-driven cavity flow and droplet splash. Results show that the applications scale reasonably well up to a thousand tasks. In-depth profiling was carried out to identify main performance bottlenecks. The calls to MPI-Allreduce in the linear algebra core libraries were found to be the main communication bottleneck. A sub-optimal performance on-core is due to the sparse matrices storage format that does not employ any cache-blocking mechanism at present.

Two possible ways to improve the performance of OpenFOAM solvers are identified. The first is the implementation of cache blocking techniques to reduce the number of cache misses in the core operations due to the random access patterns. This may require a strong effort as the basic matrix class must be revised to allow for the storage of small, contiguous blocks of scalar type as "unit" entries of the format. The second approach is the modification of the basic linear algebra routines in a way that makes them multi-threaded. This will indeed mitigate the increase in the time spent inside MPI routines since well-designed multi-threaded tasks can ideally exploit the resources provided by the largest shared memory portion of the machine. In the context of the second approach, an OpenMP-based multi-threaded matrix-vector multiplication scheme adopting the LDU storage format is developed, implemented and used in the PCG algorithm.

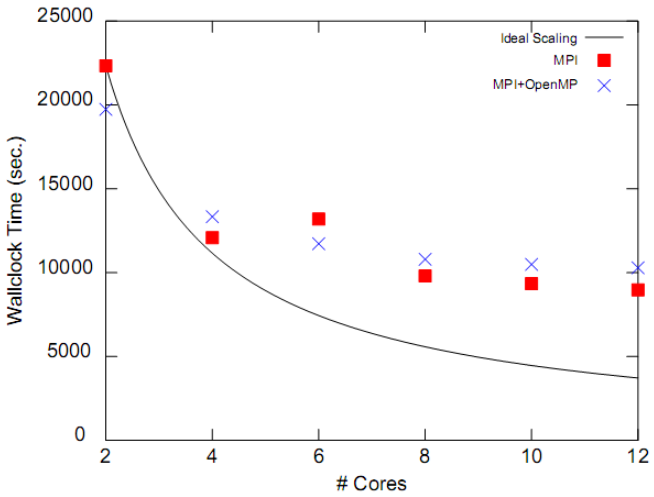


Figure 19: Scaling results obtained on a single node for the 200 × 200 × 200 cells case. A diagonal preconditioner has been used instead of an Incomplete Cholesky Factorization for the PCG algorithm

Figure 19 shows a single node scalability test on a lid-driven cavity flow mesh with 200 × 200 × 200 cells mesh. As seen in the figure, both the MPI and the MPI+OpenMP implementations deviate considerably from the ideal scaling performance when using more than 4 cores. The best performance of the hybrid code results to be 10% slower than the best performance of the pure MPI code. While the latter spawns 12 serial tasks on the node, the former spawns only 2 multi-threaded tasks (one for each node socket). This reduces the time spent in inter-task communication and improves the maximum memory available per task, the last issue being of utmost importance for many Tier-0 architectures.

2.9 Hybridization of parallel sparse matrix vector multiplication

Supported by: Cevdet Aykanat, Ata Türk (Bilkent), Seher Acer (Bilkent), Gündüz Vehbi Demirci (Bilkent)

One of the most important kernels used in scientific applications is sparse matrix vector multiplication (SpMxV), which incurs the dominant cost in the overall performance of the application. Due to the large size of the input matrix that is the output of the discretization of partial differential equations arising in a science or engineering problem, parallel SpMxV is considered as a must for such matrices.

There are three main parallel system architecture models; distributed memory model, shared memory model and hybrid model that combine the previous two models. Message Passing Interface (MPI) is used for distributed memory model where processing elements that run MPI tasks in parallel communicate via messages through the shared network. MPI provides wide portability, since it can also be used in shared memory architectures. OpenMP threads or pthreads are generally used for shared memory model where threads can read/write data from/to a shared memory. For hybrid architectures, in which nodes are connected through a network and each node consists of multiple cores, the hybrid model that combines MPI and OpenMp can be used. Although pure MPI or pure OpenMP programming model can also be used for hybrid architectures, the combination of MPI and OpenMP is expected to give better performance, since the data in the distributed and shared memories are both handled by the corresponding specialized frameworks by exploiting the strengths of both programming paradigms: the high scalability and rich functionality for process control in MPI, and the low communication overhead for small messages and fine-grain parallelism in OpenMP.

Since almost all petascale systems and specifically PRACE Tier-0 systems are hybrid architectures, in this project, the hybrid MPI/OpenMP model for SpMxV operation and its benefits are investigated. For this purpose, OpenMP support is added to a parallel SpMxV

library developed in Bilkent University, which previously only supported pure distributed-memory-parallelism via MPI. A schoolbook implementation of the stabilized biconjugate gradient (BiCGStab) algorithm is used to test the hybrid SpMxV implementation.

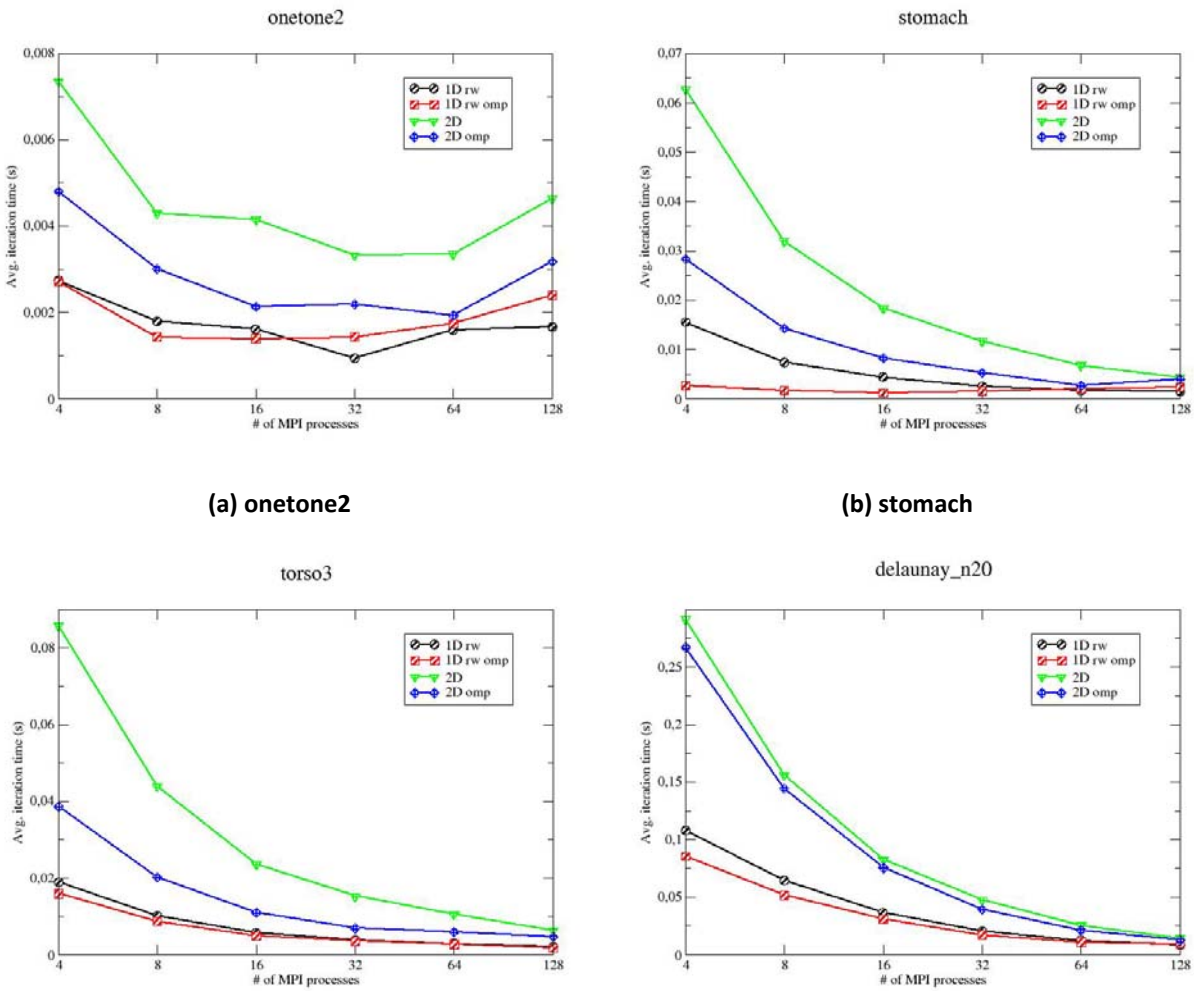
Experiments comparing the results hybrid parallel SpMxV operation with the pure MPI-parallel SpMxV operation in terms of total running time within BiCGStab algorithm are conducted on JUGENE, a BlueGene/P-based supercomputer built by IBM for Forschungszentrum Jüelich in Germany. JUGENE has 73728 nodes (294.912 PowerPC 450 cores, clocked at 850 MHz). It reaches an overall peak performance of 1 PetaFlop/s. Each node in JUGENE has four cores with a peak performance of 13.6 GFlop/s. Each core has 64 KB private L1 cache (32 KB data and 32KB instruction cache) and 4 MB private L2 cache.

BiCGStab is tested on four different test matrices with K={4, 8, 16, 32, 64, 128} MPI tasks once with OpenMP directives and once without OpenMP directives. In the runs with OpenMP directives, each MPI task forks four OpenMP threads for each OpenMP-parallel for loop. The four test matrices are onetone2, stomach, torso3 and delaunay_n20 from The University of Florida Sparse Matrix Collection. Table 8 gives the parameters of test matrices, where m, n and nnz respectively denote the number of rows, number of columns and the number of nonzero elements.

name	m	n	nnz
onetone2	36057	36057	222596
stomach	213360	213360	3021648
torso3	259156	259156	4429042
delaunay_n20	1048576	1048576	6291372

Table 8: Test matrix parameters

In Figure 20, x-axis denotes the number of MPI tasks, whereas the y-axis denotes the running time of an iteration on average in BiCGStab. “1D rw” and 2D curves represent the execution of the original (pure MPI) 1D- and 2D-parallel SpMxV, whereas the “1D rw omp” and “2D omp” curves represent the execution of the hybrid MPI/OpenMP parallel SpMxV.



(a) onetone2 (b) stomach

(c) torso3 (d) delaunay_n20

Figure 20: Running times of BiCGStab using SpMxV with and without OpenMP.

As seen in Figure 20(c) and (d), torso3 and delaunay_n20 are big enough that both MPI and OpenMP parallelization overheads can be amortized even in 128 MPI tasks and 128×4=532 OpenMP threads. However, as seen in Figure 20(a), onetone2 reaches to its best running time for maximum 32 MPI tasks and 32×4=128 OpenMP threads, and parallelization incurs more overheads for larger number of tasks and threads. Similarly, as seen in Figure 20(b), stomach reaches to its best performance around 32-64 MPI tasks. This experimental results show that adding OpenMP to MPI-parallel SpMxV gives better performance only when the granularity of each MPI task is big enough to exploit shared memory parallelism and not to incur additional parallelization overhead.

2.10 Effects of system topology in performance

Supported by: Cevdet Aykanat (Bilkent), Ata Türk (Bilkent), Gündüz Vehbi Demirci (Bilkent)

PRACE Tier-0 systems are built of hundreds of thousands of cores with large network diameters. In petascale computing, communication with distant nodes can be a bottleneck, especially if network contention is existent. It is a general belief that if virtual cut-through and wormhole routing is utilized, the message latency is independent of the distance in absence of blocking. However, especially for small messages exchanged in petascale computing systems

with large diameters, this is not true. In this project, a number of benchmarking experiments analyzing the effects of distance and contention on communication costs of nodes in PRACE Tier-0 petascale systems are conducted. The aim of this study is to guide application developers in identifying scenarios where the topology of the underlying system may affect performance and scalability of the application.

The network topology of current petascale computing systems (e.g., Cray's XT, IBM's Blue Gene family) is a three-dimensional (3D) torus. In such big systems, the diameter of the network is very large (e.g., between 20 to 60 hops for Blue Gene/P and XT5). This can have a significant effect on message latencies when multiple messages start sharing network resources. To observe these effects, *MPI Contention Benchmark Suite* and *TopoMgrAPI* from University of Illinois Parallel Programming Laboratory are run on PRACE Tier-0 system JUGENE. In Blue Gene/P a midplane composed of 512 nodes forms a torus of size 8 × 8 × 8 in all directions. Smaller allocations than a midplane are a torus in some dimensions and mesh in others. Larger allocations than a midplane (512 node) are complete tori.

Two sets of experiments are conducted to evaluate the effect of hops (links) traversed by messages, on their latencies. In the first set of experiments, the change in message latencies for varying number of hops in absence of contention is measured. One particular node is chosen from the allocated partition to control the execution. This node sends B-byte messages to every other node in the partition, and expects same-sized messages in return. For machines with multiple cores per node, this benchmark places only one MPI task per node to avoid intra-node messaging effects. The message size B is varied and for each value of B, the difference between the maximum and minimum time for sending a message to every other node is recorded. Wormhole routing suggests that message latencies are independent of distance in the absence of contention, for sufficiently large message sizes. This set of experiments show the effect of the number of hops on small-sized messages. Since the distance from the master node to other nodes varies, we observe different message latencies depending on the distance. Experiments are conducted on four allocations of BG/P, ranging in size from 128 to 1024 nodes (torus sizes 8×4×4 to 8×8×16) and the percent difference variation with increasing B is shown as a separate curve for each allocation.

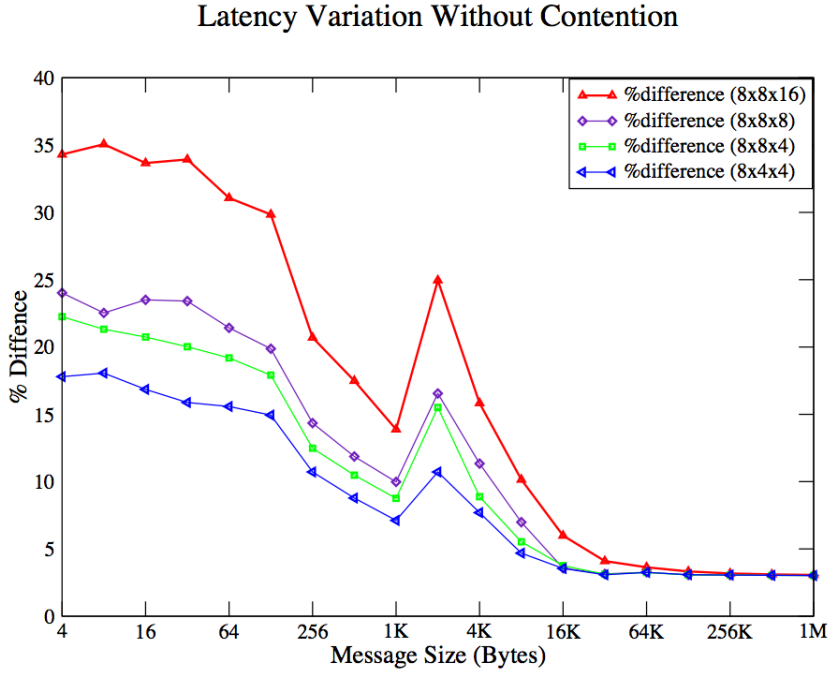


Figure 21: Latency variation without contention

As seen in Figure 21, the difference between the maximum and minimum values (% difference curves) decreases with increasing message size for all the curves. We see a kink in the curves and a corresponding jump in the message latencies at 2 KB messages. This is due to use of different routing protocols on BlueGene/P for different message sizes. For message sizes greater than 1200 bytes, the MPI rendezvous protocol is used where an initial handshake is done before the actual message is sent. Note that the % difference values are in the range of 10% to 35% for message sizes up to 8 KB (in the 8×8×16=1024 nodes line). Many applications use communication units in this range and hence it is not wise to always assume that message latencies do not depend on hops for most practical message sizes. Also note that, for a fixed message size, the difference between minimum and maximum latencies increases with the increase in diameter of the partition. 8×4×4=128 and 8×8×4=256 node partitions are not complete tori in all dimensions and hence their diameter is the same as that of the 8×8×8=512 node partition, which are 12. The diameter of the 1024 node partition is 16 and hence a steep increase in the percentage difference for the small and medium sized messages.

In the second set of experiments, message latencies in the presence of contention are measured. In these experiments the benchmarking suite places one MPI task on each core to create as much contention as possible. All MPI tasks are grouped into pairs and the smaller rank in the pair sends messages of size B bytes to its partner and awaits a reply. All pairs do this communication simultaneously. The average time for the message sends is recorded for different message sizes. To quantify the effect of hops on message latencies this benchmark is run in two modes: Near Neighbor Mode (NN): The ranks that form a pair only differ by one. This ensures that everyone is sending messages only 1 hop away (in a torus) and Random Processor Mode (RND): The pairs are chosen randomly and thus they are separated by a random number of links.

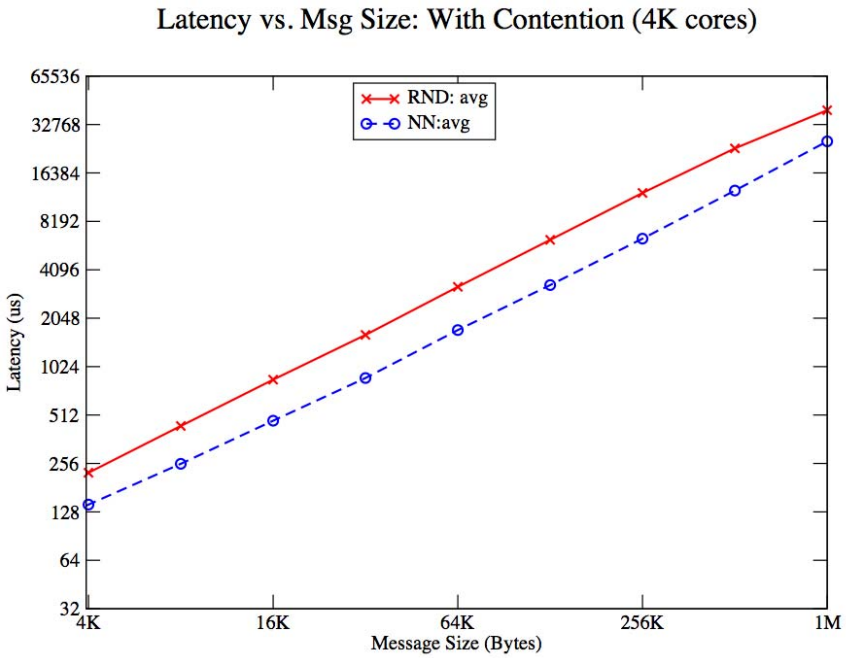


Figure 22: Latency versus message size with contention

As seen in Figure 22 random-processor (RND) latencies are more than the near-neighbor (NN) latencies. This also shows that the number of hops has a significant impact on the message latencies in the presence of contention, which increases with larger messages because of a proportional increase in packets on the network.

To sum up, the results given in this work confirm the adverse effect of the number of hops on small-sized messages. Since most of the applications discussed in this deliverable rely on

domain decomposition methods that target and usually achieve small-sized messages without considering the processor-to-processor distance during decomposition, topology-aware domain decomposition and mapping should be considered for better scaling of target applications on peta-scale systems with large diameters.

2.11 Conclusion

Subtask 7.5.A has successfully provided support to a wide range of HPC codes and user communities. Experts from Task 7.5.A not only supported a total of 6 applications by addressing their algorithmic scalability challenges, but also provided reports on diverse techniques which should be considered to enable petascaling of applications in general.

The applications which were supported were all selected from Tasks 7.1 and 7.2 and can be listed as: ELMER, SPECFEM3D, CP2K, GROMACS, NAMD and OpenFOAM. For each of these applications, algorithmic optimization subprojects, led by teams collaborating from several different PRACE member HPC centres around Europe, are conducted. For ELMER, efficient massively parallel finite element tiering and interconnecting methods; for ELMER and OpenFOAM efficient algorithms for the solution of incompressible Navier-Stokes equations, for SPECFEM3D faster boundary element methods, for CP2K more efficient sparse-matrix-matrix multiplication schemes are proposed and for GROMACS and NAMD an analysis of the workload and communication distribution is conducted.

Apart from providing support in specific applications, several algorithms and analysis that can be used by application developers in scaling their applications were also provided. These studies can be grouped into three: (i) Cache oblivious computation restructuring and reordering schemes for repeated computations that enables application developers to exploit today's deep memory hierarchies more thoroughly: Even though the benefits of such schemes were found to be significant, caution was advised since the cost of such schemes amortize only if the computations are repeated sufficient number of times. (ii) Hybrid parallel computation models combining MPI and OpenMP: By exploiting the high scalability and rich functionality for process control in MPI, and the low communication overhead for small messages and fine-grain parallelism in OpenMP, it is possible to devise more efficient algorithms, but the granularity of MPI tasks and OpenMP tasks should be designed carefully to obtain the desired effects. For example, if the problem size per MPI task falls below a certain threshold, pure MPI solutions can outperform hybrid schemes. (iii) Analysis of PRACE Tier-0 system JUGENE in terms of the effects of system topology in performance: The analysis shows that if the number of nodes utilized are high (e.g., 1K nodes or more) and the messages used in communication are small in size (e.g., up to 8 KB) and numerous, application developers should consider topology-aware schemes to achieve scalability.

The 10 projects in Subtask 7.5.A have resulted in 7 whitepapers and 3 detailed reports. In addition to this deliverable, all whitepapers will be available online and thus will be useful for users facing similar algorithmic challenges. In summary, Subtask 7.5.A produced supporting algorithmic guidelines for European HPC users.

3 Scalable Libraries

The use of numerical library routines in large-scale applications is a well-established approach to ensure optimized, portable and resilient software. The current rapidly changing hardware environment means that it is now more important than ever that code developers can rely upon a suite of libraries and routines that provide guaranteed performance and resilience for the range of computational kernels common to the majority of scientific codes.

The advantages of using established numerical libraries in large-scale application software are well recorded, and are summarised here:

1. Their usage accelerates software development time by allowing code developers to concentrate on the scientific modelling and specialist features of their code, rather than the computational core.
2. Providing the libraries are maintained, optimized and applicable to the latest architectures, good performance and scalability is guaranteed. Libraries may also be able to exploit highly specialized optimization strategies such as use of mixed-precision routines or very low-level operations for computational cores.
3. Library routines usually undergo rigorous testing procedures and therefore the user is guaranteed robust performance, numerical stability and reliable results.
4. Libraries often contain more sophisticated error reporting and error handling schemes than numerical hand-written software.
5. New numerical libraries often evolve from existing libraries and therefore upgrading to the new libraries often involves only low levels of disruption to the host application code. An example of this feature investigated here is the relatively straightforward substitution of ScaLAPACK routines with those from the ELPA library in computational chemistry codes in the project described in Section 3.1.

Projects in this sub-task have sought to analyse performance of library routines that are applicable to both specific scientific application codes in PRACE and the wider scientific community in general. Deliverable 6.1 (D6.1) [6] in the PRACE Preparatory Phase (PRACE-PP) surveyed over 70 application codes used in institutions throughout Europe and categorized them according to the algorithmic methods that capture patterns of computation and communication. The D6.1 report lists the “Seven Dwarfs” algorithms that consume the bulk of compute cycles on high-end parallel systems, as classified by Phillip Colella [7]:

- Dense linear algebra – data is stored in dense matrices or vectors. Typical algorithms would include Cholesky decomposition for symmetric systems, Gaussian elimination for non-symmetric systems and symmetric and unsymmetric eigensolvers.
- Sparse linear algebra – data is stored in compressed format as it largely consists of zeros and is therefore accessed via an index-based load. Typical algorithms would include iterative methods based on Krylov subspaces.
- Spectral methods – data is in the frequency domain and requires a transform to convert to spatial/temporal domain. They are typified by, but not restricted to, Fast Fourier Transforms.
- Particle methods – data consists of discrete particle bodies that interact with each other and/or the “environment”.
- Structured grids – represented by a regular grid. Points on the grid are conceptually updated together via equations linking them to other grids.
- Unstructured grids – data is stored in terms of the locality and connectivity to other data. Points on grid are conceptually updated together, but updates require multiple levels of redirection.

- Map-reduce methods – for example Monte Carlo methods, where calculations are independent of each other.

In this subtask we include projects that analyse libraries covering three of these main areas – dense linear algebra, sparse linear algebra and spectral methods (e.g. fast fourier transforms). The projects in each category are summarised below:

- Dense Linear Algebra
  - 3.1 *Numerical Library Eigensolver Performance on PRACE Architectures*
  - 3.2 *Petascale Enabling and Support for DALTON*
  - 3.3 *Optimizing GPAW*
- Sparse Linear Algebra
  - 3.4 *Scalable Solvers for Large Sparse Linear Systems*
- Spectral Methods
  - 3.5 *FFT Library Performance on PRACE Systems – DL_POLY*
  - 3.6 *FFT Library Performance on PRACE Systems – Quantum ESPRESSO*

Although focussing upon these three areas does not provide a truly comprehensive analysis of the algorithms underpinning scientific application codes used in Europe, the results from the application surveys undertaken in D6.1 show that it does provide a reasonable coverage of the foremost areas of algorithmic interest. Moreover, higher-level software packages such as those used for Finite Volume, Finite Element Analysis and Multigrid methods, apply to the structured grid and unstructured grid 'dwarfs' but also rely upon underlying efficient linear algebra routines. Each area also tends to throw up different challenges to obtaining good performance, including ensuring efficient CPU utilization, optimizing memory access patterns and introducing the highly scalable parallelism necessary to take full advantage of PRACE Tier-0 hardware.

As numerical library routines usually need to meet high quality software standards (including those listed above and in Section 3.8) their development often lags considerably behind advances in hardware. It is also often the case that new numerical libraries initially feature a very limited number of widely-used routines, for example Cholesky decomposition, LU factorization and then expand their range incrementally later. This is true of the successors to LAPACK and ScaLAPACK, namely the PLASMA and MAGMA libraries, which are being designed specifically for multi-core and many-core (e.g. GPUs, Intel MIC) architectures. At the time of this project, applicable routines, such as symmetric eigensolvers, were unavailable in these new libraries. PETSc is also under devlopment to introduce mixed-mode MPI/OpenMP approaches into its code-base. The latest release of FFTW also includes mixed-mode options, and this is investigated in the project summarised in Section 3.6.

The potential to re-use library code within a parallel application is constrained by numerous issues that do not affect serial numerical codes. It is tempting for designs of parallel numerical libraries to assume that the input and output data will be laid out in memory and distributed over processes in a way that is friendly to the numerical methods chosen. Furthermore, a library may be developed for message-passing (MPI), shared-memory threads (OpenMP), something more exotic (e.g. CUDA on GPUs), or a hybrid paradigm. On the other hand, the application developer, having a different problem to solve (perhaps containing sub-problems from several of the "seven dwarfs") will choose a decomposition strategy and parallelisation paradigm suited to a different set of constraints. Hence an otherwise excellent library may not be usable within a particular application, even when the target architecture and programming language are the same.

This issue is highlighted by the use of spectral methods in computational chemistry. Although Quantum Espresso uses a domain decomposition that is compatible with parallel FFT libraries such as FFTW, DL_POLY does not. To do so would require global communications to re-

distribute the data into a compatible with the library. Do the performance gains from using an optimised library outweigh the cost of re-distributing the data before and after the call to the library? We find that the answer depends on problem size.

3.1 Numerical Library Eigensolver Performance on PRACE Architectures

Contributor: Andrew Sunderland (STFC)

Code(s): LS-Dalton, GPAW, CP2K, CRYSTAL, PRMAT

Whitepaper Title: Numerical Library Eigensolver Performance on PRACE Architectures

Application Area(s): Astronomy and Cosmology, Computational Chemistry, Materials Science, Atomic and Molecular Physics

Algorithm(s) under investigation: Dense Linear Algebra, Symmetric Real/Complex Matrix Diagonalisation Methods, Divide and Conquer, Multiple Relatively Robust Representations, Modified Divide and Conquer, Block Cyclic Data Distributions.

Parallelization Technique(s): BLACS, Block Cyclic Distribution

Numerical Libraries used:

ScaLAPACK (Scalable Linear Algebra Package) – PDSYEV, PDSYEVD, PDSYEVR, using BLACS, BLAS, MKL, University of Tennessee (http://www.netlib.org/scalapack/)

ELPA (Eigenvalue Solvers for Petaflop Applications) – solv_evp_real, using MPI, BLAS, MKL, Rechenzentrum Garching der Max-Planck-Gesellschaft (RZG) (http://elpa.rzg.mpg.de/)

Hardware Platforms Investigated(s): JUGENE, CURIE and Hector (Cray XE6, UK)

Short description of any Benchmark Datasets: Several datasets are taken from CRYSTAL and PRMAT programs. The CRYSTAL datasets represent Typical Hartree-Fock systems, which are typical of those from other computational chemistry and materials science applications. Two different Kohn-Sham matrices from Hatree-Fock calculations were used for the tests with dimension N=7194 and N=20480. Both systems are real and symmetric and contain some groups of closely clustered or degenerate eigenvalues that normally can represent significant computational challenges for parallel solvers. PRMAT datasets are typical of Hamiltonian matrices. They are extracted from an Fe2+ scattering calculation. Each Hamiltonian matrix represents a sector calculation, of which there are usually around 4-10 in a typical case. The matrix dimensions are 10032, 20064 and 35904. These are roughly typical sizes for current large-scale runs.

Project Description: Parallel eigensolver operations are at the computational core of many large-scale scientific and engineering application codes. This project analyses parallel performance of established and newly developed parallel library routines for eigensolves and diagonalization, using datasets from large-scale application codes.

Main Performance Bottlenecks: Parallel eigensolvers typically represent a significant computational bottleneck to performance due to the number of floating point operations required (generally of order matrix dimension cubed) and the high level of communications required in both the reduction to tridiagonal form stage and the tridiagonal eigensolve stage. The final stage, back transformation, parallelizes trivially and is therefore inherently scalable. Eigensolvers based on Divide and Conquer (D&C) methods were introduced several years ago into ScaLAPACK 1.7. These routines were a significant improvement over the older QR and bisection and inverse iteration based routines. This project also analyses two very recent developments. Firstly ScaLAPACK 1.8 includes eigensolvers based on Relatively Robust

Representation (MRRR) methods, and secondly routines from the ELPA library that attempt to optimize the eigensolve based on a two-stage approach for the tridiagonal eigensolver along with other optimizations.

Performance Results:

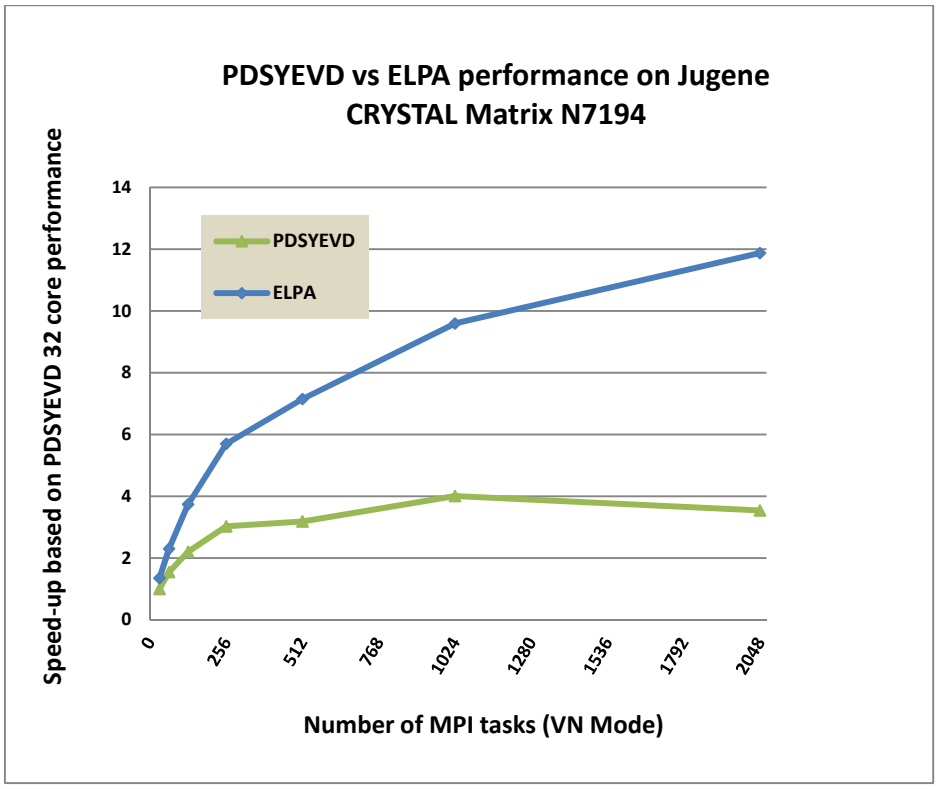


Figure 23: Performance comparison of PDSYEVD and ELPA

Figure 23 compares the performance of the established parallel symmetric ScaLAPACK solver PDSYEVD, based on a divide-and-conquer approach, against the EPLA (1 Stage) eigensolver on the PRACE JUGENE machine. The test case is a typical medium-sized matrix of dimension 7194 from simulations undertaken by the (MPP)CRYSTAL code. The ELPA routine performs around 2-3 times faster over a range of core counts. No performance improvements are observed in ScaLAPACK runs with over 1024 cores, but the ELPA routine scales out to 2048 cores. These results are representative of those presented in the whitepaper.

Best Practices: It is clear that BLACS-based ScaLAPACK routines do not scale sufficiently well to exploit the potential power of the latest range of high-end computing platforms *unless* the dataset dimensions are exceptionally large (N > 20.000). Even in these large cases, scaling does go beyond around ten thousand cores. The MPI-based ELPA library is shown to be both faster and have better scaling properties than ScaLAPACK, but these routines again do not scale much beyond a few thousand cores for typical (i.e. not exceptionally large) cases. It is to be hoped that future releases of the MAGMA, PLASMA and ELPA (hybrid programming) libraries will include symmetric eigensolver routines and therefore can improve performance at high core counts.

3.2 Petascale Enabling and Support for DALTON

Contributors: Ole Widar Saastad (UiO), Maria Francesca Iozzi (UiO), Thomas Kjærgaard (UiO), Trygve Helgaker(UiO), Simen Reine (UiO)

Code: DALTON, http://dirac.chem.sdu.dk/daltonprogram.org/

White Paper Title: Petascaling and Performance Analysis of Dalton on Different Platforms

Application Area(s): Computational Chemistry

Algorithm(s) under investigation: Molecular density-functional theory – involving analytical and numerical integral evaluation (one-electron contributions, Coulomb, exchange and exchange-correlation) and linear-algebra operations (mostly matrix multiplication, linear solvers and diagonalization routines)

Parallelization Technique(s): MPI and OpenMP

Numerical Library used:

LAPACK and ScaLAPACK (most notably PDSYGVX, PDGEMR2D)

Hardware Platforms Investigated(s): linux and darwin

Short description of any Benchmark Datasets: The valinomycine molecule (168 atoms, 1584 basis functions), the insulin molecule (787 atoms, 7604 basis functions) and a water cluster (2208 atoms, 9568 basis functions). The first two molecules represent the typical biochemical systems we would like to investigate in this project, and the water cluster demonstrates a large molecular system we would like to investigate in the future. More extensive benchmarks will be provided in the whitepaper.

Project Description: The LSDALTON module consists of a newly developed code which employs state-of-the-art linear-scaling HF and DFT technology, and is aimed for application to systems consisting hundreds to thousands of atoms.

Contribution: Simen Reine, Thomas Kjærgaard and Trygve Helgaker are responsible for the hybrid OMP/MPI implementation of LSDALTON, and Maria Francesca Iozzi and Ole Widar Saastad are responsible for the hardware and tools.

Main Performance Bottlenecks: The main bottlenecks are integral evaluation, involving the one-electron, Coulomb, exchange and exchange-correlation contribution, and wave-function and response optimization routines that rely on linear-algebra routines. For the integral evaluation specialized MPI/OpenMP routines are necessary, whereas for the wave-function and response optimization part we can exploit the ScaLAPACK library routines. Also for the efficient evaluation of the Coulomb contribution using density-fitting we rely on ScaLAPACK for the linear-equation solver used in that context.

Application Optimization work done: We have interfaced the linear algebra routines used in LSDALTON with ScaLAPACK, and tested the functionality extensively for small to medium sized systems. Application to larger systems remains to be tested.

Performance Results: For the valinomycine molecule we have preformed some preliminary testing of the current development code (see Table 9). The MPI/OpenMP scaling of the integral evaluation is not yet impressive, but the numbers give an indication of how important ScaLAPACK is for our developments.

# of MPI/OMP	1/1	8/1	64/1	128/1	64/8	128/8
Regular integral evaluation	7598	1178	317	209	79	67
Fast integral evaluation	456	78	22	-	8	-
Linear algebra	171	-	-	-	-	-

Table 9: Performance of the LSDALTON code on valinomycine (blyp/6-31+G*) in seconds

For the insuline and water cluster, timings have been limited to a pure OpenMP calculation using 12 threads, see Table 10. We rely on the ScaLAPACK interface to reduce the memory

requirements in order to enable calculations using the hybrid MPI/OpenMP scheme. This interface is currently not working for the larger systems due to a bug.

	Insuline	Water cluster
Integral evaluation	21711	19295
Linear algebra	866	2030
Memory requirements	30 GB	57 GB

Table 10: Performance of the LSDALTON code on the insuline molecule and a water cluster in seconds

For these molecules the time-dominant step is clearly the integral evaluation (which has gotten most of our focus in this project so far). However, it is also clear that good scalability of the linear algebra is essential in order to achieve petascaling for the LSDALTON code, in particular when using the fast integral evaluation routines - that uses the density-fitting approximation. Good scalability for the linear algebra part can easily be achieved by interfacing LSDALTON to the ScaLAPACK library. See the whitepaper for details.

Best Practices: The availability, maintenance and development of ScaLAPACK is absolutely essential for developments such as these.

3.3 Optimizing GPAW

Contributors: Jussi Enkovaara (CSC), Martti Louhivuori (CSC), Vladimir Slavnic (IPB) Mikael Rännar (SNIC)

Code: Real-space Grid Implementation of the Projector-Augmented Wave Method (GPAW)

Publications:

J. Enkovaara, C. Rostgaard, J. J. Mortensen et al.
Electronic structure calculations with GPAW: a real-space implementation of the projector augmented-wave method, J. Phys.: Condens. Matter 22, 253202 (2010)

J. Enkovaara (CSC), M. Louhivuori (CSC), V. Slavnic (IPB), M. Rannar (Umea Univ.), “Optimizing GPAW”, PRACE technical white paper

Application Area: Condensed Matter Physics

Algorithm(s) under investigation: Dense Linear Algebra - symmetric real matrix diagonalisation and complex hermitian positive definite matrix inversion

Parallelization Technique: MPI

Numerical Libraries used:

Elemental (V 0.71) (http://code.google.com/p/elemental/). Elemental is a C++ framework for distributed-memory dense linear algebra using MPI and is primarily developed by Jack Poulson, The Institute for Computational Engineering and Sciences, University of Texas at Austin. One main feature is that it uses element-wise distribution of matrices. Routines used: HermitianEig, Cholesky, TriangularInverse, HPDInverse.

ELPA (Eigenvalue soLvers for Petaflop Applications) (http://elpa-lib.fhi-berlin.mpg.de/). ELPA is an Open Source Fortran-based computational library for the parallel solution of symmetric or Hermitian, standard or generalized eigenvalue problems. ELPA is an MPI-only implementation and works as a "drop-in enhancement" for ScaLAPACK-based infrastructures. Routines used: solve_evp_real, solve_evp_real_2stage, cholesky_complex, invert_trm_complex

Hardware Platforms Investigated: CURIE

Short description of Any Benchmark Datasets: The symmetric (or hermitian) matrices used in the tests are random generated matrices of size 20000x20000.

Project Description: GPAW is versatile software package for simulations of various nanostructures utilizing density-functional theory and time-dependent density-functional theory. Even though GPAW is already used for massively parallel calculations in several supercomputer systems, some performance bottlenecks still exist. The goal of the project is to improve scalability and investigate some alternative numerical libraries.

Main Performance Bottlenecks: Large dense matrix diagonalizations are expected to become bottlenecks in the future and the expectations are that when the number of electrons increases, the routines used from ScaLAPACK are unlikely to scale on par with the other parts of the code. We investigated alternatives to the current ScaLAPACK based implementation.

Application Optimization Work: We looked for numerical libraries with routines that could be an alternative to the currently used routines from ScaLAPACK. After an initial search, we chose two libraries (Elemental and ELPA) to test and in this stage of the process it was decided that it would be enough to do a stand-alone test of the routines, i.e., not incorporate the investigated routines into the application.

Performance Results: The tested Cholesky + inversion routines show improvements over the corresponding ScaLAPACK routines that are used in the current implementation of GPAW. For all libraries there is no gain in using more than 1024 cores. The shortest execution time (in seconds) we find for 1024 cores and the routine from the Elemental, 8 seconds compared to ScaLAPACK's 25 seconds.

Best Practices: To incorporate ELPA into GPAW should not be too hard since it uses the same framework as the currently used ScaLAPACK. For Elemental it is a bit more difficult since it uses a different distribution and C++ classes. Both libraries are relatively new and under development, indicating that it might be a good idea to keep them under surveillance for a possible future incorporation.

3.4 Scalable Solvers for Large Sparse Linear Systems

Contributors: Murat Manguoglu (UYBHM)

Code(s): OpenFOAM

Publications

Manguoglu M, A domain-decomposing parallel sparse linear system solver, Journal of Computational and Applied Mathematics, 236, 319-325, 2011,

Manguoglu M, Takizawa K, Sameh A, Teduyar T, A parallel sparse algorithm targeting arterial fluid mechanics computations, Computational Mechanics, 48, pp.377-384, 2011

Manguoglu M, A Parallel Sparse Solver and Its Relation to Graphs, Computational Electromagnetics Workshop (August 10-13, 2011), Izmir, Turkey

Sameh A, Manguoglu M, Towards a Scalable Parallel Sparse Linear System Solver, Coupled Problems in Science and Engineering, June 20-22, 2011, Kos Island, Greece

Manguoglu M, “Parallel solution of sparse linear systems in OpenFOAM”, PRACE technical white paper

Application Area(s): Computational Fluid Dynamics and solution of general sparse linear systems

Algorithms under investigation: Sparse linear system solvers

Parallelization Technique(s): MPI and OpenMP (can also use GPUs via cuBLAS)

Numerical Library used:

MUMPS (Multifrontal Massively Parallel Sparse Direct Solver): developed by CERFACS, CNRS, ENS Lyon, INPT (ENSEEIHT)-IRIT, INRIA, and University of Bordeaux**;** http://graal.ens-lyon.fr/MUMPS/

ILUPACK: TU Braunschweig; http://www.icm.tu-bs.de/~bolle/ilupack/

MKL (Math Kernel Library): Intel Corporation, http://software.intel.com/en-us/articles/intel-mkl/

Hardware Platforms Investigated(s): CURIE

Short description of Benchmark Datasets: We use matrices from the University of Sparse Matrix Collection (http://www.cise.ufl.edu/research/sparse/matrices/) as well as linear systems extracted from OpenFOAM. The OpenFOAM matrix is extracted from the steady state 3D lid-driven cavity problem with a Reynold's number of 5000.

Project Description: Parallel solution of sparse linear systems is the main bottleneck for most scientific and engineering simulations such as the ones in computational fluid dynamics, electromagnetics, and many other areas. Well known methods include direct and iterative solvers. Direct solvers are robust but lack scalability (especially strong scalability for a fixed problem size). Iterative solvers are more scalable but lack robustness. We have verified these points in several publications for various application domains. In addition a new solver that combines the desirable characteristics of direct and iterative solvers has been presented. We included a comparison of the solver using MPI with MPI/OpenMP hybrid implementation in the PRACE technical white paper.

Main Performance Bottlenecks: Sparse direct solvers are not scalable for large 3D problems. The main reason being excessive communication at higher core counts. Arithmetic operations performed at higher rate of today's processors, however the interconnection network and memory remains to be the main bottleneck. Iterative solvers are also affected by these bottlecks such as global reduction operation that turns out to prevent scalability especially when a large number of processes are used.

Application Optimization Work Done: Our work has been focused on OpenFOAM in collaboration with WP7.2. We have been developing a new sparse linear solver called DDPS. This solver has been tested on various application domains. It is planned that the linear solver be made publicly available under GPL license, expected within the next months.

Performance Results: Detailed performance results are presented in the first three publications. The following results show the scalability of DDPS solver for the 3D lid-driven cavity problem with 4 million unknowns extracted from OpenFOAM on CURIE cluster. In Table 11, we are using DDPS such there only one partition and one thread per MPI process. In Table 12, on the other hand, we use one partition and 16 threads per MPI process. Threading is done via OpenMP and using threaded routines from Intel MKL. We observe that there is some benefit of using threads with MPI if the amount of work per process is large. However, this turns into a disadvantage as the MPI processes are communicating across nodes and the communication time starts to dominate the total time. Hence, the best time is obtained if one uses MPI only for the given problem.

nodes	processes	cores	factor	solve	total
4	64	64	0,03	0,97	1,00
2	32	32	0,09	1,82	1,91
1	16	16	0,20	19,38	19,58
1	8	8	0,41	22,66	23,08

Table 11: Scalability (given as wall clock time in seconds for factor, solve and total) of DDPS using one MPI process per core on CURIE

nodes	processes	cores	factor	solve	total
64	64	1024	0,03	3,71	3,74
32	32	512	0,09	1,46	1,55
16	16	256	0,18	6,57	6,75
8	8	128	0,37	7,06	7,44

Table 12: Scalability (given as wall clock time in seconds for factor, solve and total) of DDPS using 1 MPI process per node and 16 threads per process on CURIE

Best Practices: Through the experience we had with MPI vs. MPI/OpenMP hybrid programming, we observe that –in order to get best performance out of a given number of nodes- MPI only should be used. We showed that by using a hybrid approach with direct and iterative solvers one can obtain better scalability than direct solvers and more robustness compared to classical iterative Krylov subspace techniques.

3.5 FFT Library Performance on PRACE Systems – DL_POLY

Contributors: Andrew Sunderland (STFC), Stephen Pickles (STFC), Ivan Girotto (ICHEC), Peter Nash (ICHEC), Michael Lysaght (ICHEC), Milos Nikolic (IPB), Aleksandar Jovic (IPB), Josip Jakic (IPB), Vladimir Slavnic (IPB)

Code(s): DL-POLY

White Paper, Technical Report or Scientific Publication:

Andrew Sunderland, Stephen Pickles, Ivan Girotto, Peter Nash, Michael Lysaght, Milos Nikolic, Aleksandar Jovic, Josip Jakic, and Vladimir Slavnic. “FFT Library Performance on PRACE Systems”, 2012.

Application Areas: Molecular Dynamics, Computational Chemistry, Materials Science

Algorithm under investigation: Fast Fourier Transforms

Parallelization Techniques: MPI/OpenMP

Numerical Libraries used:

DAFT (Daresbury Advanced Fourier Transform), I.J.Bush, STFC Daresbury Laboratory, UK

FFTW (Fastest Fourier Transform in the West), Matteo Frigo, Steven Johnson, MIT, USA, http://fftw.org

FFTE (Fastest Fourier Transform in the East), Daisuke Takahashi, Tsukuba, Japan, http://www.ffte.jp

Hardware Platforms Investigated: JUGENE, CURIE

Project Description: The objective of this project is to benchmark the 3-dimensional FFTs, as used within the well-known computational chemistry code DL_POLY.

Short description of Any Benchmark Datasets: The benchmark dataset is based on a NaCL test case with 1.728.000 ions. The size of the global arrays used by the FFTs in this test case is 256x256x256. We also run benchmarks with smaller (128x128x128) and larger (512x512x512) 3D grids. The 128^3 and 256^3 are roughly typical sizes for current large-scale runs. 512^3 is an exceptionally large case, beyond most current requirements, but could possibly feature in future runs on peta-scale or exa-scale architectures.

Contribution: The contributors have obtained performance and scalability results for parallel 3-d FFT performance on the PRACE systems JUGENE and CURIE, using a benchmark code developed to study FFT performance on problem sizes and data distributions encountered in DL_POLY. In this work, the benchmark code was extended to include FFTW (which, at version 3.3.1, supports MPI-based FFTs), enabling direct comparison between FFTW, FFTE and DL_POLY's native method, DAFT.

Main Performance Bottlenecks: The main performance bottleneck of 3-d FFTs is communications. Once 3-d data is distributed over MPI processes, all-to-all communications are unavoidable. Applications that rely on FFTs adopt different data decomposition strategies; 1-d decompositions give each process a complete 2-d slab, 2-d decompositions give each process a complete 1-d pencil, while 3-d decompositions give each process a block that does not span the global domain in any dimension. *Slab* decompositions tend to perform well on small process counts; *pencil* decompositions scale better, but also eventually run out of steam. Efforts to optimize the performance of 3-d parallel FFT libraries have tended to focus on slab and pencil decompositions.

DL_POLY is unusual in employing a block decomposition throughout. To employ a 3rd party parallel FFT library in DL_POLY would require additional all-to-all communication phases to redistribute data before and after the library call.

Performance Results: Performance results for the FFTW, FFTE and DL_POLY's native FFT routine DAFT are reported for the PRACE systems JUGENE and CURIE. The comparisons are somewhat unfair to DAFT, because FFTW and FFTE only support slab and pencil decompositions, and are therefore not a plug-in replacement for DAFT in DL_POLY.

The results are ambiguous. Neither FFTE nor FFTW emerges as a clear winner on all problem sizes and architectures. Hence both FFTW and FFTE have room for improvement. DAFT, although outperformed by both FFTE and FFTW on slab and pencil decompositions at low process counts, does show promising scaling characteristics.

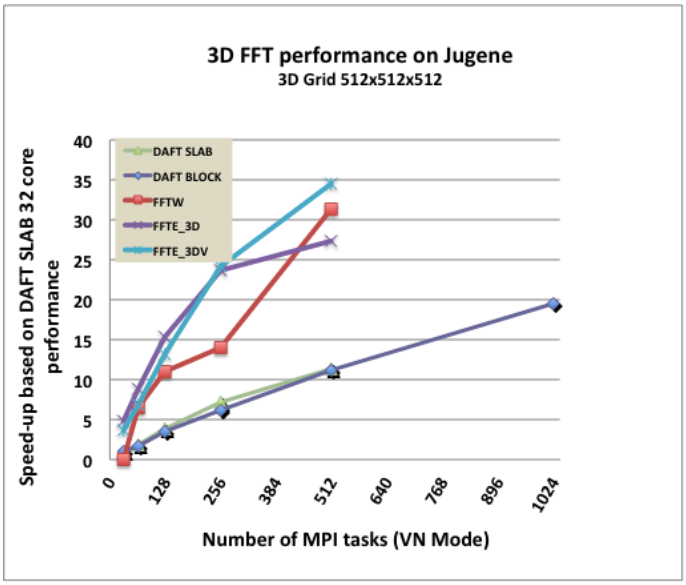


Figure 24: 3D FFT performance on JUGENE

Best Practices: The scalability of parallel 3-d FFTs remains inherently limited, owing to the all-to-all communications involved. Likewise the variety of data decompositions supported by the available libraries is also limited. While this remains true, we will continue to see FFT-dependent applications using custom parallel FFTs with bespoke communications, and little re-use of library code, often restricted to serial or threaded FFTs within a single MPI process. It is clear that exploiting shared memory within a node can help improve the scalability of slab and pencil decompositions, and there is some evidence that hybrid MPI/OpenMP implementations in some of the libraries considered are benefitting from this.

Given the current state of affairs, it is difficult for application developers to rely on 3rd party libraries to achieve portable and scalable FFT performance. As the core count increases, it is necessary to switch from slab to pencil and perhaps even to block decompositions to get the best performance, but the cross-over points depend heavily on problem size and system characteristics.

For the developers of DL_POLY, it is likely that DAFT is not the best choice at all problem sizes and core counts. Certainly, DL_POLY can hope to benefit more from shared memory, and the current work on a hybrid MPI/OpenMP implementation is well motivated. Whether it is worthwhile to modify DL_POLY so that it can exploit 3rd party parallel FFT libraries when these are favourable is a decision they will have to make.

3.6 FFT Library Performance on PRACE Systems – Quantum ESPRESSO

Contributors: Vladimir Slavnic (IPB), Milos Nikolic (IPB), Aleksandar Jovic (IPB), Josip Jakic (IPB)

Code(s): Quantum ESPRESSO

White Paper, Technical Report or Scientific Publication: Andrew Sunderland, Stephen Pickles, Ivan Girotto, Peter Nash, Michael Lysaght, Milos Nikolic, Aleksandar Jovic, Josip Jakic, and Vladimir Slavnic. *"FFT Library Performance on PRACE Systems", 2012.*

Application Area(s): Condensed Matter Physics, Computational Chemistry

Algorithm(s) under investigation: Fast Fourier Transforms

Parallelization Technique(s): MPI, OpenMP, and Hybrid

Numerical Library used:

FFTE 5.0 (MPI/OpenMP), (http://www.ffte.jp/). Routines used: FORTRAN -PZFFT1D, PZFFT2D, PZFFT3D (MPI/OpenMP), ZFFT1D, ZFFT2D, ZFFT3D (OpenMP)

FFTW3.3 (MPI/OpenMP), (http://www.fftw.org/). Routines used: C- dfftw_plan_many_dft, fw_planz, dfftw_destroy_plan, dfftw_execute_dft, fftw_execute, fftw_mpi_plan_dft_1d/2d/3d, fftw_plan_dft_1d/2d/3d, fftw_malloc, fftw_mpi_local_size_1d/2d/3d, fftw_destroy_plan, fftw_free

Internal Quantum Espresso FFTW, (http://www.quantum-espresso.org/)

Tools Employed: TotalView Debugger

Project's Web Page & Numerical Libraries web pages

Hardware Platforms Investigated: CURIE, PARADOX Cluster at IPB

Short description of Any Benchmark Datasets: FFT testing was performed on complex array of varying sizes (up to 2^30). That size is significant, because it is the largest array that fits into CURIE memory limits.

Project Description: The objective of this project is to compare performance and scalability of various implementations of FFT using specific scientific applications as the base code together with different FFT libraries. Implementations of FFT are ranging from OpenMP versions for multi-core shared memory systems to pure MPI and Hybrid versions for systems with distributed memory. Scientific application in question is Quantum Espresso.

Contribution: Benchmark of Quantum Espresso (QE) has been performed using internal QE FFTW copy and FFTW3.3 library on CURIE. For these purposes FFT benchmark code has been developed based on QE domain decomposition. For 3-d FFTs on CURIE performance comparison between FFTW3.3 and internal copy of FFTW library has been performed using the data from QE distribution. A comparison of FFTW3.3 and FFTE libraries has been performed on CURIE for different types of FFT calls: 1-d, 2-d, 3-d FFT (MPI and OpenMP), by using in-house developed benchmarks. Development of hybrid custom test code for FFTE library is in progress. 3-d hybrid FFTE benchmark code is developed but still has not been tested. FFTE developer has been contacted to clarify several identified issues with 1-d and 2-d hybrid FFTE code. Hybrid code for FFTW is developed and fully functional for any number of dimensions.

Main Performance Bottlenecks: With the used datasets we observed severe bottleneck in performing parallel FFTs. Another problem with the FFTE library package is memory allocation management since most of its code is written in Fortran77. Several possible solutions are being considered. Lack of documentation for FFTE, and to some extent for FFTW also posed a problem.

Application Optimization Work Done: Creation of a single FFTE library from the FFTE package. Full incorporation of this library into the QE benchmark code is in progress.

Performance Results: Although scalability for both libraries was far from expected, we found that the MPI implementation of FFTE 5.0 outperforms the MPI implementation of FFTW 3.3 when used for 3-d FFTs, while for 1-d and 2-d FFTs both libraries perform similarly.

Initial QE benchmark tests (on 2, 4, 8, 16, 32, 64, and 128 cores) show that, for the used datasets, 3-d FFT is performed faster when using an internal copy of FFTW compared to the external FFTW 3.3 library. However, further tests are required.

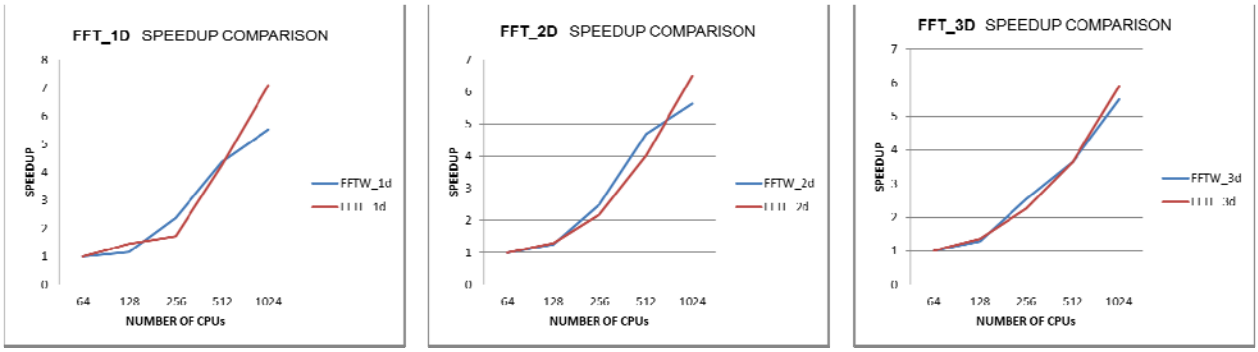


Figure 25: Scalability analysis of 1D, 2D, and 3D FFTs using FFTW and FFTE libraries

Best Practices: The observed scalability issues of all considered FFT libraries and implementations present a problem for QE performance. Further work is needed to clarify the scalability for different relevant datasets.

3.7 Research Highlights and Conclusions

The projects described here have all addressed some of the most important computational problems in a range of PRACE application codes. Due to the commonality of many of the algorithms underpinning parallel scientific application codes, these findings should also be of great interest to many other groups of scientific software developers. A particular highlight has been the analysis of FFT performance in the DL_POLY and Quantum ESPRESSO applications, described in the projects in Sections 3.5 and 3.6. This has involved a collaboration from three different institutions - STFC, IPB and ICHEC – with its aim to investigate both different FFT libraries (FFTW, FFTE, native) and different programming paradigms (MPI/OpenMP/GPU-based). The stand-alone performance results of the FFT routines must also be assessed in relation to the level of disruption involved in incorporating top-performing library routines into the existing parallel application and consequences of these changes to other parts of the code. The Task 7.5 work on DL-POLY and Quantum ESSPRESSO are part of larger-scale projects being managed and delivered in 7.2.

Another notable project is the LSDALTON parallelisation project, reported in Section 3.2. This has taken as its start point the serial code LSDALTON and has parallelised it from scratch in a relatively short period of time using ScaLAPACK library routines as the cornerstones for the parallelisation.

Finally, a huge computation bottleneck in many chemistry and physics codes is the symmetric eigensolver calculation and the results from Section 3.1 demonstrate that ELPA is worth considering as an alternative to ScaLAPACK, especially given the non-disruptive nature of the swap.

3.8 Brief Summary of Main Libraries Discussed

1. **ScaLAPACK** - The ScaLAPACK (or Scalable LAPACK) library includes a sizeable subset of LAPACK routines redesigned for distributed memory MIMD parallel computers. It assumes matrices are laid out in a two-dimensional block cyclic decomposition, http://www.netlib.org/scalapack.
2. **ELPA** – The Eigenvalue Solvers for Petaflop Applications library has been designed for the computation of a subset or all of the eigenpairs of a symmetric Hermitian System, http://elpa.rzg.mpg.de.
3. **LAPACK** - The Linear Algebra PACKage provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision, http://www.netlib.org/lapack.
4. **PLASMA** – The Parallel Linear Algebra Software for Multicore Architectures provides routines to solve dense general systems of linear equations, symmetric positive definite systems of linear equations and linear least squares problems, using LU, Cholesky, QR and LQ factorizations, http://www.netlib.org/plasma.
5. **MAGMA** – The Matrix Algebra on GPU and Multicore Architectures project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems, http:// http://icl.cs.utk.edu/magma.
6. **PETSc** - The Portable, Extensible Toolkit for Scientific computation provides sets of tools for the parallel (as well as serial), numerical solution of PDEs that require solving large-scale, sparse nonlinear systems of equations, http://www.mcs.anl.gov/petsc.

7. **FFTW** - The Fastest Fourier Transform in the West is a C subroutine library for computing the discrete Fourier transform in one or more dimensions, of arbitrary input size, and of both real and complex data, http://www.fftw.org.
8. **FFTE** – The Fastest Fourier Transform in the East is a package to compute Discrete Fourier Transforms, developed in Japan, http://www.ffte.jp.

4 Multi-core/Many-core Systems

Strong development of new CPU models at higher and higher clock speeds and at increased levels of miniaturization over several decades has led to the saturation of performance of chips based on a single-CPU design by mid-2000s. The major vendors, first AMD and then also Intel, addressed this significant problem by developing dual-core CPUs as of 2005. This idea turned out to be step in the right direction, and further development led to today's multi-core systems, which now include CPUs with tens of cores. Integrated on the same motherboard, several multi-core CPUs are now typically present in computing nodes of high-performance computing facilities and supercomputers, therefore leading to individual nodes comprising as much as several tens of CPU cores. This trend is likely to continue, and vendors now have in R&D phase's many-core CPUs, comprising several tens to over hundreds of cores.

Since their inception, multi-core CPUs have presented a programming challenge for a majority of applications. Due to frequent changes in the architecture of multi-core CPUs, in particular in their interconnect topology, cache distribution and sharing model, access to memory, many applications were not able to fully benefit from computing power nominally provided by such CPUs. The problem is even more challenging when we consider massively parallel applications, executed on thousands or tens of thousands CPUs.

The Message Passing Interface (MPI) paradigm, which is natural for homogeneous clusters, can be used for systems comprised of multi-core computing nodes. However, such an approach is obviously far from being optimal, since individual computing nodes are already small SMP systems with tens of CPU cores, and therefore different programming paradigm may be much more suitable within the node. A thread-based approach can be more efficient, leading to elimination of overheads related to MPI-style communication on shared memory systems. For instance, OpenMP paradigm is typically used to optimize performance of applications suited for shared memory systems.

The hybrid approach, combining MPI programming paradigm across computing nodes and OpenMP thread-based approach within individual nodes, is therefore seen as a method of choice for optimizing the performance of massively parallel applications on large-scale high performance infrastructures. Conceptually, porting an existing MPI application to the hybrid MPI/OpenMP programming model is easy, since OpenMP relies on preprocessing directives.

In many cases, however, introducing threads is not straightforward, and can even lead to degradation of performance. The MPI standard and its implementations define four levels of thread safety: MPI THREAD SINGLE, where only one thread of execution exists; MPI THREAD FUNNELED, where a process may be multithreaded but only the thread that initialized MPI makes MPI calls; MPI THREAD SERIALIZED, where multiple threads may make MPI calls but not simultaneously; and MPI THREAD MULTIPLE, where multiple threads may call MPI at any time. The use of MPI THREAD FUNNELED is the easiest choice, but can be far from optimal with large number of threads per MPI process. On the other hand, performance issues plague implementations of a more natural MPI THREAD MULTIPLE mode and, while it can be expected that its use could benefit application's performance, in practice it also requires significant work on refactoring application's structure and its data distribution. The programming model becomes even more complex if we add to the existing hierarchy GPGPUs, which are now typically integrated into compute nodes in newly designed and procured Tier-0 and Tier-1 systems. However, here we will not consider this, and instead will focus only on hybrid MPI/OpenMP programming model.

The next sections briefly summarize the individual projects done as part of sub-task 7.5.D. The focus of the first 4 projects is petascaling-enabling on multi-core/many-core systems for applications supported by tasks 7.1 and 7.2. The next 4 projects evaluate hybrid approach for

selected scientific computing applications, and one final project addresses MPI communication latency issues in the hybrid programming model. Each summary gives details on technical whitepapers or scientific publications (where available). The chapter concludes with a high-level discussion on the topic of hybrid programming paradigm for multi-core/many-core systems.

4.1 Petascaling enabling and support for Gromacs

4.1.1 Project summary

Project authors: Berk Hess (KTH), Erik Lindahl (KTH)

Contributors: Fabio Affinito (CINECA), Andrew Emerson (CINECA), Leandar Litov (NCSA), Peicho Petkov (NCSA), Michael Schliephake (KTH), Lilit Axner (KTH), Maria Francesca Iozzi (UiO)

Publications:

PRACE technical whitepaper: "Performance Analysis and Petascaling Enabling of GROMACS" , Fabio Affinito, Andrew Emerson, Leandar Litov, Peicho Petkov, Rossen Apostolov, Lilit Axner, Berk Hess, Erik Lindahl and Maria Francesca Iozzi

Applications area(s): Life Science

Hardware platform(s): JUGENE, PLX cluster at CINECA, Lindgren (Cray XE6) at KTH

Programming language(s): C/C++

Profiling/debugging tools: Paraver

Libraries: FFTW

Brief project description:

This project is a collaborative work between partners from Task 7.5 and Task 7.2. The tight collaboration between the partners resulted in a common work plan, where Task 7.5 partners just used their PMs to contribute to the agreed work plan. GROMACS is an important code in the ScalaLife project which intends to build a cross-disciplinary Competence Centre for life science software [8]. The collaboration between PRACE partners and ScalaLife members was very beneficial, especially because work done and experience gained in the ScalaLife project could be leveraged by the PRACE Task 7.2 and Task 7.5 work. The work within PRACE aims at evaluating the performance of GROMACS on different platforms and implementing new strategies to enable the code for petascaling molecular dynamics simulations. The activities have been organized into three tasks: (i) Optimization of GROMACS performance on BlueGene systems; (ii) Parallel scaling of the OpenMP implementation; (iii) Development of a multiple step-size symplectic integrator adapted to the large biomolecule systems. Main goal of the Task 7.5 effort was the evaluation of the hybrid version of GROMACS and comparison with the MPI only version.

4.1.2 Hybridization work

Parallel versions of GROMACS up to v4.5 use MPI only to handle the communications between the processor cores. As part of the ScalaLife project, the code developers released a hybrid MPI-OpenMP version (v4.6) where multithreading has been applied to the PME calculations and so may give performance increases on common SMP clusters. The aim of this work was to test this version by performing benchmark runs on two separate

architectures: the PLX Intel cluster at CINECA and the BlueGene/P system JUGENE in Juelich.

The hybrid 4.6 version of the code was downloaded from the site of ScalaLife and installed on both architectures according to the standard procedures provided in the ScalaLife documentation and the GROMACS v4.5 manual. On PLX, simulations were run for a water system (containing 12k atoms) using 1 or 2 MPI processes/node and up to 12 threads/node. Since these benchmarks aimed at checking the performance behaviour in very simple cases, the default '-npme' parameter provided by mdrun was used. Further investigations will check the influence of the fine-tuning of this parameter on the results.

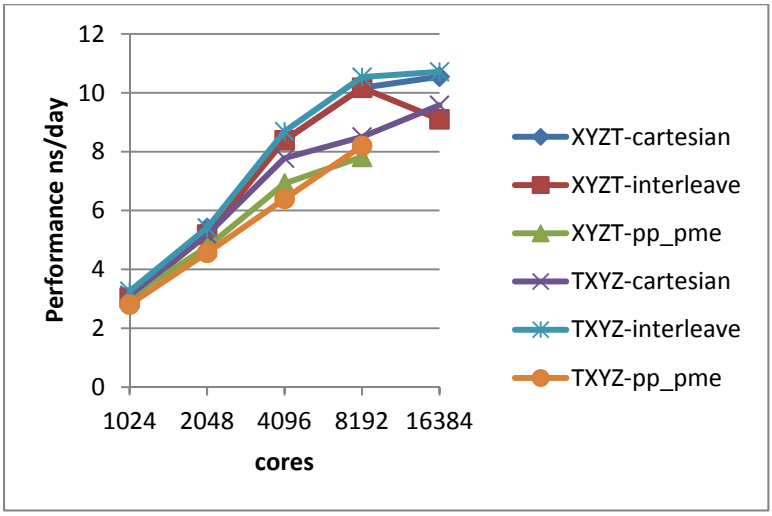


Figure 26: Gromacs performance as a function of BG/P mapping topology and different domain decomposition mode.

This work contains an extensive study of the performance of several versions of GROMACS on a diverse set of computer architectures with varying runtime conditions and program options. The aim has been to determine the optimal set of conditions for a given architecture and input - without making changes to the code itself. In the first study on JUGENE, we measured performance as a function of the topology scheme adopted at runtime and the particle-particle- and PME partitioning strategy used in GROMACS. It was found that, although there are differences in performance at high node counts, they are not particularly significant and are difficult to rationalize in terms of the options used. In fact these results illustrate the fact that on systems with very high core counts, e.g. JUGENE, the all-to-all communication scheme of the PME calculation will limit parallel scaling quite quickly. This does not mean that users should not attempt to optimize the run conditions on BlueGene-type architectures, but rather that a priori it is difficult to gauge which set of conditions would work best.

The second set of studies investigated the hybrid MPI/OpenMP version of GROMACS (version 4.6) where OpenMP threads are used for the PME calculation. We performed one set of studies on CINECA's PLX cluster comparing the hybrid version against the MPI-only version but saw little difference in performance. The fact that those runs used benchmarks that run on only up to 40 nodes and the fast Infiniband network of the cluster, are a reason for this. In fact, the documentation of the hybrid version points out those differences would be most remarkable for high node counts or for clusters with low network bandwidths. To test the influence of the number of nodes and the interconnect, further simulations were performed on the AMD Opteron system Povel and the CRAY XE6 cluster (Lindgren) at PDC. It was found that the GROMACS 4.6 release brings on average a 10% performance increase for simulations using PME method for electrostatics. For simulations that don't use PME, such as the virion system, the new release does not have an effect. The hybrid-OpenMP/MPI mode

however significantly improves the scaling of large systems on a large number of cores, typically for more than 200.000 particles, on more than 500 cores. On systems with fast interconnects such as the Cray XE6, the hybrid mode doesn't offer much advantage.

4.2 Petascaling enabling and support for Dalton

4.2.1 Project summary

Project authors: Simen Reine (UiO), Trygve Helgaker (UiO), Thams Kjeargaard (UiO), Olav Vahtras (KTH), Zilvinas Rinkevicius (KTH), Bogdan Frecus (KTH)

Contributors: Andrew Sunderland (STFC), Thomas Keal (STFC), Michael Schliephake (KTH), Xavier Aguilar (BSC), Lilit Axner (KTH), Maria Francesca Iozzi (UiO), Judit Gimenez (BSC)

Publications:

PRACE technical whitepaper: "Petascaling and Performance Analysis of Dalton on Different Platforms"

Applications area(s): Life Science

Hardware platform(s): CURIE, Lindgren at KTH and local clusters at UiO and STFC

Programming language(s): Fortran and C

Profiling/debugging tools: Paraver

Libraries: MPI

Brief project description:

Work on the Task 7.2 community code Dalton was an intense collaboration between Task 7.2, 7.4 and 7.5 partners. KTH, BSC, UiO and STFC partners used their 7.5 PMs to contribute to the agreed work plan. Similar to GROMACS, Dalton is part of the ScalaLife project [8]. The work in PRACE aims at evaluating the performance of Dalton on different platforms and implementing new strategies to enable the code for Petascaling. The activities have been organized into four tasks: (i) Analysis of the current status of the Dalton code and identification of bottlenecks, implementation of several performance improvement of Dalton Quantum Mechanical Model (QMM) and first attempt of hybrid parallelization; (ii) Implementation of MPI integral components into LS Dalton, improvements of optimization and scalability, first attempt of a hybrid density functional theory (DFT) ; (iii) Implementation of one or more interfaces between ChemShell and DALTON QMM/LS-DALTON as well as evaluate the performance of QM/MM calculations with ChemShell/DALTON for large-scale benchmark calculations; (vi) Analysis of the impact of Dalton QMM system components with Dimemas. Main goals of Task 7.5 efforts were the hybridization of Dalton QMM and the implementation of the hybrid DFT.

4.2.2 Hybridization work

We implemented a new flexible task manager, in which the manager makes a partitioning based on the time estimates of the various contributions in order to achieve first a better load balancing and second to allow a fully distributed scheme. We furthermore worked on the time estimates based on the screening integrals.

Then, we implemented an MPI buffer for improved MPI broadcasting, tested the code and identified some overheads resulting in poor scaling (lstensor) and automated the

profiling/scalability testing. We greatly improved the OpenMP performance. This is important for final implementation of hybrid DFT MPI/OMP scheme.

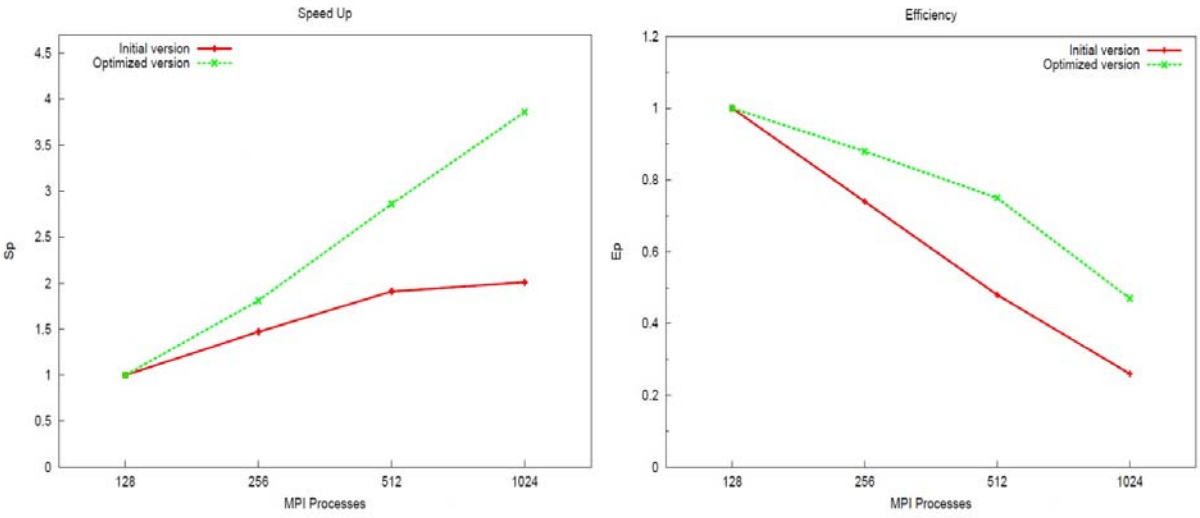


Figure 27: Speedup (left) and efficiency (right) comparison between initial and optimized version of DALTON

A lot of work was necessary to prepare for hybridization. The original Dalton version scaled until 512 cores and provided so far a parallel efficiency of around 50% (see Figure 27) whereas the optimized version using the teams of masters has an efficiency of 80% for 512 processes. Moreover, it also scales up to 1024 cores providing an efficiency of around 50%. Work on Dalton is still on-going and final results will be available in the Dalton white paper.

4.3 Petascaling enabling and support for EC-Earth3

4.3.1 Project summary

Project authors: John Donners (SARA)

Contributors: Chandan Basu (LiU), John Donners (SARA) et al.

Publications:

Chandan Basu, "High Resolution EC Earth Porting, Benchmarking on Curie" PRACE technical whitepaper

Applications area(s): Meteo-climatology

Hardware platform(s): CURIE

Programming language(s): C, FORTRAN, MPI

Profiling/debugging tools: TAU profiling tool, strace

Libraries: netCDF, Intel MKL

Brief project description:

EC-EARTH is an earth system modelling application. The three main components of EC-EARTH are IFS (for atmosphere), NEMO (for ocean) and OASIS (for coupling). We ported, benchmarked and analysed a high resolution version of the EC-EARTH configuration. The ocean component used for this configuration is ORCA025 which is a 1/4° global configuration. The atmosphere component uses T799 resolution. The scaling studies were performed on CURIE. As EC-EARTH is a large and complex program it's porting on a new

machine is a considerable challenge. The goal is to test the scalability of this model and to determine and analyse the bottlenecks.

4.3.2 Hybridization work

EC-EARTH is made up of three different components. These components have quite different build environments. The EC-EARTH package provides a semi-automatic utility called eexcon. The important thing in the EC-EARTH compilation is to set up an appropriate xml file with description of system specific settings for package paths, compilers, flags, libraries etc. We used Intel compiler version 12.1.7.256, bullxmpi version 1.1.8.1 and netCDF version 4.1.1 in our compilation.

EC-EARTH compilation produces multiple binaries, one each for NEMO, OASIS and IFS. The bullxmpi on CURIE is capable of launching MPMD runs. For efficient runs it is needed that IFS, NEMO and OASIS processes are not mixed up in the same node. We have created specialized scripts for efficiently submitting jobs on targeted nodes. We run high resolution EC-EARTH up to 3500 MPI processes. At this scale the efficiency is ~50% of the most efficient run. The most efficient run is obtained at ~1034 MPI processes. When EC-EARTH is run with less than 1034 MPI processes the efficiency is lower. This is because for less than 1034 processes NEMO processes finish earlier and wait for IFS processes.

We profiled EC-EARTH with TAU. TAU instruments the source code with lightweight timers and then compiles and links. Both MPI functions and user functions are profiled with TAU, this profiling has negligible overhead. We observed the I/O access pattern of EC-EARTH through strace. This shows that few small files are frequently opened and read.

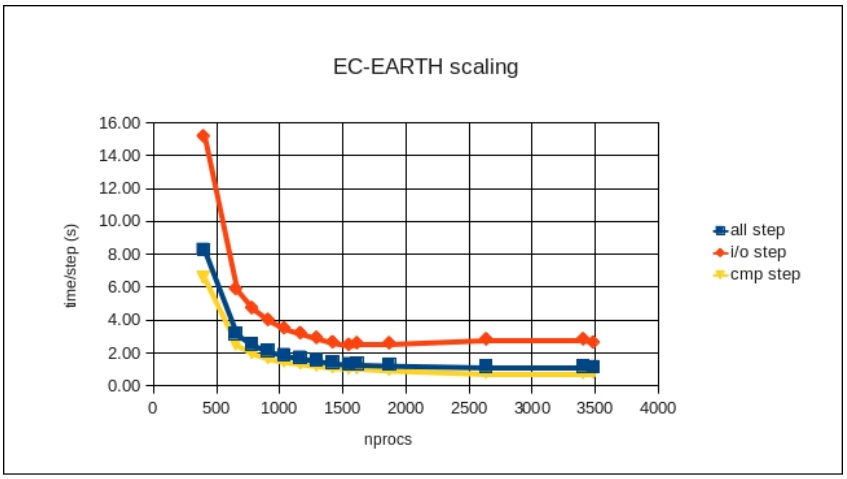


Figure 28: EC-EARTH scaling analysis

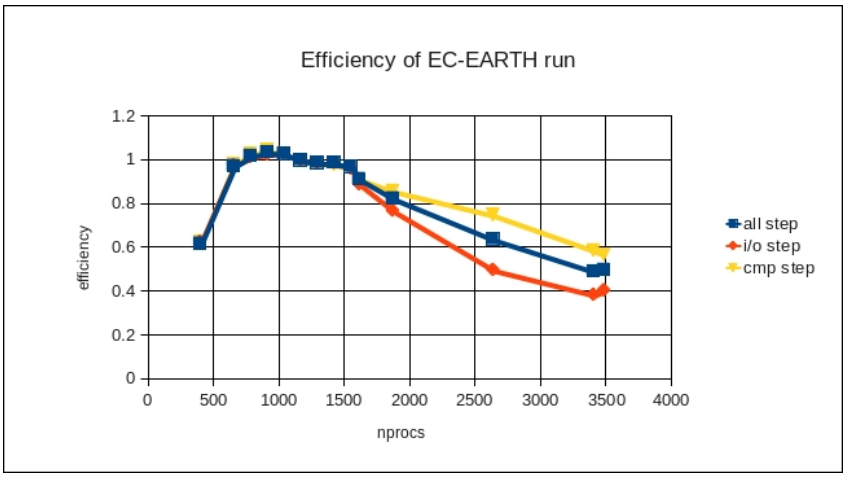


Figure 29: EC-EARTH efficiency analysis

4.4 Petascaling enabling and support for Elmer

4.4.1 Project summary

Project authors: Georgios Goumas (ICCS), Konstantinos Nikas (GRNET)

Contributors: Vasileios Karakasis (ICCS), Nectarios Koziris (ICCS), Peter Råback (CSC)

Publications:

Vasileios Karakasis, Georgios Goumas, Konstantinos Nikas, Nectarios Koziris, Juha Ruokolainen, and Peter Råback, "Using State-Of-The-Art Sparse Matrix Optimizations for Accelerating the Performance of Multiphysics Simulations", accepted for PARA 2012: Workshop on State-of-the-Art in Scientific and Parallel Computing

Applications area(s): Computational Engineering, Computational Fluid Dynamics, Earth and Climate Science

Hardware platform(s): 24 nodes of 2-way quad-core Intel Xeon E5405 processors, 1 Gb/s Ethernet

Programming language(s): C, C++

Profiling/debugging tools: none

Libraries: Boost, LLVM, Clang

Brief project description:

Scaling applications for Peta- or Exascale systems is very challenging as large classes of applications do not scale either with the number of nodes, the number of cores within a single node, or, even worse, with both. This project focuses on scalability issues arising within a single node. It integrates the Compressed Sparse eXtended (CSX) sparse matrix storage format into the Elmer multi-physics software package. This is, to our knowledge, one of the first approaches of evaluating the impact of an innovative sparse matrix storage format on a "real-life" production multi-physics software. CSX aims at improving the performance of the Sparse Matrix-Vector Multiplication (SpMxV) kernel, which is used by the solver for 60-90% of its total execution time.

4.4.2 Hybridization work

Elmer allows a user to specify a library to perform the SpMxV. Its interface however, does not capture the notion of tuning the sparse matrix representation. So, ICCS changed the interface to include a pointer to the tuned matrix representation. The CSX matrix is constructed in two phases: first, an internal representation is built from the initial CSR matrix; then this representation is scanned for substructures and the final CSX matrix is produced.

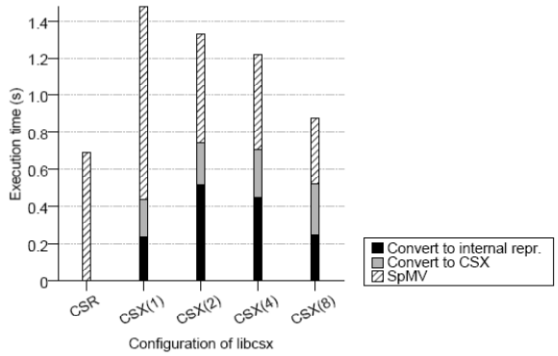


Figure 30: Libcsx's execution time breakdown for vortex3d before preprocessing optimizations (46 SpM×V calls)

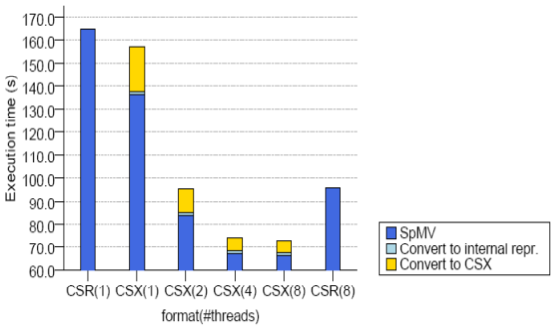


Figure 31: Libcsx's execution time breakdown for vortex3d after preprocessing optimizations (2000 SpM×V calls)

Figure 30 shows the execution time breakdown of our library, *libcsx*, when using one or more cores on a dual SMP quad-core Intel Xeon E5405. Two key observations can be made: first, the time required to convert CSR to the internal data structure takes up more than half of the preprocessing time; second, the time to build CSX does not scale with the number of cores, despite being multithreaded.

Detailed profiling of the code revealed that the problem was caused by the memory allocations, performed both during the construction of the internal representation and the construction of the final CSX matrix. Figure 31 shows the execution time breakdown on the same platform after addressing the aforementioned bottlenecks. Compared to the initial implementation, the preprocessing cost is now reasonable and consistent. The conversion time is now minimal and constant across the different thread configurations, while the rest of the preprocessing time scales reasonably with the number of threads.

As Elmer's CSR implementation is single-threaded, we created a multithreaded version to ensure a fair comparison. The experimental platform consisted of 192 cores (24 nodes of two-way quad-core Intel Xeon E5405 processors interconnected with 1 Gb/s Ethernet) running Linux 2.6.38. GCC 4.5 along with LLVM 2.9 was used for the runtime code generation for CSX. The size of the problems used for the evaluation was increased to be adequately large for our system. We opted for sizes leading to matrices larger than 576 MiB, which is the aggregate cache of the 24 nodes. Finally, a simple Jacobi (diagonal) preconditioner was used for all the tested problems.

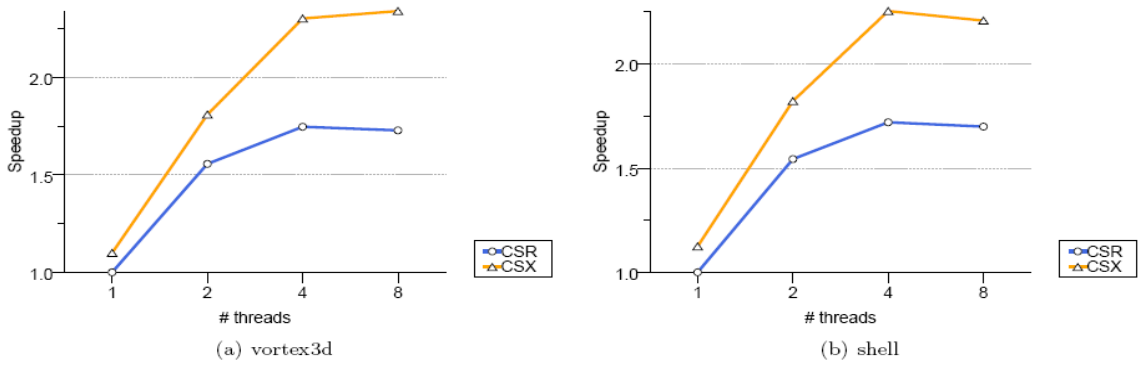
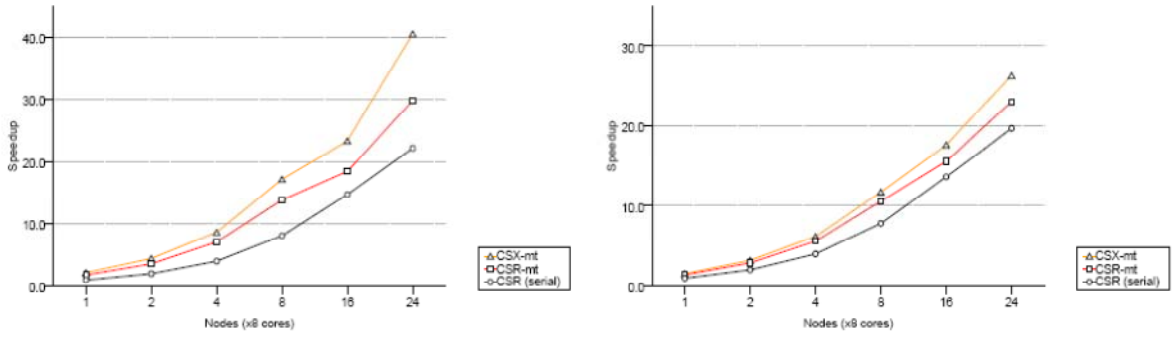


Figure 32: Intra-node speedup (1000 linear iterations)

Figure 32 presents the speedup achieved within a single node. For each thread count, the most favourable thread configuration for the SpMxV kernel, i.e., the one with the least possible sharing of the highest-level caches was selected. This placement achieves the highest

performance with the least possible threads and explains the plateau of the speedup encountered by both CSR and CSX in the 8-thread configuration, as in this case, the contention in the common bus becomes apparent. In both problems, CSX performs considerably better than CSR, despite its preprocessing cost. More specifically, CSX achieves a performance improvement of around 35% over CSR.



(a) Speedup of the total execution time spent inside the SpMxV library, including the preprocessing cost in the case of CSX.

(b) Speedup of the total solver time.

Figure 33: Average speedup of Elmer up to 192 cores using CSX library (1000 linear system iterations)

Figure 33 shows the average speedups achieved by simply the SpMxV code (Figure 33(a)) and the total solver time (Figure 33(b)) using the original Elmer CSR, our multithreaded CSR and CSX (including its pre-processing cost). CSX achieves a significant performance improvement of 37% over the multithreaded CSR implementation, which translates to a noticeable 14.8% average performance improvement of the total execution time of the solver. We believe that this improvement could be even higher if other parts of the solver exploited parallelism within a single node as well, since the SpMxV component would become then even more prominent, allowing a higher performance benefit from the CSX optimization. Regarding the preprocessing cost of CSX, it was fully amortized within 224–300 linear system iterations.

To summarize, scaling applications for peta- or exascale systems is very challenging as large classes of applications do not scale either with the number of nodes, or with the number of cores within a single node, or, even worse, with both. This work focused on scalability issues arising within a single node. More specifically, it integrated the CSX format into the Elmer multi-physics software package. CSX was found able to improve the performance of the SpMxV component by nearly 35% compared to the multithreaded CSR when executed on a single node and offered a 15% overall performance improvement of the solver in a 24-node, 192-core SMP cluster.

4.5 Evaluating the hybrid approach for HYDRO

4.5.1 Project summary

Project authors: Philippe Wautelet (IDRIS), Pierre-François Lavallée (IDRIS)

Contributors: Philippe Wautelet (IDRIS), Pierre-François Lavallée (IDRIS)

Publications:

PRACE technical whitepaper: ”HYDRO” by Pierre-François Lavallée, Guillaume Colin de Verdière, Philippe Wautelet, Dimitri Lecas, Patrick Corde, Jean-Michel Dupays

Applications area(s): Computational Fluid Dynamics

Hardware platform(s): JUGENE, CURIE

Programming language(s): Fortran 90

Profiling/debugging tools: none

Libraries: MPI+OpenMP

Brief project description:

HYDRO is a 2D Computational Fluid Dynamics code (~1500 lines) that solves Euler's equations with a Finite Volume Method using Godunov's scheme and a Riemann solver at each interface on a regular mesh. The goal of this project was to study the impact of building and running a hybrid MPI/OpenMP version of the code in terms of complexity of development, performance, scalability, memory usage, memory affinity, binding, thread level support (MPI_THREAD_FUNNELED, MPI_THREAD_MULTIPLE) and maturity level of debugging and performance tools.

4.5.2 Hybridization work

Starting from a pure MPI version and an OpenMP parallel version, both of which were already available, we combined the two approaches to create a hybrid MPI+OpenMP version of HYDRO. For reasons of portability and simplicity, the MPI_THREAD_FUNNELED level of thread support was used. Porting of the code to the MPI+OpenMP paradigm turned out to be straightforward; the additional porting/coding effort being very limited compared to the time needed to develop the pure MPI or the OpenMP version.

We used a 2D domain decomposition for the MPI level and a coarse-grained OpenMP parallelization. Note that obtaining any gain in term of performance or scalability by using a hybrid version for this kind of code is not obvious. In fact, the MPI version is already very well balanced up to several thousand cores and the communication scheme is nearly optimal as it involves only neighbourhood point-to-point communications except for the time-step computation, which involves a global reduction. The strong scaling results presented here were run on the Blue Gene/P system at IDRIS. From small to relatively high number of cores, the performances of pure MPI or MPI+OpenMP approaches are very similar, but once the number of cores is over 4096, the pure MPI implementation begins to lose scalability, whereas the hybrid approach keeps a near perfect scalability up to 16384 cores and even continues to scale up sub optimally to 32768 cores.

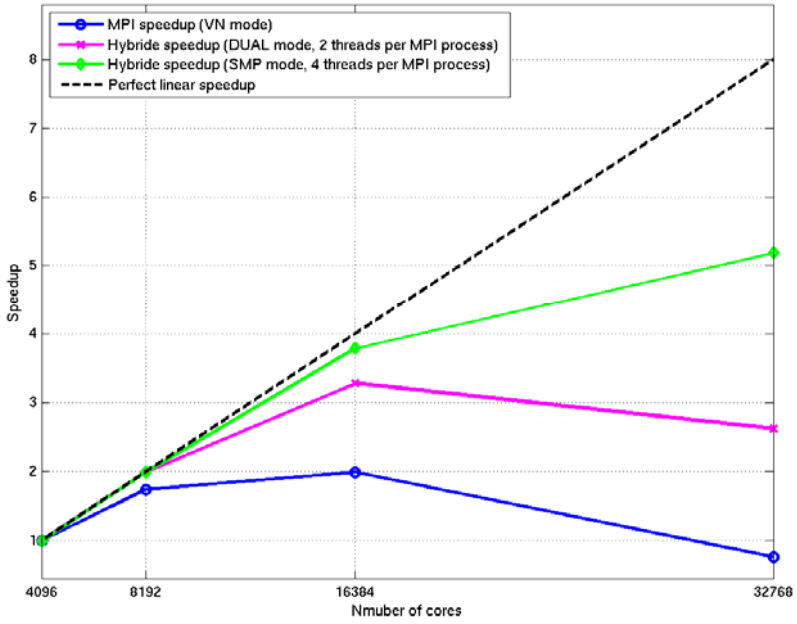


Figure 34: Scaling analysis of the HYDRO code (weak scaling, speedup based on the execution time of MPI version on 4096 cores)

The best scalability can be achieved on homogeneous petascale architectures. It pushes scalability limitations far beyond those of the pure MPI approach. The two levels of parallelism perfectly fit the hardware characteristics of various machines (either fat or thin nodes).

Hybrid parallel programming, mixing MPI and OpenMP, preserves the advantages of the two approaches. Scalability is better by a reduction of both the number of MPI messages and the number of processes involved in collective communication and by a better load balancing. On the other hand the total memory consumption is optimized through the OpenMP shared-memory approach, savings in replicated data in the MPI processes and in the used memory by the MPI library itself.

Concerning the thread level support, MPI_THREAD_FUNNELED level is much simpler to use than the MPI_THREAD_MULTIPLE. Creating an MPI_THREAD_FUNNELED hybrid version starting from the available MPI and OpenMP parallel versions is straightforward. The MPI implementation of MPI_THREAD_MULTIPLE thread level support is not always optimal, leading on some machines to poor scalability. In these cases, the lower level MPI_THREAD_FUNNELED is preferable.

However, the necessity of using two explicit levels of parallelization increases both the number and the complexity of bugs in the code. Binding, memory affinity or optimal number of threads per MPI process, are parameters that can have a huge impact on performance and scalability of the code. The quality of the OpenMP parallelization is essential to get a hybrid version that is scalable and efficient. Debugging and performance analysis tools exist, but unfortunately they are still immature and not as robust as those for pure MPI applications. Therefore it seems of little use to apply a hybrid parallelization scheme to codes that have neither scalability limitations nor a huge memory usage.

4.6 Implementation of improved hybrid parallelization on Vlasiator

4.6.1 Project summary

Project authors: Sebastian von Alfthan (FMI, Finland)

Contributors: Dusan Stankovic (IPB), Vladimir Slavnic (IPB)

Publications:

Sebastian von Alfthan, Dusan Stankovic, Vladimir Slavnic, "Scaling Vlasiator using Hybrid MPI and OpenMP parallelization", PRACE technical whitepaper

Applications area(s): Astrophysics/Earth and Climate Science

Hardware platform(s): CURIE, PARADOX at IPB

Programming language(s): C++

Profiling/debugging tools: Scalasca

Libraries: Boost (Program Options, MPI), Zoltan, MPI+OpenMP

Brief project description:

Vlasov-hybrid simulation (Vlasiator) is a simulation where electrons are fluid and ions are distribution functions, enabling the description of multi-component plasmas without noise and on scales unreachable by existing techniques such as magnetohydrodynamic (MHD) simulations. The work on this project is related to performance improvement of the hybrid (MPI/OpenMP) version of the code that will show better scaling than a pure MPI one.

4.6.2 Hybridization work

IPB received an already hybridized MPI+OpenMP code from FMI. Their task was to examine simulation performance and to possibly improve the performance of the hybrid version. Scalasca was used for profiling, mostly to test the behaviour of some specific code parts. However, since Vlasiator itself has a detailed time logging module, values shown in the benchmarks are taken from the internal Vlasiator log files.

Tests were run on 512 and 1024 total cores, the number of the threads per MPI process was increased from 1 to a maximum of 32. The code was compiled both with the Intel icpc compiler and GNU g++. Measured performance shows that Vlasiator has a computation versus communication ratio of approximately one, which means that even a small increase in communication performance o could significantly increase total simulation throughput. The optimal ratio of processes/threads per compute node was shown to be 4/8 (without usage of SMT), and weak scaling tests showed that the simulation throughput per core doesn't decrease much as the number of cores (and thus the simulation data volume) increases from 512 to 1024 (in fact, it decreases from 7.32 cells/s per core to 7.18).

The extensive profiling revealed that the main bottlenecks are not related to threading, therefore the optimization work on the hybrid part of Vlasiator could not be performed to the extent planed. The main bottleneck is actually bad I/O scalability, which can be partially amortized by raising number of threads per process, thus lowering total number of MPI processes, which on the contrary reduces network saturation, and increases communication time. Tests showed that 4 processes per node are needed to saturate the available InfiniBand bandwidth, because only the master thread is allowed to call routines for MPI communication.

One of the problems experienced was that the profile files generated by Scalasca for Intel and GNU compilers could not be compared, because the instrumentation of GNU compiled code

was unsuccessful. Since the Intel compiled code showed a slightly better performance, the idea was to show where exactly this difference in performance comes from. Another problem was the lack of an MPI implementation on CURIE, which supports the MPI_THREAD_MULTIPLE communication mode. Such implementation could possibly allow better utilization of InfiniBand network, thus lowering time spent in communication and improving performance, as explained in the previous paragraph. In case such a library will become available until the end of PRACE-1IP, additional results will be provided in the deliverable D7.1.2. More details on the above mentioned issues, and also a detailed distribution of time between computation, communication and I/O operations can be found in the Vlasiator whitepaper.

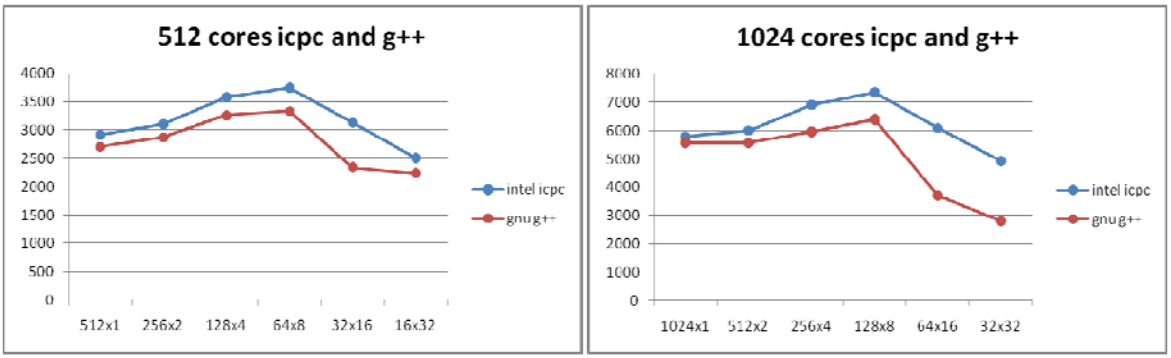


Figure 35: Performance (Cells/s) on CURIE with total of 512 and 1024 cores with 64 cells per core, and variable number of processes and threads (in format proc/thread)

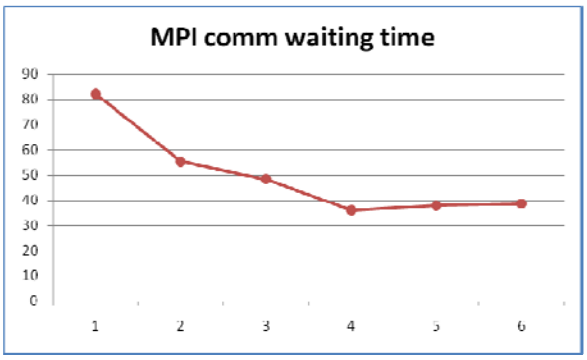


Figure 36: Average time spent in waiting for MPI communication to complete (in seconds), with regard to number of MPI processes per node on CURIE with 512 total cores and 64 cells per core

Vlasiator simulations showed improvements in some areas when hybridization was applied, but in some others, benefits were negligible, or the performance was even lower. Increasing the level of hybridization by creating more threads per single MPI process has helped to reduce parallel I/O time, which is not really a part of the simulation itself, but needs to be performed when creating restart files. External numerical libraries were not used and most of the computation was in basic floating-point operations.

The hardware platform used has Intel Nehalem ccNUMA memory inside a single node, with 4 separate CPU and RAM slots. Memory latency increases when accessing a memory location in a slot non-local to the CPU, which could happen when the number of threads per node is greater than the number of physical cores per chip. Tests with various memory allocation patterns showed that memory access time is not a major bottleneck for Vlasiator, but for future work it would be useful to implement a memory that could handle memory locality issues on such platforms.

4.7 Evaluating and implementing hybrid approach for SPECFEM3D_GLOBE

4.7.1 Project summary

Project authors: Marcin Zielinski (SARA)

Contributors: Marcin Zielinski (SARA), John Donners (SARA)

Publications:

Marcin Zielinskia, John Donners, "Evaluating and implementing hybrid approach for SPECFEM3D GLOBE" PRACE technical whitepaper

Applications area(s): Earth and Climate Science

Hardware platform(s): IBM Power6 (Huygens @ SARA, Amsterdam, The Netherlands)

Programming language(s): Fortran 95

Profiling/debugging tools: Scalasca

Libraries: none

Brief project description:

The entire project focused on an evaluation of the code for possible introduction of OpenMP and its actual implementation and extensive tests. Major time consuming parts of the code were detected and thoroughly analysed. The most time consuming part was successfully parallelized using OpenMP. Very extensive test simulations using the hybrid code allowed for many further improvements and validations of its results. Possible improvements have also been discussed with the developers to implement in the near future.

4.7.2 Hybridization work

This hybridization of the code was meant to introduce a better flexibility in handling the amount of parallel tasks and to enable better scaling of the application when moved towards petaflop supercomputers to run global scale simulations of very high accuracy.

The entire SPECFEM3D_GLOBE package has been ported to the IBM Power6 system at SARA in Amsterdam. Porting of the package to the local system, test runs, profiling and analyses took approximately 1.5 months. The profiling procedure of the specfem3D (solver from hereafter) has shown that up to approximately 90% of the solver execution time is spent in one particular subroutine called *compute_forces_crust_mantle_Dev*, which consists of a one, big Fortran DO-loop. A more accurate manual profiling exhibited one even smaller, inner DO-loop, which is executed for approximately 30-35% of the entire solver execution time.

The first implementation of OpenMP within the solver has started for this smallest, constituent DO-loop. First tests of the initial hybrid code (called v1) have revealed that the porting was successful, giving proper results and scaling, and the hybridization of the entire *compute_forces_crust_mantle_Dev* subroutine DO-loop was the next step to create a final, hybrid solver code (called v2 at this stage). Development and tests of both v1 and v2 hybrid codes was spanned across two months.

During the development of the v2 hybrid code, the main challenges were to keep the original MPI communication within the OpenMP block correct and to probe and properly allocate hundreds of variables, inside the parallel region, as SHARED or PRIVATE. Explicit barriers had to be placed before and after each MPI communication call to reassure a full synchronization between the threads. Moreover, detailed analyses showed that a few shared

arrays were suspected to be written simultaneously by two or more threads at the same time. Therefore, they have been put inside CRITICAL regions to prevent from possible racing conditions. After the v2 code proved to be giving proper results from the test simulations, full performance tests have been done. They've been divided into two groups, performed for the same model, but with different parameters, number of MPI tasks and varying number of OpenMP threads. Test #1 were limited to 2304 MPI tasks and done on high-resolution, regional scale white test #2 were limited to 2400 MPI tasks and done on a global scale.

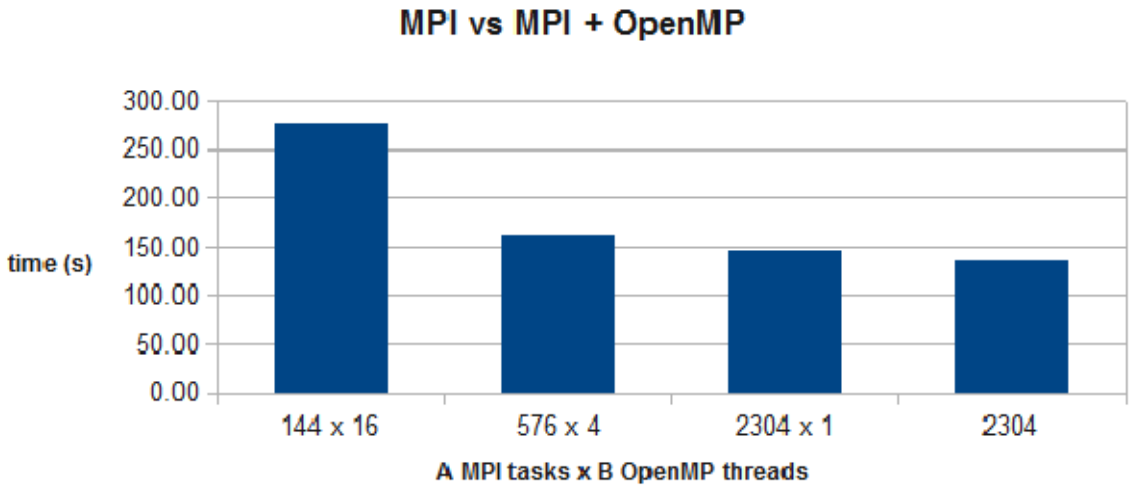


Figure 37: Performance scaling dependency for various numbers of MPI tasks and OpenMP threads per each MPI tasks, on the Power6 testing machine for Tests #1

The performance tests of the v2 code have shown that it is still possible to gain an extra speed-up. A more detailed test of the v2 code revealed that only one of the shared variables introduced a racing conditions and only that one has been kept inside a CRITICAL region. To improve the performance of the v2 code even further, an OpenMP scheduling mechanism of the main DO-loop has also been implemented. A thorough tests for different numbers of OpenMP threads and for different scheduling mechanisms with different chunk sizes have shown, that the best fitting scheduling policy for the solver is 'dynamic' with the chunk size equal to the number of OpenMP threads or equal to twice the amount of threads. The scheduler is chosen specifically per each simulation, from within the run-time environment. An improved and thoroughly tested version of the v2 code, renamed to v5, was ready for another full performance tests. They've been done precisely the same way as the #1 and #2 for v2 code. Improvement analyses of the v2 code, extensive tests and validations and full performance tests of the v5 code took another 2.5 months.

Figure 37 presents the total times, based on the v5 code, spent in the solver with various number of threads (16, 4, 1) and MPI tasks (144, 576, 2304) on IBM Power6 machine for the same model with the same parameters. Shortly, a simulation with 576 MPI tasks and 4 OpenMP threads shows that the number of nodes can be decreased from 2304 with a very little increase in execution time. Moreover, based on the timing of the simulation with 2304 MPI tasks and 1 OpenMP thread, it can be assumed that there is even more room for further, possible improvements.

The SPECFEM3D_GLOBE application is not dependent on any external library except the standard math libraries of compilers and MPI. The v5 hybrid code has been merged with the official SVN repository of SPECFEM3D_GLOBE and possible improvements have been already discussed with the main developers, to implement in the near future.

4.8 Evaluating the hybrid approach for GPAW

4.8.1 Project summary

Project authors: Jussi Enkovaara (CSC), Martti Louhivouri (CSC), Mikael Rännar (Umeå Univ.)

Contributors: Petar Jovanovic (IPB), Vladimir Slavnic (IPB)

Publications:

PRACE technical whitepaper: "Optimizing GPAW" by Jussi Enkovaara, Martti Louhivouri, Vladimir Slavnic, Mikael Rännar, Petar Jovanovic

Application area(s): Physics, Materials Science, Nanoscience (density-functional theory)

Hardware platform(s): CURIE, PARADOX at IPB

Programming language(s): Python, C

Profiling/debugging tools: TAU

Libraries: Python, LAPACK/ScaLAPACK, BLAS, MPI+OpenMP

Brief project description:

The goal of this project was to measure performance and scalability of hybrid version of GPAW and find possible optimizations. Even though the current pure MPI version of GPAW is achieving good performance, it is expected that increasing number of cores per CPU will make a hybrid MPI/shared memory implementation compulsory in the near future. A preliminary hybridization of the most important C functions is tested and performance is compared to the non-threaded MPI version.

4.8.2 Hybridization work

Performance of the preliminary hybrid implementation of GPAW has been tested and scaling has been compared to the current pure MPI version. Scaling tests have been performed on a dataset for a cluster of 702 Si atoms, by measuring the execution time as a function of number of MPI processes and threads (in combinations: 512x1, 256x2, 128x4, 64x8, 32x16, 1024x1, 512x2, 256x4, 128x8, and 64x16). Execution time for runs on 512 cores is shown in Figure 38. Results demonstrate that the introduction of threads decreases the performance up to tenfold, as in the case of 512x1 vs. 32x16, where execution time increased from 15 minutes to 2 hours 32 minutes. Profiling of computation and communication has been done for runs with smaller number of cores due to smaller scale of examples and glitches with MPI libraries on CURIE, which has not shown any obvious bottlenecks in the code. However, Figure 39 suggests that there could be problems regarding the cache usage. It is suspected that threads, while operating on shared array data, which gets cached into the same cache line, are causing frequent cache invalidations for each other.

Profiling of computation and communication has been done with a smaller example of C60 atom for brevity. Graphic presentation of program trace is given on Figure 40, which shows two run configurations: part a) of the figure shows trace for 2 processes with 8 threads each, and part b) shows 16 single threaded processes (i.e. pure MPI, without threads). Figure 40, a) shows that there is a lot more communication and waiting (shown as red rectangles) than in part b) where there are only two clusters of dense message exchange and almost no waiting. This suggests that multiple threads when trying to communicate using MPI calls, because synchronization issues within MPI library. Such claim is also consistent with results in Figure 38, which show that different MPI libraries can have significant impact on performance.

The main challenge has been dependency on MPI library's support for MPI_THREAD_MULTIPLE, because libraries provided on CURIE had no support for it. This dependency arises from the need to use MPI communication from within OpenMP threads. However, many current libraries do not offer this level of thread support by their default configurations, and many warn about low maturity of that feature and possible performance degradations.

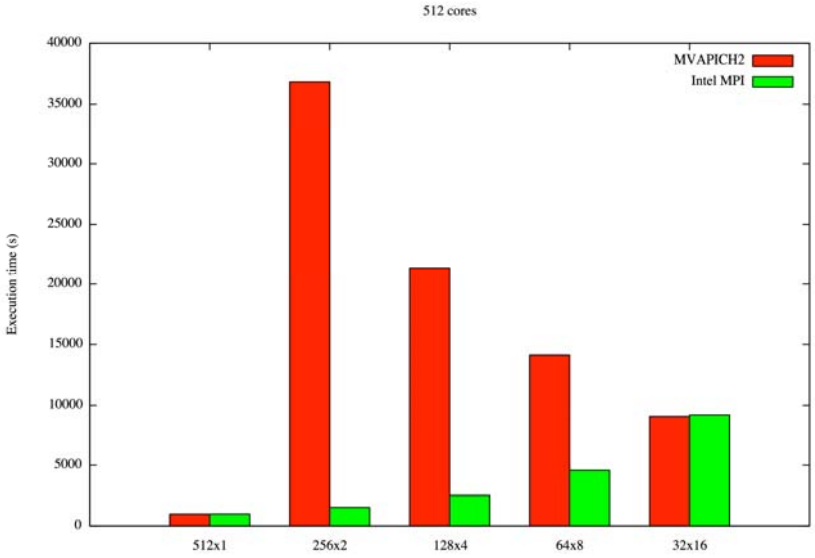


Figure 38: Execution times for Si cluster with 702 atoms on 512 cores. We observe significant increase in the execution time of hybrid version as number of threads per process is increased.

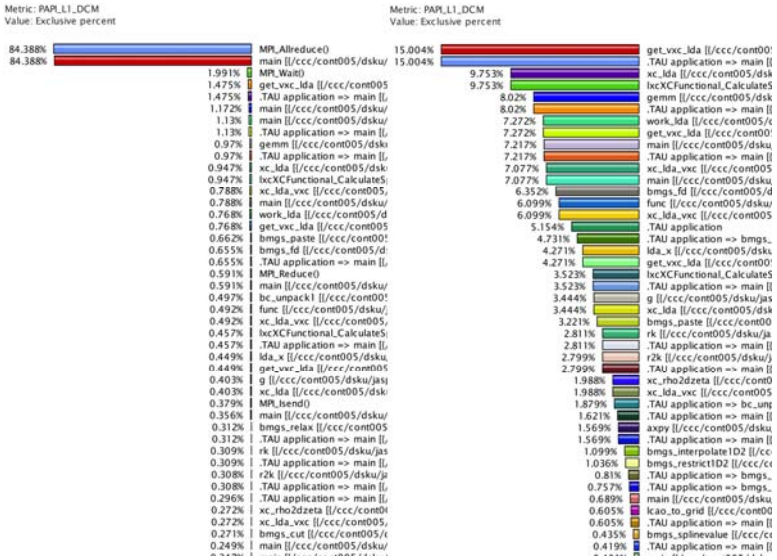


Figure 39: L1 cache misses in MPI-only version (left) and hybrid version (right)

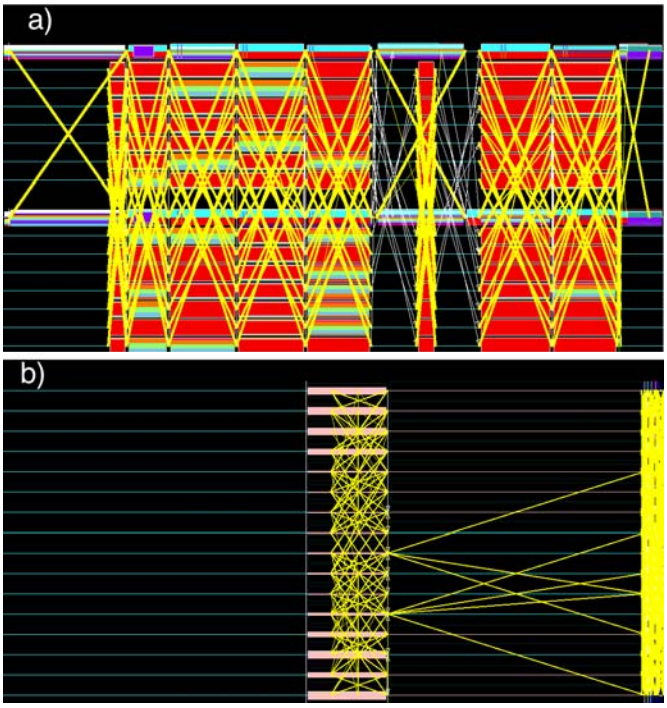


Figure 40: MPI communication trace: a) 2 processes with 8 threads; b) 16 processes without threads.

It was observed that MPI libraries compiled with MPI_THREAD_MULTIPLE are having problems with the SLURM resource manager, such as not actually spawning any threads, or failing to launch the library's process manager. Some of the tested configurations of OpenMPI were even unable to establish communication between processes. The configuration that seems to work best is MVAPICH2 configured with "--enable-threads=multiple --with-pmi=slurm --with-pm=no" parameters.

The current hybrid version of GPAW does not provide an improvement over the pure MPI version. Tests show that execution time increases when threads are activated. MPI_THREAD_MULTIPLE supported by the MPI library can pose difficulties during installation, and might be one of the causes for the observed lower performance. Along this line, it has been also noticed that the threaded version can run faster as long as it is running inside a single MPI process. Better results obtained with the Intel's MPI library (Figure 38) could be explained by better synchronization and locking strategies inside the library used to support the required level of thread safety. This suggests that performance gains could be achieved by avoiding reliance on MPI_THREAD_MULTIPLE.

4.9 Improving MPI communication latency on Euroben kernels

4.9.1 Project summary

Project authors: Chandan Basu (SNIC-LiU)

Contributors: Chandan Basu (SNIC-LiU)

Publications:

Chandan Basu, "Improving MPI communication latency on Euroben kernels", PRACE technical whitepaper

Applications area(s): Parallel programs, collective operations

Hardware platform(s): CURIE

Programming language(s): C, MPI

Profiling/debugging tools: none

Libraries: none

Brief project description:

On modern clusters with many-core/multi-core nodes there are large differences in bandwidth and latency between cores in the same node and cores across different nodes. This leads to a non-uniform network topology for MPI processes with respect to bandwidth and latency. This might impact the data transfer rate and consequently the scalability of codes. On such systems, topology-unaware MPI programs can create network congestion. Thus it is important to write topology-aware algorithms, which exploit intra-node bandwidth/latency more and less rely on inter-node bandwidth/latency. Although it is generally difficult to introduce network awareness in application programs, for MPI collective operations it is possible to rewrite the collectives in a topology-aware way. In this project we have demonstrated the impact of topology-aware MPI_Alltoall routine on the performance of a kernel called mod2f from Euroben benchmark suite. Euroben benchmark suite has many synthetic benchmark programs. The C-MPI version of mod2f program is suitable for testing of our approach, as it is communication-intensive. We have tested the scalability and performance of this modified version of mod2f with respect to the performance of the original mod2f for different data sizes with encouraging results.

4.9.2 Hybridization work

The mod2f routine is a 1-D Fast Fourier Transform (FFT). The MPI version of mod2f implements FFT on distributed arrays. The main communication overhead comes from a routine called *gtrans*. This routine does a global transposition of arrays. To achieve the global transpose *gtrans* calls *MPI_Alltoallv* routine. As it is well known *MPI_Alltoall*/*MPI_Alltoallv* are very communication intensive MPI calls and thus can prevent scaling to very large core counts. While running, the code prints out communication overhead information, which shows that most of the time is spent in communication routines.

We developed a topology aware version of *MPI_Alltoallv* called *MPI_Alltoallv_tuned*. In mod2f we replace the *MPI_Alltoallv* calls with *MPI_Alltoallv_tuned* calls. However the interface of *MPI_Alltoallv_tuned* is not exactly the same as the one for the standard *MPI_Alltoallv*. Some data structures passed to *MPI_Alltoallv_tuned* are different from *MPI_Alltoallv*. Those data structures are additionally defined and initialized at appropriate places in the code.

We needed to do some structural changes in the code for improving it's efficiency. In the original version of mod2f the send/receive/displacement counts for *MPI_Alltoallv* are computed within gtrans. However these counts do not change for a particular FFT operation. It is sufficient to compute those counts only once outside the iterative loop; the code was modified accordingly. This modification is also very suitable for our approach. Because in our approach send/receive/displacement counts are passed as structures and their computation is more complex than send/receive/displacement counts for *MPI_Alltoallv*.

In the modified version of the code the original version or the tuned versions are activated by pre-processor flags. As we are interested in seeing the scaling effects on large numbers of cores/nodes we choose relatively large problem sizes. We performed our tests on CURIE, results are shown in the Figure 41. The results revealed that execution times of mod2f_tuned are much lower than for the standard mod2f version.

The current *MPI_Alltoallv_tuned* works for MPI_COMM_WORLD, the total number of MPI processors should be divisible by the number of cores/node. There are a few ad-hock solutions

used in the current version of M*PI_Alltoallv_tuned*, e.g., statically allocated work buffers. However, these are just limitations of the current implementation and could be removed.

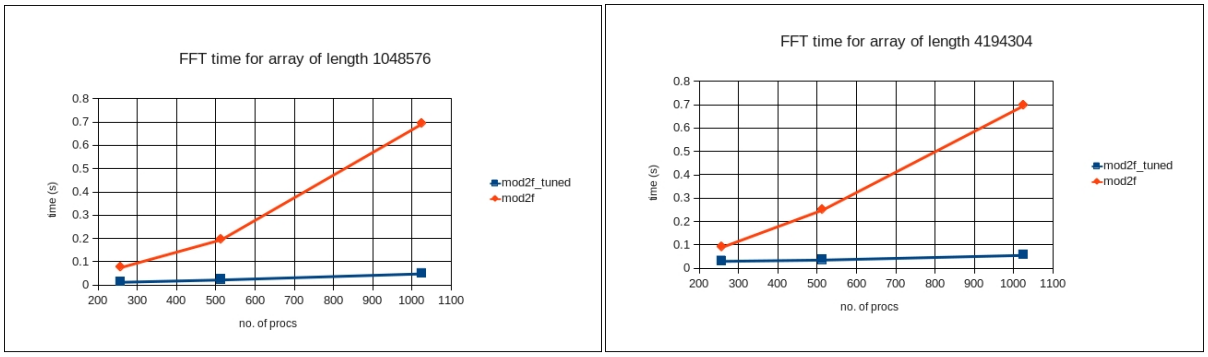


Figure 41: Scaling analysis of mod2f and mod2f_tuned for array lengths of 1048576 and 4194304.

4.10 Conclusions

The summaries presented in the previous sections give a detailed overview of the main results obtained within sub-task 7.5.C, which focused on hybrid programming models for multi-core/many-core systems. In particular, the first 4 projects report the support activity on petascaling enabling for applications selected within Tasks 7.1 and 7.2. As expected, the hybrid programming model boosted the performance of some applications, while severe issues were identified in some cases:

- In collaboration with ScalaLife, the first hybrid version of the GROMACS package was released and benchmarked. It provides a significant increase in the performance for large systems.
- DALTON is also about to release a hybrid version of its DFT component, which is substantially improved through the 7.5.C contribution.
- A detailed scalability and profiling study of EC-Earth3 revealed bottlenecks of the hybrid version of the code.
- As a result of the work on the ELMER project, the CSX sparse matrix format is integrated into the code, and the performance is improved by around 35% compared to the previously used CSR.

The work within the sub-task also focused on evaluating the use of hybrid OpenMP/MPI programming for several further codes. The hybrid programming model proved to be quite challenging in some cases:

- HYDRO code was successfully ported to the hybrid MPI/OpenMP paradigm, and shows significant improvement in the scalability compared to the pure MPI implementation. In particular, hybrid code scales almost perfectly up to 16.384 cores, and continues to scale sub-optimally to 32.768 cores.
- A scalability and profiling study of the Vlasiator code has identified the optimal number of threads per MPI process, as well as the source of the main performance bottleneck. It also analyzed possible further improvements to the code.
- The work on SPECFEM3D_GLOBE resulted in porting the code to the IBM Power 6 system, and introduced a hybrid version of the code. Weak scalability tests were run, bottlenecks identified, and future improvements of the code suggested.
- The evaluation of the hybrid implementation of GPAW revealed that it gives degraded performance compared to the pure MPI implementation. The main reason for this was identified by profiling, and was found to be related to severe L1 cache misses, which

are not present in pure MPI runs, as well as increased MPI communication wait times. MPI_THREAD_MULTIPLE is also identified as a possible culprit.

The work on the final project, aiming to evaluate the introduction of network topology-aware MPI communication routines, has obtained excellent results for the mod2f kernel from the Euroben benchmarking suite. It demonstrates that, in order to increase performance of hybrid codes, MPI communication routines should be adjusted to take into account the topology of the interconnecting network.

In conclusion, the obtained results show that the effective use of the hybrid programming model is quite challenging and could have severe limitations due to issues related to MPI, such as the use of MPI_THREAD_MULTIPLE, as well as due to inefficient, network-unaware communication. The inherent complexity of achieving optimal performance of pure MPI applications is further complicated by the introduction of an OpenMP thread-level. Frequent changes in hardware implementation of intra-CPU and intra-node communication of cores, sharing of caches and accessing the memory prohibit efficient use of higher abstraction levels in programming. Therefore, achieving optimal performance usually requires adjusting the programming model to the hardware implementation, and leads to the significant effort necessary for porting the code from one architecture to another.

5 Accelerators

In recent years two HPC programming frameworks, CUDA and OpenCL, have been widely adopted by the computational science community for writing programs that execute on heterogeneous platforms (CPU multi-core processors combined with accelerator technology). OpenCL (Open Computing Language) is an open standard for parallel programming of heterogeneous systems that provides multi-platform support for products from most hardware vendors such as AMD, NVIDIA, IBM, Intel, Apple and others. Based on the C language, OpenCL requires the application programmer to write kernels to be executed on many-core platforms. Apart from the important benefit of having no vendor dependency, applications written in OpenCL are capable of running efficiently on several operating systems supporting massively parallel hardware including multi-core CPUs, multi-vendor GPUs, and potentially future hardware as well.

CUDA is a quickly maturing software development framework provided free of charge by NVIDIA to develop applications targeted for NIVDIA GPUs. It is currently the most mature and widely used GPGPU development platform: according to recent NVIDIA reports, over 300 million CUDA Capable GPUs have been sold, with over one million downloads of the CUDA Toolkit and over 150 thousand registered CUDA developers. NVIDIA recently highlighted the growth of the CUDA eco-system [9] worldwide with the impressive figure that today approximately 500 universities or research institutions teach CUDA within their program of education. Two of the five Gordon Bell finalists at SC2011 (one awarded with the Gordon Bell "Special Achievement in Scalability and Time-to Solution" prize for his work on the Tsubame 2.0 supercomputer) are part of the recent proliferation of reports in the scientific and technical literature. CUDA represents an extension of the C programming language through the addition of a small number of keywords to simplify the development of efficient applications for CUDA-enabled GPGPUs. Although OpenCL has fairly broad support from several prominent manufacturers, (in response to a common concern that CUDA is developed by a single vendor tailored for the vendor's products), CUDA is today's most expressive and widely used GPGPU development framework.

Despite the growing community and the continuous improvement of the NVIDIA software environment, scientific communities are generally reluctant to tackle the porting of scientific codes to GPU computing platforms. With the upcoming release of Intel's Many Integrated Cores (MIC) architecture, Intel aims to enter into the accelerator computing market with a solution that is intended to facilitate the exploitation of accelerator co-processors. Intel promises that programming MIC will only require the compiling of source code with specific options. However, at the time of writing, Intel has not yet released key pieces of information regarding MIC, such as the number of cores, pricing, and power consumption of the next product to market. Although some PRACE partners have access to the MIC architecture (protected by non-disclosure agreements) there is no general access available to PRACE partners with the consequence that this product has not yet been evaluated in PRACE-1IP.

AMD technology has also not been considered here. Over the last few years, AMD accelerators have not shown themselves to be popular in the European HPC market. Aside from a few small installations, focused mainly on local testing and development, there are currently no large-scale architectures accelerated by AMD technology on the European scene.

In fact, all major GPU-based HPC installations are enabled by NVIDIA technology. Since 2010, when NVIDIA began to ship the Tesla 20 series (~ 500 GFlop/s peak of performance in double precision) the top 5 positions of the TOP500 list have increasingly featured systems equipped with NVIDIA GPU cards. The first appearance occurred in June 2010 with the 'Nebulae' supercomputer installed at the National Supercomputing Centre in Shenzhen and since then at least three positions within the top 5 of the TOP500 list have been held by

supercomputers with CPU+NVIDIA GPU-based architectures (lists published in November 2010 and June/November 2011).

Both NVIDIA's and Intel's roadmaps for the next few years strengthen the expectation of the huge increase in the compute density per node in the next generation of supercomputers. In the short term, some of the leading HPC centres in the US will base their HPC technology on the accelerator-based heterogeneous architecture. In September 2011, the Texas Advanced Computer Center (TACC) announced Stampede [10], a new 10-petaflop leadership class of supercomputer based on Intel MIC architecture, expected to be up and running in January 2013, representing a substantial commitment by Intel to deliver massively parallel many-core hardware systems. In tandem with this development, the collaboration between Cray and NVIDIA brought to the light the design of XK6 system [11], based on Cray XE6 blade equipped with AMD Opteron 6200 series processor and NVIDIA GPUs. This architecture is planned to be used in two of the most powerful multi-petaflop supercomputers in the world: Titan at ORNL [12] announced in October 2011 and the new Blue Waters project at NSCA [13], re-announced in November 2011 after IBM has withdrew from the original project in August 2011. These installations are expected to be up and running in 2013 and late 2012, respectively.

Although U.S. and Eastern Asia (China and Japan) have published long-term commitments to build world-class supercomputing platforms based on many-core technology, Europe is currently not committed to an accelerator-based Tier-0 system. However, a number of European HPC centers have recently commissioned medium size systems based on NVIDIA's GPU technology. Three of these systems are today listed within the top 15 of the Green500 list [14], which provides rankings of the most energy-efficient supercomputers in the world (see table below). Moreover, a number of other National HPC centres such as FZJ, HLRS, PSNC, IPB, ICHEC and EPCC are deploying (at different levels) technologies and resources to enable GPU computing in Europe. The Tier-1 GPU systems installed at CINECA, HLRS, PSNC and IPB are accessible for European users through the PRACE-2IP DECI call.

Rank	MFlops/W	Site	Computer	Total Power
7	1266.26	BSC	Bullx B505, Xeon E5649 6C 2.53GHz, Infiniband QDR, NVIDIA 2090	81.50 kW
8	1010.11	GENCI	CURIE Hybrid Nodes - Bullx B505, Nvidia M2090, Xeon E5640 2.67 GHz, Infiniband QDR	108.80 kW
13	891.88	CINECA	iDataPlex DX360M3, Xeon E5645 6C 2.40 GHz, Infiniband QDR, NVIDIA 2070	160.00 kW

Table 13: The three European GPU systems ranked in the top15 of the Green500

Several European experts in GPU computing expressed interest in collaborating for this subtask. Mainly through projects selected in Task 7.2, a modest but reasonable amount of resources (~30PM) was made available to analyse the possible impact of GPU computing. Quantum-ESPRESSO, DL_POLY and OpenFOAM are the main scientific applications analysed in this subtask across different projects, as described in the following sections. An additional work of performance analysis for 3DFFT libraries was considered, as this problem is applicable to many codes. The extension of a CUDA library for QCD simulation was developed in the context the preparatory access project "Lattice QCD at the Peta-flops scale" selected in Task 7.1.

5.1 Lattice QCD on Accelerators

Contributors: Alexei Strelchenko (CASTORC), Giannis Koutsou (CASTORC)

Publications:

PRACE technical whitepaper: "Extending the QUDA library for Domain Wall and Twisted Mass", Alexei Strelchenko (CASTORC)

5.1.1 Project Goal

The goal of this project was to extend QUDA [15], an open source library for performing calculations in lattice QCD on Graphics Processing Units (GPUs) using NVIDIA's CUDA platform. QUDA includes optimized implementations of a number of different discretizations of the continuum QCD fermion operator, such as the Wilson and Staggered fermion actions. It also implements a range of iterative solvers for these fermion actions, i.e. CGNE (for normalized equations), BiCGStab and recently the domain decomposition solver. For the first part of the project we implemented the non-degenerate twisted mass fermion operator, in the context of the preparatory access project "Lattice QCD at the Peta-flops scale" which was awarded computer time on CURIE in the first round of preparatory access calls. The second part of the project consisted of MPI-parallelization of the Domain Wall fermion operator, an operator which was already available in QUDA albeit only for a single GPU.

5.1.2 Result description

The main kernel operation of QUDA is sparse matrix-vector multiplication (which represents a re-casting of various fermion operators). These are available for double, single and half precision in order to exploit a novel mixed precision technique [16] which allows one to obtain the solution in full double precision accuracy while using only single or half precision arithmetic for the bulk of the computation. Additionally, even-odd (or red-black) preconditioning is used according to the problem at hand. Therefore, this work, including the non-degenerate twisted-mass fermion operator, consisted of implementing the operator for all three arithmetic precisions and for even-odd ordering.

For the case of the Domain Wall operator, parallelizing across multiple GPUs was carried out using MPI, where, upon partitioning the lattice, each GPU is assigned a 4-dimensional sub-volume. Updating sites of the vector on or near the boundary requires data from neighboring GPUs, which requires the allocation of dedicated buffers (i.e. halos) in GPU device memory. This work therefore consisted of modifying the computational kernels, which implement the Domain Wall operator, so as to be aware of the partitioning and thus read data from the appropriate locations.

Below, we present performance results for the two operators we have implemented. We timed both the matrix-vector multiplication as well as the iteration time when using the operator in an iterative solver.

a) Twisted-mass non-degenerate operator

The results are given for a 32×64 lattice with parameters κ=0.163272, μ-bar=0.19 and ε= 0.15. The runs were performed on NVIDIA M2090 GPUs. For the matrix-vector multiplication, we achieve 33 GFlop/s in double precision, 136 GFlops/sec in single precision and 162 GFlop/s in half precision.

To compare with typical performance on the power-efficient BlueGene/P cluster (IBM PowerPC450), we time an inversion on both a number of BlueGene nodes and a single GPU and quote the time per inversion. On the GPU, for a tolerance of 10^{-10}, we require 174.3

seconds for 1116 iterations in mixed precision, compared to 184 seconds for 1383 iterations for the same tolerance on 256 cores (64 nodes, theoretical peak performance 870.4 Gflops in double precision). This means that the GPU accelerated inverter on a single device with power consumption ≤ 225W provides the same performance level as 64 Blue Gene nodes which require 0.34×13.6×64 = 2559 W in total.

b) Parallelized Domain Wall operator

Results for our MPI Domain Wall operator are given for a $28^3 \times 64 \times 4$ lattice, with mass parameters: M_5=-1.0 and M_0 = 0.0138. The performance, compared with that obtained on a CPU system, is given in Figure 42.

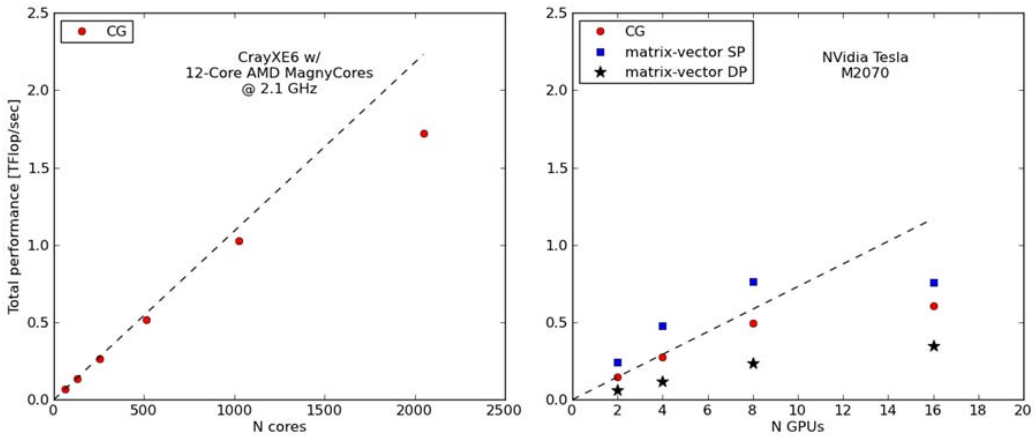


Figure 42: Performance comparison on CPU (left) and GPUs (right) for the domain wall solver (4.63x13.24, dashed line indicates ideal scalability).

Although we see a degradation of the scaling in the CG, which is the effect of the degradation of the performance of the double-precision version at the 16 GPU point, we can say that on average the GPU implementation offers a speed-up of around 6 times when compared to a single CPU socket. Namely, QUDA's performance is around 37.6 GFlop/s per M2070 GPU, compared to the average of 6.2 GFlop/s we obtain for a single CrayXE6 "MagnyCores" 6-core socket at 2.1 GHz.

5.1.3 Consideration and conclusion

Overall the project objectives were met; the QUDA library was extended to implement two extra fermion operators, thus extending the potential user base of this software package. Implementation of the non-degenerate twisted-mass operator will allow the utilization of GPUs for a wider set of problems than was already possible with QUDA. These problem types are relevant to the European Twisted Mass Collaboration, one of the larger lattice QCD collaborations in Europe. For the case of the Domain Wall fermion operator, parallelizing the matrix-vector multiplication effectively enables the use of GPUs for this branch of lattice QCD, since this operator is particularly demanding on memory, due to the fact that it is defined on a five-dimensional grid. For realistic lattice sizes the problem does not fit on a single GPU, meaning that if one is to use GPUs, a multi-GPU implementation is necessary. The code developed within this project is temporarily available in a separate branch of the official QUDA repository on git-hub [17] and it will be merged to the master branch before the next release.

5.2 Accelerating the scf calculation of the Quantum-ESPRESSO suite

Contributors: Filippo Spiga (ICHEC), Ivan Girotto (ICHEC), Carlo Cavazzoni (CINECA)

Publications:

PRACE technical whitepaper: F. Spiga and I. Girotto, phiGEMM: a CPU-GPU library for porting Quantum ESPRESSO on hybrid systems, Proceeding of 20th Euromicro International Conference on Parallel, Distributed and Network-Based Computing -- Special Session on GPU Computing and Hybrid Computing, IEEE Computer Society, ISBN 978-0-7695-4633-9, pp. 368-375 (2012)

Enabling of Quantum-ESPRESSO to petascale scientific challenges (PRACE white-paper)

5.2.1 Introduction

QUANTUM-ESPRESSO (QE) [18], [19] is an integrated suite of computer codes for electronic-structure calculations and materials modelling based on density-functional theory, plane waves basis sets and pseudo-potentials to represent electron-ion interactions. It is freely available and distributed under the terms of the GNU General Public License (GPL). Within this project GPU-Computing experts from ICHEC and CINECA undertook the task of extending the PWscf [20] code included into the QE suite to fully exploit NVIDIA GPU capabilities.

5.2.2 Project Description

The PWscf code is designed for highly scalable execution with the focus of maintaining a considerable level of portability. As reported in PRACE-2IP-D8.1.2, this is mainly achieved with a modularized structure that implements several levels of parallelization and the systematic use of standardized scientific and mathematical libraries such as BLAS, LAPACK and FFTW. The activities performed in this sub-task aim to enable the PWscf code to include NVIDIA CUDA libraries to exploit the use of accelerators. This model can be generally extended to other scientific codes as well as different architectures (i.e., AMD, Intel) to leverage the full capability of heterogeneous computing nodes. We implemented on CUDA only a few additional computational kernels to improve the overall performance. The GPU code was developed gradually, starting from the serial version of the code. The most computationally expensive sections of the Self-Consistency Field (SCF) [see Figure 13, PRACE-2IP-D8.1.3] were gradually enabled to run on NVIDIA GPUs. The performance of each section was assessed step by step. The accelerated version mainly involves:

- GEMM operations were replaced transparently with phiGEMM operations
- 3D-FFT was accelerated using CUFFT wherever possible
- Computationally expensive routines were replaced by CUDA kernels
- LAPACK functions were replaced with equivalent MAGMA functions

GEMM operations (mainly DGEMM or ZGEMM) may represent up to 35-45% of all the wall-time of a typical representative benchmark. We firstly undertake the task to enable the execution of matrix-matrix multiplication on GPU systems developing the phiGEMM [21] library that extends the basic mapping presented in [22]. Figure 43 demonstrates how it is possible to obtain more than 700 GFlop/s on a hybrid 12-Intel-Cores system equipped with 2 C2050 NVIDIA GPUs using phiGEMM. The phiGEMM library is easily linkable to any scientific application. Moreover it can be leveraged into a high-exposure, commonly accepted HPC benchmark for conventional and co-processor accelerated computers.

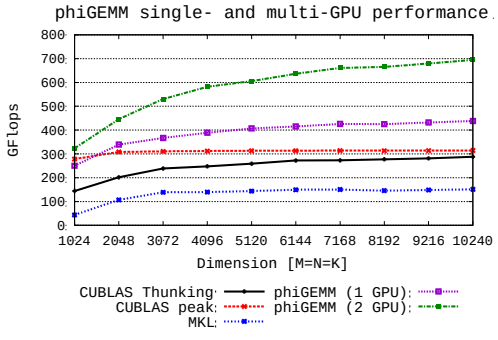


Figure 43: PHIGEMM performance obtained using one and two GPUs

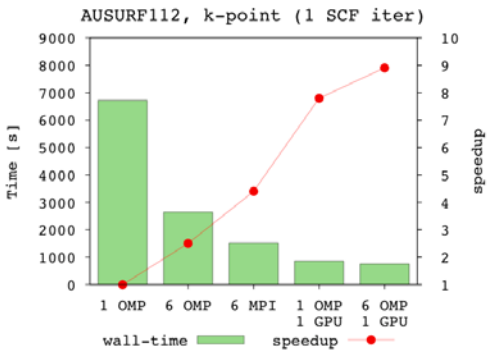


Figure 44: Best results obtained running PWSCF on one six-core Intel Xeon X5650 and one Tesla C2050

5.2.3 Conclusion

The serial version shows a speed-up that varies between 6 and 9, depending on the physical system. In Figure 44, we compare the best performances obtained using both only one CPU and one CPU plus one GPU while running on a given single node system. For this test we used 'AUSURF112', a medium size input benchmark for PWscf, (a gold surface of 112 atoms).

The distributed version still represents an on-going challenge. In this case, the phiGEMM library and the majority of the implemented CUDA kernels are re-used since they act on local data, but the main problems performed on distributed data need further investigation. A new parallel 3D-FFT approach is under development. We are looking for the best implementation to minimize the inter-process communication and perform CUFFTs (eventually batched) on re-aggregated data. Early experiments (with parallel 3D-FFT executed on CPU) were performed on up to 32 nodes of CINECA PLX (12 Intel cores and two Tesla M2070 x node) running one SCF step over gamma-point on a medium-size system of 216 atoms (Ge, Mn, Te, Sb). Results show a speed-up of between 2.5 and 3.5 if comparing CPU versus GPU versions of the code running on the same number of CPU cores (Figure 45). Obviously, in the case of the GPU version GPUs are used on top of the CPU system. Figure 46 compares the time of simulation on a given number of nodes either using GPUs+CPUs or only CPUs. The use of 144 cores and 24 GPUs together outperforms the 144-CPU-cores-only computation by a factor of 3.3 (8510 versus 2510, elapsed time seconds). The same time-to-solution is achievable using 384 cores or more on the same platform.

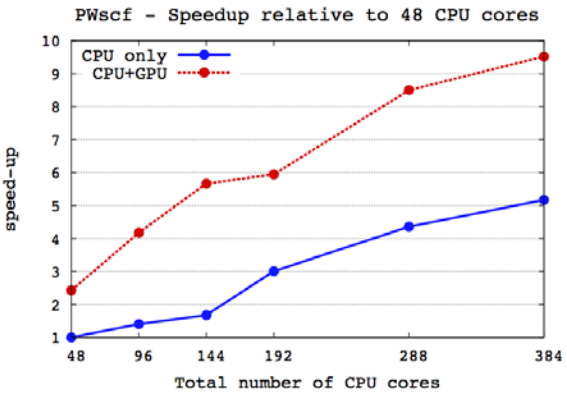


Figure 45: Speed-up obtained by the GPU version over 48 CPU cores

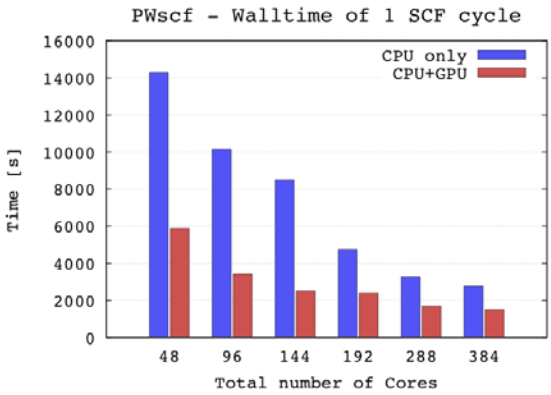


Figure 46: Wall-time execution

5.3 OpenFoam

Contributors: Christos Theodosiou (Aristotle Univ.), Paschalis Korosoglou (Aristotle Univ.), Giovanni Erbacci (CINECA), Ivan Spisso (CINECA)

5.3.1 Project goal

The goal of this project is to contribute to the OpenFOAM project performed in Task 7.2. A full report will be delivered for D7.2.2 (due at M24) investigating possible benefits of using state-of-the-art GPU linear solver libraries. Two libraries were selected as a result of a brief investigation carried out to identify available GPU solvers:

- An OpenFOAM plugin based on the speedIT library [23] developed by Vratis Ltd. and delivered under commercial license. The library includes Conjugate Gradient and Bi-Conjugate Gradient linear solvers with Diagonal Preconditioning.
- The ofgpu library based on CUSP [24] and developed by Symscape. The library is freely available under GPL license but it does not include either hybrid MPI+CUDA or multi-GPU enabled versions. This implementation includes Conjugate Gradient and Bi-Conjugate Gradient linear solvers with several Pre-conditioners, e.g., Diagonal, Algebraic Multi-Grid, etc.

The on-going analysis is being carried out on the CINECA's PLX GPU Cluster. Only preliminary results are shown in this section. For more information please refer to the final Task 7.2 white paper.

5.3.2 Project description and result analysis

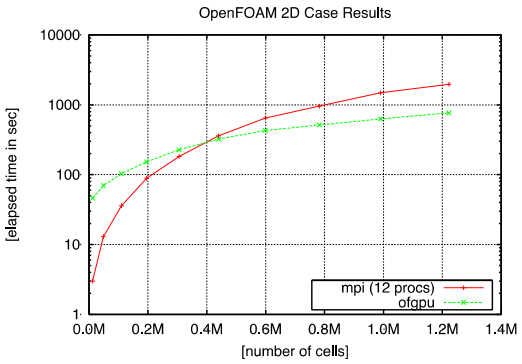


Figure 47: Timing results for different problem sizes in 2D case

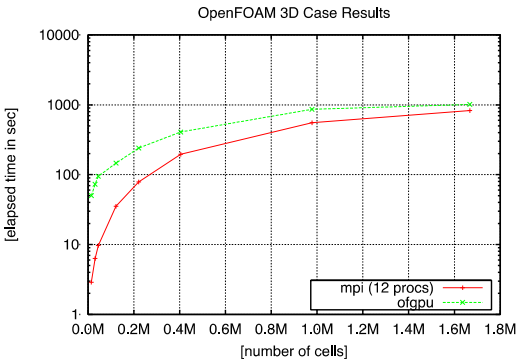


Figure 48: Timing results for different problem sizes in 3D case

As documented in [24] the ofgpu library was built on top of OpenFOAM (2.1.x) including additional header files provided as a patch by Symscape [25] and other free software packages provided by NVIDIA such as CUDA, CUSP and the Thrust library. Note that single precision is required for building the ofgpu library for OpenFOAM. To analyze the benefit of the GPU usage, a CPU version of OpenFOAM (2.1.x) was also built. Both versions were compiled with GNU C Compiler, version 4.5.1.

For both 2D and 3D cavities the same basic iterative solvers were used: Preconditioned Conjugate Gradient (PCG) for the pressure and Preconditioned Bi-Conjugate Gradient (PBiCG) for the velocity field. PCG is used for all types of symmetric positive definite systems while PBiCG is the modification of PCG that is derived for asymmetric systems. The preconditioners used in the CPU case for the pressure and velocity fields are the Diagonal-based Incomplete Cholesky (DIC) and Diagonal-based Incomplete LU (DILU), respectively. These are commonly applied to symmetric positive definite systems and asymmetric systems. Different preconditioning schemes such as smooth aggregation and diagonal for the pressure and velocity fields have been used for the GPU version, as the ofgpu library does not support DIC and DILU. Smooth aggregation is based on the multigrid method.

Figure 47 shows performance results obtained running a 2D test case for a predefined number of compute steps for a varying problem size ranging from 12.225 up to approximately 1,2 million cells. All tests were calculated for the same overall time interval using the same time-step. Note that, for the parallel MPI case, the time to decompose and recompose the domain is not measured – only the solver time has been considered. As can be seen from these results, for problem sizes larger than ~0.45M cells, the GPU enabled solver outperforms the MPI version while running on a full PLX cluster node. Moreover, as the time to decompose and restructure the domain in the MPI case is not measured in these results the contributors lead to the conclusion that for problem sizes is the order of 1M cells the ofgpu is preferred as long as using single precision arithmetic is not considered as a compromise. However, for the 3D results, Figure 48 shows that the GPU enabled solver does not perform better than the MPI version of OpenFOAM even for a problem size of 1.66M cells. Nonetheless, the time differences between the MPI version and the GPU enabled solvers reduce as the problem size increases.

In the context of Task 7.5, CINECA in collaboration with Vratis Ltd. investigated the possible benefits of using the speedIT library as a GPU linear solver library for OpenFOAM. The on-

going tests concern a large test-case with 80M cells which requires 66 GB of memory to be decomposed. Such a big domain requires the use of specific fat nodes available on the PLX cluster. The actual tests show that when compared SpeedIT with diagonal+PCG (Preconditioned conjugate gradient) on CPU, the speedIT libary has a better performance. However, the comparison of SpeedIT vs. GAMG (Geometrical Algebraic Multi-Grid) was not as promising due to a much larger number of iterations per time step (SpeedIT needed hundreds while GAMG a few tens). Further tests are replacing the diagonal+PCG with AMG (Algebraic Multi-Grid)+PCG, and are still in progress. Results should be available in the near future.

5.4 DL_POLY

Contributors: Mariusz Uchronski (PSNC-WCNS), Agnieszka Kwiecien, Marcin Gebarowski (PSNC-WCNS), Peter Nash (ICHEC), Michael Lysaght (ICHEC), Ilian Todorov (STFC)

Publications:

Michael Lysaght, Mariusz Uchroński, Agnieszka Kwiecien, Marcin Gebarowski, "Benchmarking and analysis of DL_POLY 4 on GPU clusters", PRACE technical whitepaper

5.4.1 Project goal

DL_POLY is a molecular dynamics simulation application developed by the STFC, UK. A GPU version of DL_POLY is also available within the official release. The main goals of the project are listed below:

- Development of a hybrid implementation of the DL_POLY application using OpenCL along with a comparison of performance with the existing CUDA implementation.
- A performance analysis of the CUDA implementation of DL_POLY on a GPU cluster along with re-factoring of CUDA port to mirror the latest vanilla DL_POLY4 release

WCNS GPU computing experts in collaboration with ICHEC and STFC undertook this work in connection with Task 7.2 for which DL_POLY was selected as a community code.

5.4.2 Project Description and Result analysis

In the last year several significant modifications have been made to the vanilla DL_POLY4 code by the lead developers at the STFC. Where these modifications have affected the "GPU-enabled" modules, mirror-like modifications have been implemented within the CUDA+OpenMP code at ICHEC. Updated GPU-enabled modules have been committed to the source code's main 'CCPForge' trunk repository. On top of the aforementioned mirroring, an error-handling module has been developed for the CUDA+OpenMP-based code and several bugs affecting memory and structure alignment have been fixed. This has allowed for the benchmarking of DL_POLY4 over many more nodes of ICHEC's Stoney GPU cluster than was previously feasible. Results will be reported in the forthcoming DL_POLY GPU white-paper.

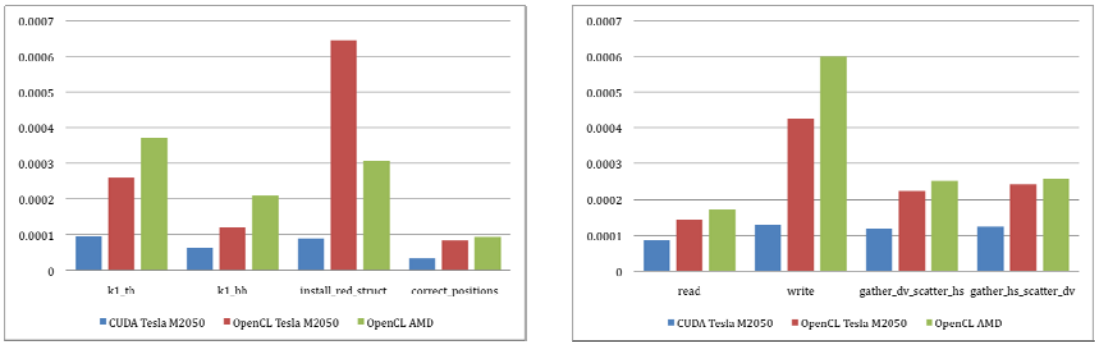


Figure 49: CUDA vs OpenCL for DL_POLY constraints shake component

For the OpenCL related project, tests have been executed on a GPU development platform at WCNS and on one of the CURIE Hybrid nodes at CEA (see the table reported on this chapter). The two GPU computing nodes are equipped with 2 x AMD Radeon HD 6950 and 2 x NVIDIA Tesla M2050, respectively. The two systems are comparable in terms of double-precision peak performance (~ 500 GFlop/s). Figure 49 shows the average execution time using 2 MPI processes and 2 OpenMP threads on the H2O test benchmark. Although developers recognize that the OpenCL version of DL_POLY requires additional tuning, in general the code so far developed seems to be poorer in performance than the CUDA version. The biggest difference between average duration time per invocation for Tesla M2050 GPU is for kernel: install_red_struct (OpenCL code is 7x slower than CUDA code) and write operations (OpenCL code is 3x slower than CUDA code). For other kernels, OpenCL calls are two times slower than particular CUDA calls. Comparing the same OpenCL version, NVIDIA GPUs generally outperform AMD GPUs. However, the OpenCL code was initially developed on NVIDIA GPUs not taking into account the specificities of AMD's GPU technology.

5.4.3 Conclusion

The initial plan of using the *swan* library for porting CUDA to OpenCL failed, as it does not support C++ like templates in kernel code that are often used in DL_POLY-CUDA. Developing OpenCL code still needs more effort than developing CUDA code but OpenCL code can be run on different platforms – AMD GPUs, AMD APUs and multicore CPUs. The results presented above show that the OpenCL code still requires improvements to achieve high performance on different architectures. As it is also reported in the literature [27],[28],[29].

So far, only the 'constraints shake' DL_POLY component has been successfully ported to OpenCL. The experience achieved during developing OpenCL code for this DL_POLY component will help during other OpenCL related work for DL_POLY (or other scientific applications). The WCNS team will continue developing the OpenCL version of DL_POLY in PRACE-2IP WP12.2 as part of the task, "Optimization of SHAKE and RATTLE algorithms". This implementation will also be evaluated on the PRACE prototype (AMD APU) installed at Poznan Supercomputing and Networking Centre (PSNC) as a part of WP9-1IP.

5.5 Analysis of 3DFFT on multi-GPU systems

Contributors: Ata Türk (Bilkent), Mustafa Korkmaz (Bilkent), Filippo Spiga (ICHEC), Rob Farber (ICHEC), Michael Lysaght (ICHEC)

Publications:

Michael Lysaght, Mariusz Uchroński, Agnieszka Kwiecien, Marcin Gebarowski, "Benchmarking and analysis of DL_POLY 4 on GPU clusters", PRACE technical whitepaper

Rob Farber, CUDA Application Design and Development, Morgan Kaufmann; 1 edition (November 14, 2011), ISBN-13: 978-0123884268

5.5.1 Introduction

This project aims to provide developers with references for performing three-dimensional (3D) Fast Fourier Transform (3D-FFT) execution on NVIDIA GPUs. Both distributed and intra-node implementations have been considered. The analysis was made in relation to community codes selected in Task 7.2: Quantum-ESPRESSO and DL_POLY.

5.5.2 Project Description

CUFFT is an FFT library freely provided by NVIDIA to provide a simple interface for computing FFTs on NVIDIA's GPU technology. NVIDIA claims that the application of this library within a single node strongly outperforms common FFT libraries such as FFTW or Intel-MKL. In the case of a single core implementation of DL_POLY we replaced the call to the custom-developed 3D FFT DAFT library with a call to an equivalent 3D CUFFT call. With this configuration we found a speedup of ~8 over DAFT running on a single CPU for a dataset of typical size used in DL_POLY.

In the case of the porting of PWscf code of Quantum-ESPRESSO more potential speed-up can be obtained. One of the main bottlenecks of this code is the series of 3D-FFTs that are performed to transform the wave functions from real to reciprocal space and vice versa. At ICHEC, Rob Farber has implemented a template code, which allows for the execution of a series of 3D FFTs on a multi-GPU platform. The code is freely available and downloadable from the web-page repository [30] of his CUDA book. The same model presented by Rob Farber has been implemented in the non-distributed version of the PWscf code, presented in previous sections.

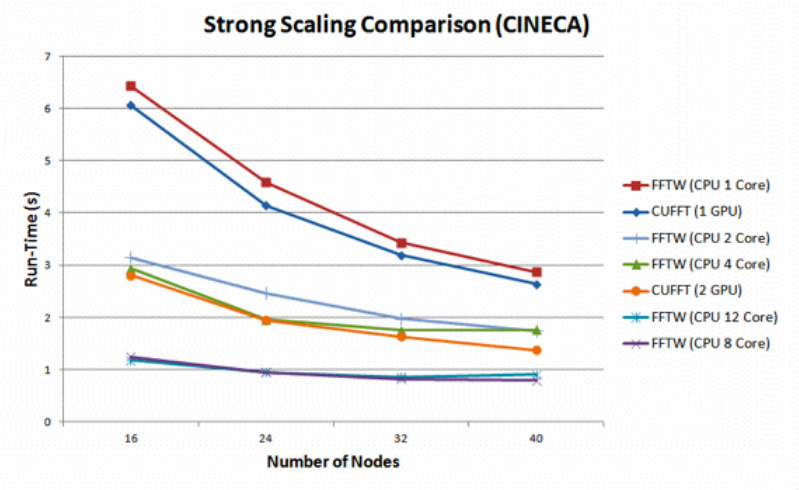


Figure 50: Performance obtained with DiGPUFFT library performing P3DFFT on 10243 grid

Currently, NVIDIA has no distributed version of its CUFFT library available so any call to the 3D CUFFT routine must occur on a single CPU node affined to a single GPU. With the possible performance advantages of using third-party FFT libraries within scientific code, in this project we investigated the performance of a distributed GPU-enabled 3D FFT library. Parallel Three-Dimensional Fast Fourier Transforms (P3DFFT) is a library that claims to be

optimized for large data sets. The library is currently being developed at the San Diego Supercomputer Centre and has recently been ported to GPUs by a group at the Georgia Institute of Technology (where the ported library is dubbed DiGPUFFT). P3DFFT uses 2-D, or pencil, decomposition as opposed to a 1-D or slab decomposition. It is written using Fortran 90 and MPI. On each node P3DFFT computes local 1D FFTs using third party FFT libraries, which by default is FFTW, though IBM's ESSL and Intel's MKL may serve as drop-in replacements. For DiGPUFFT, a custom CUFFT wrapper was developed for use within P3DFFT, making NVIDIA's CUFFT library an additional local FFT option. A performance analysis of the use of this library was performed on the CINECA PLX GPU cluster (see Figure 50).

5.5.3 Conclusions

The high performance of the CUFFT library reported by NVIDIA has been verified. The CUFFT library is easily implemented, even within complex scientific codes, such as DL_POLY and Quantum-ESPRESSO. However, this simple model can be considered only for single hybrid GPU node.

The performance benefit of using a distributed GPU-enabled FFT library over a pure MPI-based FFT library is, at the moment, less clear. While Figure 50 does indicate a performance advantage in using DiGPUFFT over P3DFFT, it should be noted that the dataset is 16 times larger than is typically used in both DL_POLY and Quantum-ESPRESSO. It should also be noted that the results above are for a single precision dataset, whereas molecular dynamic simulations generally requires double precision. The DiGPUFFT library is currently only enabled for single precision calculations and it is expected that any performance advantage currently seen over P3DFFT will diminish further when using double precision.

Considering the impressive performance obtainable by performing single node FFT other solutions should be further analysed. While such an implementation will obviously not scale, the significant speedup seen here may point to the potential benefit of using a 'gather-scatter' approach for the FFT problem. This approach is currently implemented on a GPU version of a molecular dynamics code such as LAMMPS. ICHEC staff is currently in the process of implementing this technique with the PWscf code. Although no results are currently available, this implementation will be included within the next (version 5.0) official release of the package.

5.6 Conclusions

This sub-task aimed at contributing to projects for both Task 7.1 users and Task 7.2 application codes. In the case of Quantum-ESPRESSO, a full GPU version of the code has been developed. Users can download the freely available GPU version of the PWscf code to perform simulations on small to large-scale hybrid systems equipped with NVIDIA GPUs. The package along with the phiGEMM library is available in QE-Forge [19]. ICHEC staff will present this project at the next GTC (GPU Technology Conference) – the most renowned GPGPU conference worldwide. The phiGEMM library has been mentioned and analysed in a number of articles and relevant talks. An Italian researcher was recently granted PRACE Type C preparatory access for a project proposal that implements a real-world scientific application on the GPU enabled nodes of the CURIE Tier-0 system.

A comparison between performance results for the CUDA and OpenCL versions of DL_POLY was presented. Currently, CUDA outperforms OpenCL for all test cases, but additional work on tuning the OpenCL version will be performed within WP9-PRACE1IP and WP12-PRACE2IP. Despite the disparity in terms of performance, the value of having a

portable hybrid version of the code based on an open standard is fully recognized by the lead DL_POLY developers. In a separate project a performance analysis of 3-D FFTs on GPUs was carried out, delivering a valuable starting point for developers who want to implement GPU-enabled FFT libraries within their scientific codes.

Additional operators, including the MPI version of an existing GPU-based operator, were developed to extend the QUDA user community. The addition of these kernels can either be used for analyzing existing gauge configurations for the calculation of key hadronic observables, or included in existing simulation codes to accelerate certain algorithms used in the generation of gauge field configurations. An analysis of the state-of-the-art of the main GPU linear solver libraries available for OpenFOAM was performed.

From the work performed in this sub-task, it was found that at the time of writing, the enabling of scientific codes on accelerator-based architectures requires a considerable amount of effort. A successful case study was presented where ICHEC and CINECA GPU computing experts worked together with scientists to enable the PWscf code (part of the Quantum-ESPRESSO suite) to large-scale systems equipped with NVIDIA GPUs. This project required around 12PMs. Smaller or more fragmented amounts of effort can be considered, either to perform analysis for further development projects, or to improve the software environment but they will not be sufficient to enable complex code to run on distributed and accelerated systems.

Accelerators are currently an important opportunity for supercomputing. However, in most cases it is found that, while GPU-enabled scientific applications strongly outperform the CPU-based versions on the small scale, (in some cases on several few distributed nodes) they loose this performance benefit on a larger scale. Normally when developing GPU-based distributed applications, one has to contend with two main issues when increasing the number of compute nodes: balancing GPU and CPU workloads requires more effort and communication becomes the main bottle-neck due to the lack of bandwidth between distributed GPUs. Better integration between the accelerated co-processors and the back (CPU+memory) system may certainly help toward solving these problems. Improving both the experience and expertise within the scientific community along with more mature software environments would also help to reduce the current effort needed to enable complex scientific applications. Intel is promising this solution with the MIC product and its compiler. On the other hand NVIDIA is also progressing in both the areas of technology and software. Both Denver and the European Mont-Blanc projects have the goal of proposing a more integrated and low-power consumption system for the next generation of supercomputers along with the longer-term Exascale project. OpenACC project aims to facilitate the porting of legacy software.

For completeness of this work the sub-task leader completed an informal survey across some of the PRACE partners that are hosting Tier-1 GPU platform. Relevant people from Juelich, ICHEC, CINECA and PSNC have shared their experience on both GPU enabled production environment and software management. While most of the interviewed reported positive experiences from users who have accessed to GPU resources, all confirmed the concerns reported previously. Accelerators are still delivered as an additional piece of hardware plugged on top of traditional CPU system. Other than the introduction of an additional point of failure, this requires extended effort for monitoring and software management (drivers, resource manager, tool, libraries, etc.). Accounting GPU usage is still an open problem as well as it is in PRACE and there are no reliable systems to manage efficiently mixed classes of users on the same system (CPU only, CPU and GPU, heavy GPU) to maximize the resource usage. In the roadmap to Exascale accelerators must be considered as an important opportunity for sustainable solution but a lot more need to be done.

6 Novel HPC Languages

A number of novel programming languages have been developed with the aim of addressing the complexity of parallel programming. Unlike for instance GPGPU languages, which trade code complexity for potentially high performance, most novel programming languages raise the conceptual level of parallel programming in the hope to increase programmer productivity while at the same time maintaining relatively high level of performance.

Over the last years a few novel concepts have made their way into the awareness of the HPC community as they target some of the shortcomings (related to performance and programmer productivity) of traditional HPC programming models, particularly as we aim for Exascale sized systems. Specifically we need new ways to access remote memory that allow the runtime to hide network latencies, to describe data-parallelism parallelism in our codes which is exploited efficiently by SIMD- or vector-like execution units, and to schedule coarse-grained concurrent sections of code with negligible synchronization overhead.

The concept of a *partitioned global address space (PGAS)* allows to access remote data, as for instance an element of an array, much like local data or array elements. The programmer need not send and receive data explicitly as is the case with the message-passing paradigm. Instead, a suitable runtime or library takes care of the necessary data transfer in the background. The advantage is, that the code developer may often write code in much the same way as on a shared-memory system without caring about data distribution. Various project partners have investigated PGAS languages, in particular Coarray Fortran (CAF), Unified Parallel C (UPC), and Chapel. Of these languages, CAF has the lowest level of abstraction (remote array elements are referenced through an additional array index), followed by UPC (array indexing does not betray distributed nature; loops with data-parallel semantics), and finally Chapel with the highest abstraction level (fully transparent, arbitrary data distribution; data-parallel semantics not limited to loops; task-parallelism).

A range of programming models address *data-parallelism* in a shared memory space. OpenMP, the most widely used example of this programming paradigm, is characterized by its directive-based approach. Source code annotations inside comments are interpreted and translated into executable code only by a suitable compiler and ignored by all others. The approach results in very portable applications but suffers from limited language expressiveness. The novel model Array-Building Blocks (ArBB) takes the opposite approach and provides a C++ class and template library offering a range of container objects (e.g. vector, matrices) with data-parallel semantics; it also provides methods to do typical operations on container objects including for instance complex parallel loops or reductions. The library takes care of scheduling operations on those objects taking into account data layout and the architecture of the systems.

Task-parallelism is a more general concept than data-parallelism. Different parts of an application (tasks) may be executed in parallel as long as they are independent. In most programming models of this family the programmer needs to make sure that dependent tasks do not run in parallel by using locks and synchronization barriers (as done for instance in pthreads or OpenMP via directives). The language Cilk also uses nesting of tasks to guarantee proper execution order (and to implement data-parallelism). The StarSs programming model allows specifying data-dependencies between tasks. The resulting task dependency graph is then used to execute tasks out of order similarly to the super-scalar processor architecture.

The next sections briefly summarize the individual projects done as part of sub-task 7.5.E, respectively. Each summary points to detailed technical whitepapers or scientific publications if available. The chapter concludes with a high-level discussion on the topic of novel HPC programming languages.

6.1 Evaluating the UPC approach for HYDRO

6.1.1 Project Summary

Contributors: Jean-Michel Dupays (IDRIS), Dimitri Lecas (IDRIS)

Publications:

PRACE technical whitepaper: HYDRO. Pierre-François Lavallée, Guillaume Colin de Verdière, Philippe Wautelet, Dimitri Lecas, Patrick Corde, Jean-Michel Dupays

Application Area(s): CFD

Hardware platform(s): IBM P755, Cray XE6 (Hermit)

Programming language(s): UPC

Profiling/debugging tools: None

Libraries: None

Brief project description:

HYDRO is a 2D Computational Fluid Dynamics code (~1500 lines) that solves Euler's equations with a Finite Volume Method using Godunov's scheme and a Riemann solver at each interface on a regular mesh. The purpose of this project was to investigate transparent remote-memory access offered by the PGAS language UPC as a replacement for explicit message-passing with MPI. Note, that HYDRO is used as vehicle in various other investigations within WP7 and WP9.

6.1.2 Contribution

In order to understand how to deal with the domain decomposition in HYDRO, we first study a simple example with a 2D heat conduction resolution. Since the distribution (blocksize) in UPC must be known at compilation time and we only known the domain size at execution time, we have implemented three methods to do the domain decomposition. In the first method, we use a fixed block size at compilation time; in the second one each UPC thread (i.e. each instance of the distributed application) manages one block and the size of this block is identical between threads; in the third method each UPC thread manages one block but the size can be different between threads. The three methods have poor performance in comparison with the C version; this is mainly due to the use of a pointer to shared variable instead of the C pointer arithmetic. On our Power7 P755 with Berkeley UPC the use of pointer-to-shared is twenty time slower than the use of a regular C pointer. We use the first method for the UPC implementation of HYDRO since this method minimizes the modification of the code. Around 50 lines have been modified, mainly to add the shared attribute in declaration and definition functions and 20 lines were added mainly in the initialization. The code changes have been implemented in three days, without taking into account the time spent on the different ways to manage the domain decomposition.

The characteristics of the data set:

- Number of points of the domain: $N_x * N_y = 10000^2 = 10^8$
- Number of iterations: 10
- Total memory footprint: 3GB

We ran the test case on two platforms:

- IBM P755 (IDRIS, France), 32 cores per node, 128 Gb/node, Berkeley UPC compiler
- PRACE Tier-0 system CRAY XE6 Hermit (HLRS, Germany), 16 cores per node, 32GB/node, Cray UPC compiler

Results in term of elapsed time and scalability are presented in the two following tables:

	C	UPC					
	Mono	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
Elapsed time (s)	1112	1722	868	440	321	217	131
Speedup	1	0.7	1.3	2.5	3.5	5.1	8.5

Table 14: UPC HYDRO performance on IBM P77

	C	UPC					
	Mono	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
Elapsed time (s)	2287	2669	1948	1140	644	360	200
Speedup	1	0.9	1.2	2	3.6	6.4	11.4

Table 15: UPC HYDRO performance on CRAY XE6

In the most compute intensive part of the code, i.e. the Godunov subroutine, HYDRO uses temporary working buffers for reading values from the shared domain, so all computation are done without using any pointer-to-shared data. This additional data movement explains why the sequential UPC version has less performance than the sequential C version, however not as dramatic as expected from the 2D heat conduction UPC experimentation. The code changes have been implemented in three days, without taking into account the time spent on the different ways to manage the domain decomposition. We spent roughly 15 days experimenting with the different options to do the domain decomposition in order to evaluate benefits and drawbacks of each

6.1.3 Lessons learned

UPC provides a way to quickly add a global view of memory to a sequential code. It makes it easy to write a parallel shared memory version of an existing C code. But UPC suffers from severe limitations leading to poor performance when using a naïve implementation:

- The blocksize is required to be known at compile time, making it hard to get a good cache re-use.
- The distribution is one dimensional, making a 2D-block distribution impossible without using a shared array of pointers to local arrays.
- There is no equivalent to MPI datatypes, communicators or OpenMP schedule constructions.
- In order to achieve performance, we sometimes had to use the same decomposition that was used in the MPI version (i.e. halo cells around compute domain).
- The compilers are still in experimental phase (with varying levels of maturity).
- Data races must be explicitly avoided using synchronizations (barrier, locks, etc).

6.2 Parallel Benchmark Suite for Fortran Coarrays

6.2.1 Project Summary

Contributors: David Henty (EPCC)

Publications:

> A Parallel Benchmark Suite for Fortran Coarrays, D. Henty, in: Proceedings of the International Conference on Parallel Computing (ParCO), 2011, 30 August - 2 September 2011, Ghent, Belgium.

Application Area(s): Synthetic benchmark

Programming Language(s): Fortran Coarrays

Hardware platform(s): Cray X2, Cray XT4, Cray XE6, Intel Infiniband Cluster

Profiling/debugging tools: None

Libraries: None

Brief project description:

Coarrays are a feature of the new Fortran 2008 standard that enable parallelism using a small number of additional language elements. A new array declaration syntax allows for remotely accessible variables, with data allocated across multiple images. The execution model is that of a Partitioned Global Address Space (PGAS) language. Since Fortran coarrays are in their infancy, and full compiler support has only recently emerged, it is important to understand the performance characteristics of any parallel operations. Although benchmark suites exist for well-established models such as MPI, OpenMP and UPC, none are currently available for coarrays. The results of such benchmarks are important as they guide both the applications programmer and the compiler or library developer. In this paper we describe a low-level benchmark suite for Fortran coarrays that measures the performance of a selection of basic parallel operations. We present initial performance results on Cray architectures and a general-purpose Intel cluster. We hope this suite will help in the development and uptake of these new parallel features of the Fortran language.

6.2.2 Contribution

The benchmark code was written from scratch and run on a number of platforms. All work was done by the author except for some benchmark runs on internal Cray development systems, which were done by colleagues from Cray.

This project aimed to see if bottlenecks caused by MPI performance at large scale could be solved by using Fortran coarrays instead. It was shown that in some situations, coarrays can outperform MPI on platforms with appropriate hardware support. Performance results were collected on a range of platforms (see the paper for sample results).

6.2.3 Lessons learned

Although Fortran coarrays are relatively new, their performance can be very good on modern Cray hardware (such as the XE6), which has native compiler support and a communications network (GEMINI) that is optimized for remote memory access. Point-to-point synchronization can give better performance than global synchronization on large numbers of cores when the number of neighbours is relatively small (e.g. less than 10). On Cray systems without hardware support (e.g. the XT4 with Seastar2+ interconnect) performance is poorer than MPI. Coarrays are supported on general clusters in the recent Intel compilers. However,

performance is generally very poor and only the most basic synchronization operation (sync all) was supported at the time of the study.

6.3 Chapel

6.3.1 Project Summary

Contributors: José Gracia (HLRS)

Publications: None

Application Area(s): Synthetic benchmark

Hardware platform(s): Intel Nehalem Cluster, Cray XE6 (Hermit)

Programming language(s): Chapel

Profiling/debugging tools: None

Libraries: None

Brief project description:

The purpose of the project is to evaluate the asynchronous PGAS language Chapel in an HPC context. Chapel is a new PGAS-like programming language developed by Cray, initially as part of the DARPA HPCS project. The language features a number of very high-level abstractions, as for instance the concepts of *locale* (execution site, e.g. compute node), *domain* (set of indices over which variables are defined), and *distribution* (mapping of domain elements to locales). These features allow for simple code making use of transparent remote memory access and powerful data parallelism. However, Chapel has been known to suffer from low performance, in particular for the Euroben kernels used in PRACE-PP (see deliverable PP-D6.6). The objective of this study is to monitor the performance of Chapel for typical HPC application kernels as development and optimizations of the language and its runtime progress.

6.3.2 Contribution

The project built on the Chapel ports of the Euroben kernels mod2am/MxM and mod2as/SpMxV, which have been done during PRACE-PP based on Chapel v0.9. The kernels have been ported successively to Chapel version 1.1, 1.2, 1.3, and 1.4, to follow the further evolution of Chapel, both regarding its high-level syntax, but also performance optimizations in the runtime. The ports were targeted to exploit performance optimizations (both, by the compiler and manually by the programmer) that become accessible in new Chapel versions. While we have kept to the general guideline to write simple Chapel code, we have considered refactoring the code to allow the compiler or runtime to do certain performance optimizations that currently are not possible otherwise.

6.3.3 Lessons learned

Over time we have observed clear improvement of Chapel for the kernels under consideration regarding single-node, multi-threaded performance. With some hints by the developer, Chapel is able to produce code with performance within a factor of two of hand-written C code. Also, it is now possible to link Chapel code to C, e.g. to use the LAPACK library. This however, was not tested. Regarding the multi-node performance, we see improvement over time, particularly on low latency networks as the Cray Gemini, but the performance is still at least an order of magnitude lower than that of MPI. The main bottleneck remains the fact that

remote memory accesses continue to happen at a very low granularity, in some instances even element-by-element. While the Chapel team has optimized whole-array transfers to be done efficiently in a single message, this does not help to improve the performance of the two EuroBen kernels investigated here.

Conceptually, Chapel is one of the most powerful HPC languages available. Chapel code in general is simple and clear as it allows separating the actual algorithm from implementation details concerned with data distribution and movement. However, the compiler and runtime implementation have not yet matured to deliver production quality performance for realistic HPC problems.

6.4 Intel Array-Building Blocks (ArBB)

6.4.1 Project Summary

Contributor: Volker Weinberg (LRZ)

Publications: PRACE technical whitepaper: V. Weinberg, Data-parallel programming with Intel Array Building Blocks (ArBB)

Application Areas: Synthetic benchmarks, linear solvers

Hardware platforms: SuperMIG, Intel MIC based prototype machine

Programming language: Intel Array Building Blocks (ArBB) v1.0.0.030

Profiling/debugging tools: None

Libraries: libarbb, MKL v10.3

Project description:

The goal of the project is to evaluate the C++ based programming language Intel Array Building Blocks (ArBB). ArBB is a high-level data-parallel programming environment designed to produce scalable and portable results on existing and upcoming multi- and many-core platforms. It is a combination of the former Intel Ct technology and the RapidMind development platform. The latter has been evaluated by LRZ during PRACE-PP (see deliverable PP-D8.3.2).

6.4.2 Contribution

We ported the following mathematical kernels as representatives of scientific codes to ArBB:

- a dense matrix-matrix multiplication (mod2am/MxM)
- a sparse matrix-vector multiplication (mod2as/SpMxV)
- a 1-D complex Fast Fourier Transformation (FFT) (mod2f/FFT)
- a conjugate gradients solver for sparse linear systems (CG)
- a Gauss-Seidel and a Jacobi solver

Performance measurements are done on LRZ systems:

- SuperMIG. Intel Xeon Westmere-EX@2.4 GHz, 9.6 GFlop/s double precision peak performance per core, 40 cores per node, 256 GB shared memory per node
- Intel Many Integrated Core (MIC) architecture prototype machine.

The ArBB code is optimised for the underlying hardware architecture at run-time by a JIT-compiler. Due to NDA restrictions we only present the Westmere-EX based results (double precision) in Figure 51.

Particularly for mod2am we tested a variety of possible ArBB implementations (Figure 51(a)). The highest ArBB performance (64% of peak on a single core) could be obtained with the optimised mod2am port `arbb-mxm2`. The performance of code compiled with the current version of ArBB is still rather poor compared with MKL. For large data sets the codes mod2am and mod2as scale up to approximately 15 cores (Figure 51 (b)) and 30 cores (Figure 51 (d)), respectively. In other cases, especially for the FFT, scaling is insufficient.

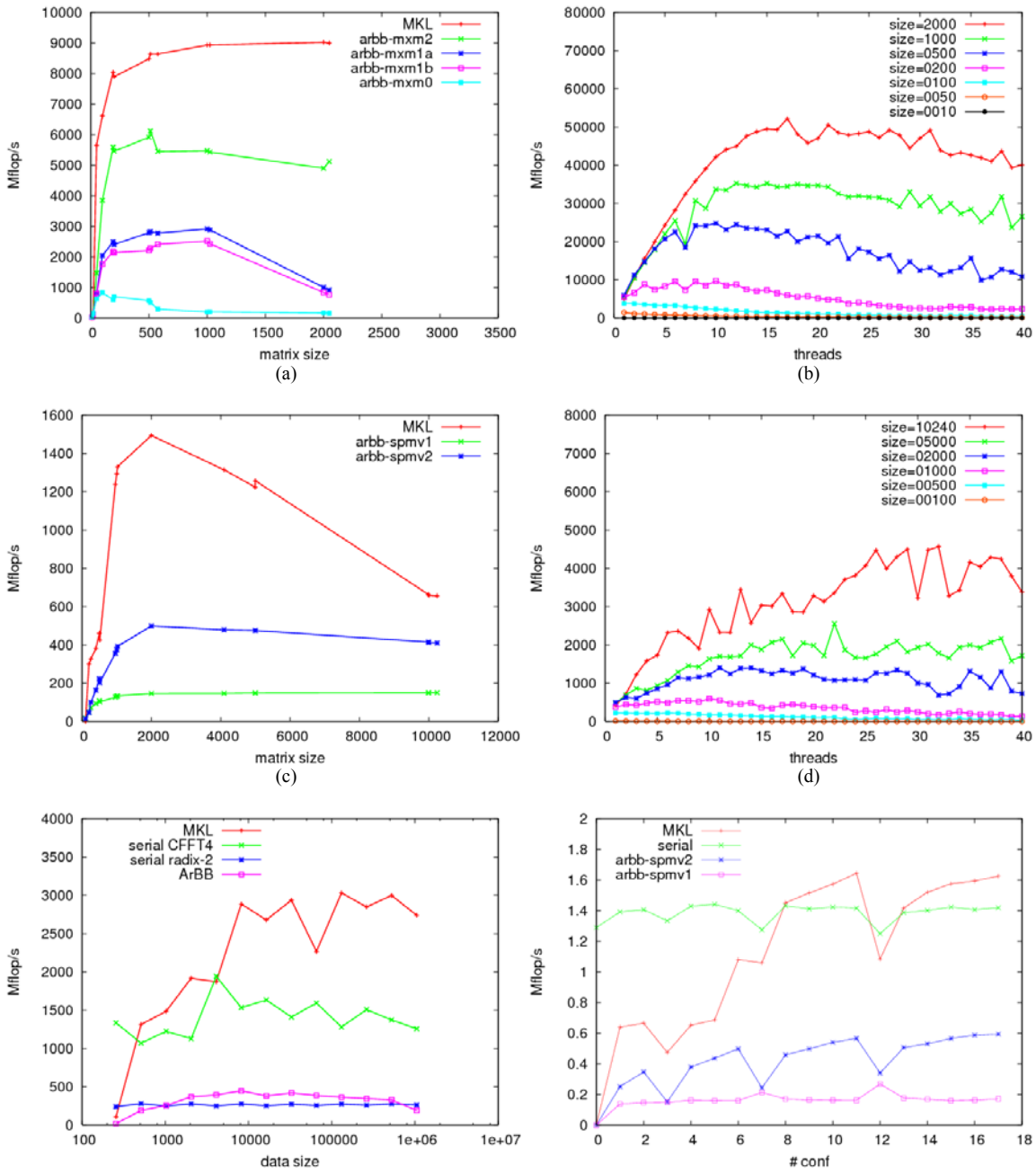


Figure 51: Single-core performance of various ArBB implementations in comparison with MKL for the EuroBen kernels mod2am (a/b), mod2as (c/d), mod2f (e) and the conjugate gradients solver (f). In (e) and (f) we also show the performance of serial implementations. Scaling of the optimised mod2am ArBB port arbb-mxm2 with the number of threads (as specified by the environment variable ARBB_NUM_CORES) is presented in (b) for various matrix sizes, the scaling of the mod2as ArBB port arbb-spmv2 is shown in (d).

6.4.3 Lessons learned

Intel ArBB is a powerful language for expressing data-parallelism in a simple way. The main advantages of ArBB are the availability of various operations for manipulating vectors and matrices and the simple, serial math-like semantics to express parallelism. The development time using ArBB is rather low for people who are used to program in C++. ArBB is currently limited to x86 based shared memory systems and (under NDA) the Intel MIC architecture. Compared to the former RapidMind product (which supported many- and multi-core CPUs, GPGPUs and the Cell processor) the portability of ArBB is very limited. The performance and the scaling of code compiled with the current version of ArBB are still rather poor. Intel decided not to merchandise ArBB as a product in near future.

6.5 Cilk and hybrid UPC/Cilk Programming

6.5.1 Project Summary

Contributors: Frederic-Gerald Morcos (JKU), Martin Polak (JKU), Volker Strumpen (JKU)

Publications: None

Application Area(s): Benchmark, cache-oblivious algorithms

Hardware platform(s): Distributed memory architectures with fat shared-memory nodes

Programming language(s): Cilk, UPC

Profiling/debugging tools: None

Libraries: FFTW, MKL

Project Description:

The goal is to explore an alternative to the de-facto standard programming model for exascale architectures, the mixed MPI-OpenMP model. We have evaluated UPC-Cilk as an interoperable alternative hybrid model, because it offers a uniform shared memory programming interface. The key points of interest are expressivity, performance, and scalability.

6.5.2 Contribution

We have applied the hybrid UPC/Cilk programming model to a cache-oblivious matrix transposition. UPC was used for the distributed memory parallelization across multiple nodes, while Cilk took care of the shared-memory parallelization on the node. We have achieved speedups up to 4 compared to the proprietary Intel (MKL) implementation using MPI/OpenMP. Throughput of the inter-node network was identified as the primary bottleneck.

Experimental Results: We report the performance of a cache-oblivious matrix transposition with UPC/Cilk, and, for comparison, of a matrix transposition using the proprietary Intel MKL library based on MPI and OpenMP. We performed iso-memory experiments across nodes, i.e. each node uses the same amount of memory and nodes are added proportional to increasing the problem size. Figure 52 (left) shows the execution time of UPC/Cilk, Figure 52 (right) the execution time of MKL over different problem sizes and numbers of nodes and Figure 53 shows the speedup of UPC/Cilk over MKL. Each curve in the plots corresponds to a different number of cores employed per node.

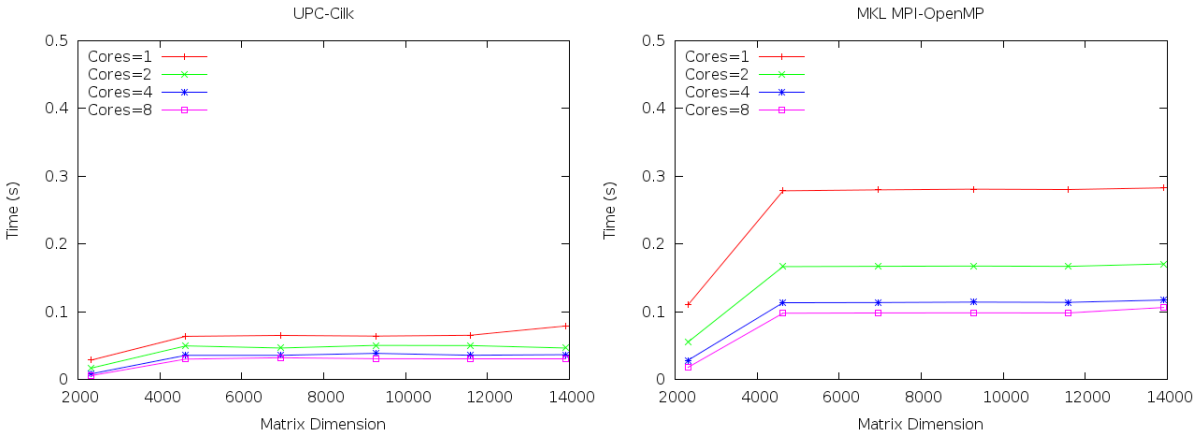


Figure 52 Iso-memory execution times of cache-oblivious matrix transpose with UPC/Cilk (left) and MKL (right). Different curves represent different number of cores per node, while the number of nodes increases in relation to the matrix size.

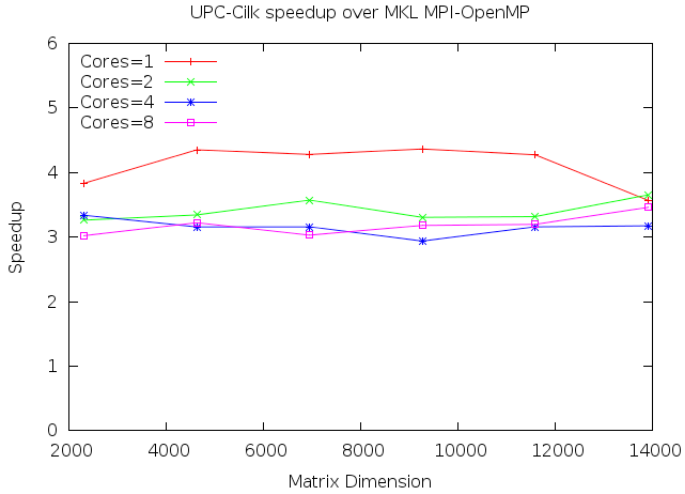


Figure 53 Speedups of iso-memory matrix transpositions comparing UPC/Cilk with Intel's MKL. Different curves represent different number of cores per node, while the number of nodes increases in relation to the matrix size. We find that our UPC/Cilk implementation outperforms MKL by a factor of 3-4, depending on the number of cores used per node.

6.5.3 Lessons learned

UPC provides an expressive yet concise language for abstract modeling of parallel computations on distributed memory machines. We mixed a UPC matrix transpose with Cilk using a parallel cache-oblivious transposition algorithm at the shared memory level (a single node) and demonstrated promising results over the Intel proprietary MKL with MPI and OpenMP. We draw the following conclusions:

- UPC presents an efficient, concise and expressive alternative to MPI.
- Cilk provides a parallelization model well suited for recursive cache-oblivious algorithms.
- Mixed UPC/Cilk programming is an abstract yet efficient tool for large parallel computations.
- Cache-oblivious algorithms are a major advantage for memory-intensive computations.

6.6 Hybrid Programming with MPI/StarSs

6.6.1 Project Summary

Contributors: José Gracia (HLRS)

Publications:

Hybrid parallel programming beyond MPI/OpenMP. J. Gracia, Ch. Niethammer, M. Hasert, S. Brinkmann, R. Keller. In: Proceedings of Cray User Group Meeting (CUG), 2012, 29 April – 3 May, Stuttgart, Germany.

Hybridising MPI with the asynchronous task parallel programming model StarSs. J. Gracia, Ch. Niethammer, M. Hasert, S. Brinkmann, R. Keller, C. W. Glass, International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2012, 10 – 13 July, Madrid, Spain.

Application Area(s): CFD, Lattice Boltzmann Method

Hardware platform(s): Intel Nehalem Cluster, Cray XE6 (Hermit)

Programming language(s): Fortran, MPI, StarSs/SMPSs

Profiling/debugging tools: Temanejo, Paraver

Libraries: None

Brief project description: StarSs is a novel task-based parallel programming model developed at BSC. Unlike the OpenMP tasks model which follows a classical fork/join semantics, StarSs uses the actual arguments passed in and out of a given task function to build a runtime task dependency tree. This allows the programmer writing parallel applications without explicit synchronization between tasks. Data dependencies between tasks are used to drive an efficient scheduler which takes into account data locality and has the ability for look-ahead down the task dependency tree. The purpose of this project was to combine StarSs with MPI for distributed parallelisation and test the model on a production Lattice-Boltzmann code [31].

6.6.2 Contribution

Lattice-Boltzmann, as most applications in scientific computing, is naturally data-parallel; it consists of three consecutive algorithmic steps: compute, apply boundary conditions, and exchange ghost cells with neighbours. In order to generate a large number of tasks, we have added an additional layer of domain decomposition (referred to as tiles) on top of the existing MPI domains. Next, the algorithm was blocked by introducing an additional loop spawning all tiles. Inside the loop the actual compute subroutine is called with a tile rather than the original MPI domain as argument. Essentially, this is a kind of multi-dimensional loop blocking, but applied to the code in its entirety (and thus not suitable as an optimization technique to address cache-misses due to temporal blocking). Data dependencies are chosen such, that task doing calculations on outer tiles, i.e. those that contain data that needs to be communicated via MPI to neighbours, can be scheduled early on with high priority in other to accelerate the critical path leading to the subroutine doing ghost cell exchange. While MPI communication is being done, the runtime continues to schedule those tasks that perform calculations on inner tiles.

The resulting code was benchmarked against the original MPI-only version, as well as a hybrid OpenMP/MPI version. The OpenMP part uses tasks similar to the StarSs version, but requires synchronization barriers were StarSs uses data dependencies. The hybrid MPI/StarSs

version already out-performs the MPI-only version on a single node by a small but statistically significant margin. Across node, the StarSs/MPI version scales much more efficient than the MPI version (see Figure 54). Performance modelling showed that the ratio of the scaling efficiencies of MPI/StarSs over MPI-only is equal to the number of StarSs threads per MPI process, in our case the full eight cores on a node. The hybrid MPI/StarSs model thus leverages the full potential of hybridization, at least for the code and use-case under consideration. It also out-performs the hybrid MPI/OpenMP version.

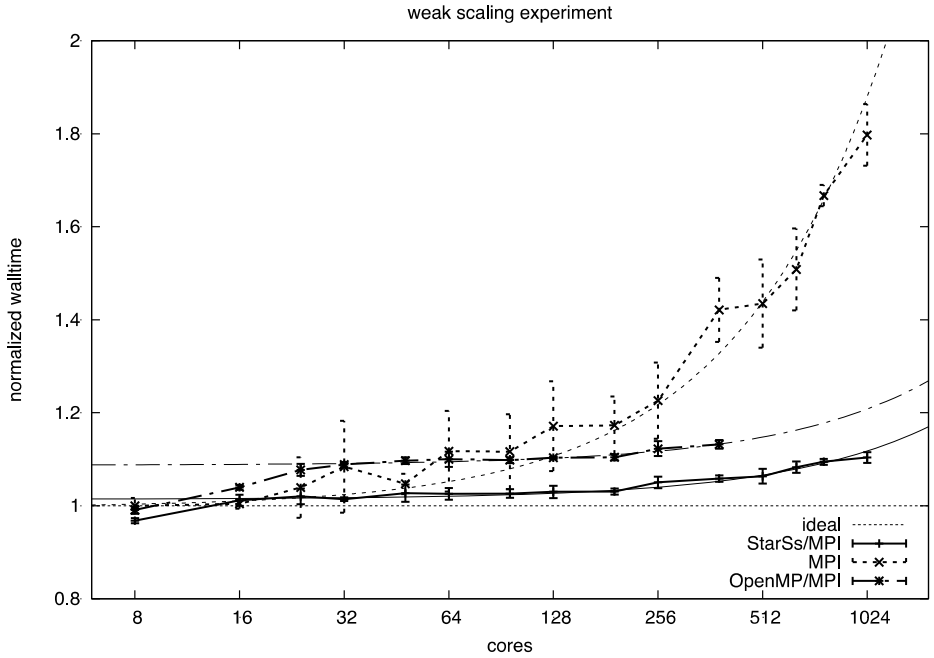


Figure 54 Weak scaling performance comparison for hybrid MPI/StarSs, hybrid MPI/OpenMP, and MPI-only versions. The smooth lines are fits to performance models.

6.6.3 Lessons learned

Applying the task-parallel programming model StarSs to a data-parallel application will in general require the developers to re-factor their code. In our study, however, the re-factoring benefitted the MPI-only version as well.

The focus on data dependencies between tasks allows for an application without any explicit synchronization. In our case the scheduler was able to keep all compute cores busy at all times; also the scheduler was able to overlap MPI communication with computation thus effectively hiding communication latencies. The hybrid OpenMP/MPI version is not able to hide these latencies and suffers from load-imbalances at synchronization barriers.

The StarSs/MPI version scales much better than MPI-only; more specifically by a factor equal to the number of StarSs threads per MPI process. For a given problem size, it will thus scale further out by the same factor in terms of compute cores. On machines as the Cray XE6 Hermit at HLRS, this factor could be as large as 32.

6.7 Conclusions

None of the novel parallel programming languages or models discussed in the previous sections has emerged as clear general-purpose substitute for the standard OpenMP/MPI (or pure MPI), even if some models, as e.g. UPC/Cilk or MPI/StarSs, clearly outperform classical models for certain problems or parallel patterns.

This is particularly true if one accepts the premise that most developers of production codes will be prepared to port their applications (ideally incrementally), but very reluctant to write them from scratch. From this perspective, the least *disruptive* approach is StarSs: the same production code will continue to run on traditional systems while part of the code is being ported. With few limitations the same is true for UPC as the new language keywords, could be masked by pre-processing macros. To some extent this is also possible for Cilk, however the programming model lends towards recursive algorithms which are not widely spread in HPC. The fact that remote memory is addressed through an additional index in CAF makes it difficult in general to maintain a common code base. Technically, ArBB is just a library for standard C++, but in practise it should be considered a disruptive new language. Finally, Chapel is a complete new language and requires re-writing any given application from scratch.

Technically the biggest problems for HPC are (i) synchronization, as this will manifest load-imbalance, and (ii) latency of data access which leads to idling of compute cores. Most novel models do not address these problems satisfactorily.

Consider a three-fold nested loop, as for instance matrix multiplication. Doing remote data access in the innermost loop will severely harm performance as latency becomes dominant. Clearly no one would write such code in MPI. However, in UPC or any other PGAS-like language, the programmer might not realize that remote memory access is taking place. To date, the compilers do not take into account latencies and thus do not re-order loops nor do pre-fetch data efficiently (or do at least bulk transfer). In some cases they are not even allowed to do those optimizations as the loops imply sequential, in-order execution unless explicitly stated otherwise by the programmer. PGAS languages in practise do not have look-ahead capabilities and the programmer is often forced to mimic message-passing and initiate data transfers explicitly ahead of time.

Another big problem is synchronization among concurrent threads or processes. At every synchronization point in any model, compute cores will have to wait for late-arrivals. This manifests accrued load-imbalances. In some cases the programmer is aware of synchronization points. But particularly in data-parallel schemes, synchronization is done implicitly in order to guarantee sequential consistency of the memory model. Most models require the programmer to synchronize tasks manually, but do only provide very crude control over task dependencies as for instance global join operations.

Only StarSs takes into account true data dependencies between tasks. Ideally, a StarSs code does not require any implicit or explicit synchronization and will keep compute resources busy as long as the problem offers enough task concurrency. Also, the StarSs runtime has look-ahead capabilities based on the task dependency graph and may take scheduling decisions that serve latency hiding and accelerate the critical path through the graph. The latest developments on StarSs allow offloading compute kernels onto GPGPUs while taking care of data movements in order to optimize data locality. The problem of StarSs is that it does not offer any solution for data-parallelism other than refactoring code to convert data- to task-parallelism.

The big advantage of most novel programming models is that they make writing parallel code easier by raising the abstraction level and taking care of implementation details. If the number of source code lines or development time (of a skilled developer!) for a code written from scratch is taken as a metric, basically all models discussed here are more productive than classical OpenMP or MPI. The increase in productivity in general disappears if one adds performance to the metrics. However, in some cases this can be attributed to the still in-mature state of compiler and runtime (UPC, Chapel), or to missing semantics and expressiveness in the language (as the missing data-parallelism in StarSs).

The question however remains whether novel programming models aiming for a higher programmer productivity will have a chance to exploit future Exascale system efficiently *at all*. It is not unconceivable that the complexity of future architectures will force the programmer to *lower the abstraction level* and code closer to the hardware, not the opposite, in order to achieve anything near acceptable performance. In that case any code will need significant, *disruptive* refactoring or rewriting.

In any case application codes should be (re-)designed to address multiple levels of parallelism:

1. *Vectorization of innermost loop* to exploit vector/SIMD units as AVX or SSE.
2. *Fine-grained (blockable) loop parallelism* suitable today for e.g. OpenMP parallel loops or streaming processors on GPUs. Optionally the loops should be blockable to allow efficient cache re-use when running on cache architectures.
3. *Coarse-grain task parallelism* by laying out subroutines and their interfaces such that the code will work on a subset of its data; This could be used today with OpenMP tasks or StarSs.
4. *Disentanglement of computation and communication* will allow issuing communication in the background and thus overlapping it with computation. This approach will also make dynamic load-balancing simpler.
5. *Avoidance of global synchronization* (including global collectives); where possible this should be replaced by local synchronization between few partners or replaced with data-dependencies as provided by StarSs.

7 Summary and Conclusions

The main focus of WP7 is to increase performance and scalability of important user applications on Tier-0 level, improving also performance and scalability on Tier-1 level. While the overview of the 36 projects that formed Task 7.5 shows several successful approaches, the deliverable also illustrates how difficult it can be to substantially improve application performance. In many projects an incremental improvement is clearly visible; however the examples in which scalability or performance could be increased by an order of magnitude or more are missing.

The five different areas that were under investigation are:

- Scalable algorithms
- Scalable libraries
- Multi-/Many-core systems
- Accelerators
- Novel HPC languages

The main chapters of the deliverable cover those areas and come to their individual conclusions. The following paragraphs summarize the experiences gained from the different subtasks and highlights future works:

Several newly developed *algorithms* have been included in important PRACE applications or at least tested with realistic input data from such applications. In some cases this yielded interesting and impressive results, other projects seem to have underestimated the amount of time and effort necessary to include new algorithms in big software packages like modern application codes. It would be beneficial if following PRACE projects could be used to further shape and improve both the algorithms and their integration in existing software packages. For sure, the development of new algorithms is the most time and resource consuming way to prepare applications for new generations of high-end computing systems. In addition, it is clear that not all ideas and projects will be successful, but if an idea for a new algorithm is successfully transformed and embedded into an application, the development of new algorithms is the only possibility to substantially increase performance and scalability. Therefore, investment in the development of new algorithms is indispensable for the transition to Exascale computing.

The projects on *scalable libraries* feature several successful collaborations among PRACE partners in assessing new libraries and comparing libraries against each other. Their findings on dense linear algebra, sparse linear algebra and FFT libraries are of interest to a big part of the scientific computing community. In addition to the individual results, it should be stressed that, where suitable mathematical libraries exist, their usage should be one of the "best practices" used in any HPC code. Since all mathematical libraries are constantly improved and new libraries emerge, it is important to periodically compare their performance against each other, especially on the Tier-0 systems. The PRACE system guides contain further information about which libraries are available and contain recommendations on peculiarities of the systems.

To better utilize *multi-/many-core systems*, a number of projects ported existing MPI codes to a mixture of OpenMP and MPI - or tried to improve performance of such a hybrid parallelization scheme. Some projects were quite successful, e.g. the hybrid Hydro code showed significantly performance improvements over the pure MPI version for core counts above 16k cores. However, several projects demonstrated that obtaining performance improvements via hybrid parallelization is not straight forward. The overall results imply that the hybrid programming model is still not well understood. It is often unclear whether the MPI library severely limits performance or scalability and how this could be improved by

adding a second level of parallelization hierarchy. All projects that targeted Petascale enabling were able to show incremental improvements. Several of the projects which worked on Task 7.2 community codes will continue their work until the end of PRACE-1IP. Final results will then be reported in the corresponding white papers or the final deliverable D7.2.2 "Final report on collaboration with communities".

In the area of *accelerators* a limited number of GPGPU projects have been carried out. However, the interest in this area is huge thanks also to the availability of GPU equipped PRACE Tier-0 systems, like CURIE which has been recently extended with a 200 TFlop/s hybrid partition based on the latest NVIDIA M2090 GPUs. Furthermore, in many other centers, small to medium size GPU based systems are available. The results of the different projects illustrate the wide variety of potential experiences with accelerated systems. In some cases, porting to GPUs significantly increased performance but there are also projects were - at least preliminary- results indicate that porting to GPUs has little advantage. One should not forget the overhead that comes with a GPU enabled production code that often is just another branch that needs to be maintained. As long as the number of GPU experts in HPC stays quite small, developing and handling GPU codes is difficult. It seems more realistic to use GPUs in those cases where it is possible identify and isolate calls to common mathematical libraries that are available for GPUs. In those cases it is possible to get good performance improvements for a modest amount of porting effort without harming their maintainability or robustness.

The assessment of *novel HPC languages* is a continuation of the efforts on performance and productivity assessment started already in the PRACE preparatory phase. However, the state-of-the-art of those languages is still not convincing. Experimental languages like Chapel or ArBB follow a "revolutionary" idea but fail to deliver performance. More mature languages like the PGAS languages UPC and CAF, which are already supported by several compilers and interconnects, can sometimes deliver better performance than pure MPI code but the increase in performance is rather small and the programming effort is comparable; in fact the programming model is not very different. The use of Cilk shows good results but usually requires a major rearrangement of the basic algorithm. If this rearrangement is successful, Cilk can deliver quite good performance. StarSs is gaining more and more momentum; they have an interesting approach that is supported by a well performing backend.