SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2011-2.3.5 – Second Implementation Phase of the European High Performance Computing (HPC) service PRACE

PRACE-2IP

PRACE Second Implementation Phase Project

Grant Agreement Number: RI-283493

D8.3
Re-integration into Community Codes
Final

Version: 1.0
Author(s): Claudio Gheller, CSCS
Date: 21/08/2013

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-283493**	
	Project Title: PRACE Second Implementation Phase Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: D8.3	
	Deliverable Nature: Report	
	Deliverable Level: PU *	**Contractual Date of Delivery:** 31/08/2013
		Actual Date of Delivery: 31/08/2013
	EC Project Officer: Leonardo Flores Añover	

* - 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: Re-integration into Community Codes**	
	ID: D8.3	
	Version: <1.0>	**Status:** ***Final***
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D8.3.docx	
Authorship	**Written by:**	Claudio Gheller (CSCS)
	Contributors:	Thomas Schulthess, CSCS; Fabio Affinito, CINECA; Alastair McKinstry; Laurent Crouzet, Marc Torrent, CEA; Andy Sunderland, STFC; Giannis Koutsou, Abdou Abdel-Rehim, CASTORC; Miguel Avillez, UC-LCA; Georg Huhs and Mohammad Jowkar, BSC; Guillaume Houzeaux, BSC; Charles Moulinec, Xiaohu Guo, STFC; Vít Vondrák, David Horák, Václav Hapla, VSB; Peter Raback, Juha Ruokolainen, Mikko Byckling, CSC; Bärbel Große-Wöhrmann, Florian Seybold, HLRS; Andrew Porter, Stephen Pickles, STFC; William Sawyer, CSCS; Alexandra Charalampidou, Nikos Anastopoulos, GRNET
	Reviewed by:	Mark Bull, EPCC; Dietmar Erwin, FZJ
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	26/06/2013	First skeleton	
0.2	08/07/2013	Introduction added	
0.3	12/07/2013	Section 2 written	
0.4	23/07/2013	Inputs from partners added	
0.5	26/07/2013	First two sections improved	
0.6	03/08/0213	More inputs from partners	
0.7	06/08/2013	First partial proof reading	
0.80	09/08/2013	Final proof reading	
1.0	21.8.2013	Final version for approval	

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, scientific applications, libraries, performance modelling.

Disclaimer

This deliverable has been prepared by Work Package 8 of the Project in accordance with the Consortium Agreement and the Grant Agreement n° RI-283493. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the Project and to the extent foreseen in such agreements. Please note that even though all participants to the Project are members of PRACE AISBL, this deliverable has not been approved by the Council of PRACE AISBL and therefore does not emanate from it nor should it be considered to reflect PRACE AISBL's individual opinion.

Table of Contents

List of Figures

References and Applicable Documents

[1] http://www.prace-ri.eu
[2] Deliverable D8.1.1: “Community Codes Development Proposal”
[3] Deliverable D8.1.2: “Performance Model of Community Codes”
[4] Deliverable D8.1.3: “Prototype Codes Exploring Performance Improvements”
[5] Deliverable D8.1.4: “Plan for Community Code Refactoring”
[6] Deliverable D8.2: “Refactoring and Algorithm Re-engineering Guides and Reports”
[7] http://fusionforge.org/
[8] http://wiki.org/
[9] http://www.mediawiki.org/wiki/MediaWiki
[10] http://en.wikipedia.org/wiki/Main_Page
[11] http://wikimediafoundation.org/wiki/Home
[12] http://www.gputechconf.com/page/home.html
[13] http://on-demand.gputechconf.com/gtc/2013/poster/pdf/P0166_ramses.pdf
[14] http://www.itp.uzh.ch/rum2013/Ramses_users_meeting/Workshop.html
[15] http://www.itp.uzh.ch/~markusw/RUM_2013/Gheller1.pdf
[16] Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B.

Lang, H. Lederer, P. R. Willems, Parallel Computing 37, 783-794 (2011) http://elpa-lib.fhi-berlin.mpg.de
[17] http://icl.cs.utk.edu/plasma
[18] http://bazaar.canonical.org
[19] http://buildbot.net
[20] Organizing Software Growth and Distributed Development: The case of Abinit Pouillon Y, Beuken JM, Deutsch T, Torrent M, Gonze X Computing in Science & Engineering 13 (vol. 1), 62 (2011)
[21] http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
[22] http://user.cscs.ch/hardware/monte_rosa_cray_xe6/index.html
[23] http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html
[24] http://www.nvidia.com/object/cuda_home_new.html
[25] http://www.khronos.org/opencl/
[26] http://www.openacc-standard.org/
[27] https://developer.nvidia.com/cuda-fortran
[28] A G Sunderland, C J Noble, V M Burke and P G Burke, CPC 145 (2002), 311-340.
[29] UKRmol: a low-energy electron- and positron-molecule scattering suite, http://oro.open.ac.uk/33130/
[30] K L Baluja, P G Burke and L A Morgan, CPC 27 (1982), 299-307.
[31] IBM Engineering and Scientific Subroutine Library (ESSL), http://www-03.ibm.com/systems/software/essl/
[32] Eigenvalue SoLvers for Petaflop-Applications, http://elpa.rzg.mpg.de.
[33] CUDA Toolkit, http://docs.nvidia.com/cuda/cublas/index.html.
[34] Intel Math Kernel Library 11.0, http://software.intel.com/en-us/intel-mkl.
[35] The Matrix Algebra on GPU and Multicore Architectures project, http://icl.cs.utk.edu/magma/
[36] PRACE WP8 PFARM Wiki, http://hpcforge.org/plugins/mediawiki/wiki/pfarm
[37] CCPForge PRMAT/PFARM, http://ccpforge.cse.rl.ac.uk/gf/project/prmat-dcse/
[38] Software Engineering Group at STFC, http://www.stfc.ac.uk/CSE/randd/26606.aspx
[39] Apache Subversion, http://subversion.apache.org/
[40] Exomol database of molecular line lists, http://www.exomol.com/
[41] Centre for Theoretical Atomic, Molecular and Opical Physics, Queen's University, Belfast, http://www.qub.ac.uk/schools/SchoolofMathematicsandPhysics/Research/CTAMOP/
[42] http://www.efda.org/jet/jet-iter/
[43] http://goog-perftools.sourceforge.net/doc/tcmalloc.html
[44] http://hpcforge.org/plugins/mediawiki/wiki/icom/index.php/Main_Page
[45] Jason Holt, Adrian New, Hedong Liu, Andrew Coward, Stephen Pickles, Mike Ashworth, "Next Generation Ocean Dynamical Core Roadmap Project: Executive Summary", 2013.
[46] Jason Holt, Adrian New, Hedong Liu, Andrew Coward (NOC), Stephen Pickles, Mike Ashworth (STFC), "Next Generation Ocean Dynamical Core Roadmap Project: Final Report", 2013.
[47] Claire Lévy, "On the fly conclusions from the NEMO Enlarged Developer's Committee Meeting", Paris, June 2013.
[48] C. M. Maynard, M. J. Glover, N. Wood (The Met Office), R. Ford, S. Pickles (STFC Daresbury Laboratory), D. Ham (Imperial College), E. Mueller (University of Bath), G. Riley (University of Manchester), "Weather prediction and climate modelling at Exascale: Introducing the Gung Ho project", presented at EASC2013, Edinburgh, 2013.
[49] "Met Office/NOC: Joint White Paper", 2013. Available on-line at: https://forge.ipsl.jussieu.fr/nemo/wiki/2013WP/EnlargedDevComJune2013

[50] Stephen M. Pickles and Andrew R. Porter, "Developing NEMO for Large Multi-core Scalar Systems: Final Report of the dCSE NEMO project", 2012. Available on-line at: http://www.hector.ac.uk/cse/distributedcse/reports/nemo02/
[51] A. R. Porter, S. M. Pickles, M. Ashworth, "Final report for the gNEMO project: porting the oceanographic model NEMO to run on many-core devices", Daresbury Laboratory Technical Reports, DL-TR-2012-001 (2012).
[52] Andrew Porter, Stephen Pickles, Mike Ashworth (STFC Daresbury Laboratory), Giovanni Aloisio, Italo Epicoco and Silvia Movavero (CMCC), "Hybrid Strategies for the NEMO Ocean Model on Multi-core Processors", presented at EASC2013, Edinburgh, 2013.
[53] A coupled finite volume solver for the solution of incompressible flows on unstructured grids, M. Darwish et al., Journal of Computational Physics 228 (2009) p. 180-201
[54] https://launchpad.net/fluidity
[55] A coupled finite volume solver for the solution of laminar/turbulent incompressible and compressible flows, L. Mangani, C. Bianchini, 5th OpenFOAM® Workshop, June 22-24, 2010, Gothenburg, Sweden
[56] A fully integrated Coupled Solver using Block-GAMG acceleration, L. Mangani et al., 7th OpenFOAM® Workshop, June 25-28, 2012, Darmstadt, Germany
[57] Block-Coupled Simulations Using OpenFOAM, I. Clifford, 6th OpenFOAM® Workshop, June 13-16, 2011
[58] Conservative Regridding When Grid Cell Edges Are Unknown -- Case of SCRIP, J. Chavas et al., Technical Report, CEA, 2013 . http://arxiv.org/abs/1302.1796
[59] http://www.hector.ac.uk/cse/distributedcse/reports/prmat/
[60] http://qe-forge.org/gf/project/phigemm/
[61] http://www-03.ibm.com/systems/technicalcomputing/solutions/bluegene/
[62] https://en.wikipedia.org/wiki/Sandy_Bridge
[63] http://www.mcs.anl.gov/petsc/
[64] Colella, P.; Woodward, Paul R., Journal of Computational Physics (ISSN 0021-9991), vol. 54, April 1984, p. 174-201.
[65] Conservative Regridding When Grid Cell Edges Are Unknown -- Case of SCRIP, J. Chavas et al., Technical Report, CEA, 2013 . http://arxiv.org/abs/1302.1796
[66] http://www.unidata.uc
[67] ar.edu/software/netcdf/
[68] http://www.cs.sandia.gov/Zoltan/
[69] https://software.ecmwf.int/wiki/display/GRIB/Home
[70] https://sites.google.com/site/abinit2013workshop
[71] http://spomech.vsb.cz/feti/

List of Acronyms and Abbreviations

AMR	Adaptive Mesh Refinement
API	Application Programming Interface
BLAS	Basic Linear Algebra Subprograms
BSC	Barcelona Supercomputing Center (Spain)
CAF	Co-Array Fortran
CCLM	COSMO Climate Limited-area Model
ccNUMA	cache coherent NUMA
CEA	Commissariat à l'Energie Atomique (represented in PRACE by GENCI, France)
CERFACS	The European Centre for Research and Advanced Training in Scientific Computation
CESM	Community Earth System Model, developed at NCAR (USA)
CFD	Computational Fluid Dynamics
CG	Conjugate-Gradient
CINECA	Consorzio Interuniversitario per il Calcolo Parallello (Italy)
CINES	Centre Informatique National de l'Enseignement Supérieur (represented in PRACE by GENCI, France)
CNRS	Centre national de la recherche scientifique
COSMO	Consortium for Small-scale Modelling
CP	Car-Parrinello
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CUDA	Compute Unified Device Architecture (NVIDIA)
CUSP	CUda SParse linear algebra library
DFPT	Density-Functional Perturbation Theory
DFT	Discrete Fourier Transform
DGEMM	Double precision General Matrix Multiply
DKRZ	Deutsches Klimarechenzentum
DP	Double Precision, usually 64-bit floating-point numbers
DRAM	Dynamic Random Access memory
EC	European Community
ENES	European Network for Earth System Modelling
EPCC	Edinburgh Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
EPSRC	The Engineering and Physical Sciences Research Council (United Kingdom)
ESM	Earth System Model
ETHZ	Eidgenössische Technische Hochschule Zürich, ETH Zurich (Switzerland)
FFT	Fast Fourier Transform
FP	Floating-Point
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
FT-MPI	Fault Tolerant Message Passing Interface
FZJ	Forschungszentrum Jülich (Germany)
GB	Giga (= 2^{30} ~ 10^{9}) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^{9}) bits per second, also Gbit/s

GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GCS	Gauss Centre for Supercomputing (Germany)
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
GGA	Generalised Gradient Approximations
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPL	GNU General Public Licence
GPU	Graphic Processing Unit
HDD	Hard Disk Drive
HLRS	High Performance Computing Center Stuttgart (Germany)
HMPP	Hybrid Multi-core Parallel Programming (CAPS enterprise)
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HP2C	High Performance and High Productivity Computing Initiative
HPL	High Performance LINPACK
ICHEC	Irish Centre for High-End Computing
ICOM	Imperial College Ocean Model
ICON	Icosahedral Non-hydrostatic model
IDRIS	Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
IEEE	Institute of Electrical and Electronic Engineers
IESP	International Exascale Project
I/O	Input/Output
IPSL	Institut Pierre Simon Laplace
JSC	Jülich Supercomputing Centre (FZJ, Germany)
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
LBE	Lattice Boltzmann Equation
LINPACK	Software library for Linear Algebra
LQCD	Lattice QCD
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
MBPT	Many-Body Perturbation Theory
MCT	Model Coupling Toolkit, developed at Argonne National Lab. (USA)
MD	Molecular Dynamics
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
MIPS	Originally Microprocessor without Interlocked Pipeline Stages; a RISC processor architecture developed by MIPS Technology
MKL	Math Kernel Library (Intel)
MPI	Message Passing Interface
MPI-IO	Message Passing Interface – Input/Output
MPP	Massively Parallel Processing (or Processor)
MPT	Message Passing Toolkit
NCAR	National Center for Atmospheric Research
NCF	Netherlands Computing Facilities (Netherlands)
NEGF	non-equilibrium Green's functions,

NERC	Natural Environment Research Council
NEMO	Nucleus for European Modelling of the Ocean
NERC	Natural Environment Research Council (United Kingdom)
NWP	Numerical Weather Prediction
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
OS	Operating System
PAW	Projector Augmented-Wave
PGI	Portland Group, Inc.
PGAS	Partitioned Global Address Space
PIMD	Path-Integral Molecular Dynamics
POSIX	Portable OS Interface for Unix
PPE	PowerPC Processor Element (in a Cell processor)
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PSNC	Poznan Supercomputing and Networking Centre (Poland)
PWscf	Plane-Wave Self-Consistent Field
QCD	Quantum Chromodynamics
QR	QR method or algorithm: a procedure in linear algebra to factorise a matrix into a product of an orthogonal and an upper triangular matrix
RAM	Random Access Memory
RDMA	Remote Data Memory Access
RISC	Reduce Instruction Set Computer
RPM	Revolution per Minute
SGEMM	Single precision General Matrix Multiply, subroutine in the BLAS
SHMEM	Share Memory access library (Cray)
SIMD	Single Instruction Multiple Data
SM	Streaming Multiprocessor, also Subnet Manager
SMP	Symmetric MultiProcessing
SP	Single Precision, usually 32-bit floating-point numbers
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
STRATOS	PRACE advisory group for STRAtegic TechnOlogieS
TB	Tera ($=2^{40} \sim 10^{12}$) Bytes (= 8 bits), also TByte
TDDFT	Time-dependent density functional theory
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
UML	Unified Modelling Language
UPC	Unified Parallel C
VSB	Technical University of Ostrava (Czech Republic)

Executive Summary

This document collects and presents the results produced by Work Package 8 'Community Code Scaling' in PRACE-2IP (WP8) at the end of its two-year programme. The main focus of WP8 was the re-design and refactoring of a number of selected codes for scientific numerical applications, in order to effectively run on coming generations of supercomputing architectures, optimally exploiting their innovative features. This was achieved thanks to the working methodology developed, which was based on a close synergy between scientists, community software developers and HPC experts, each bringing their unique competences and skills. A further major outcome of this methodology was the promotion of a "culture" in high performance computing awareness among community software developers, consolidating their knowledge base to face the challenges posed by innovative computing systems, and enabling them to autonomously adopt and exploit the most appropriate software and hardware solutions.

WP8, together with scientists and community software developers from Astrophysics, Climate, Material Science, Particle Physics and Engineering, re-designed and re-factored eighteen codes. The accomplished work is presented together with the status of the code validation and re-integration process. Code validation was performed by community software developers and was based on a regression analysis needed to prove that the refactored algorithms are bug-free and give correct results. Such process was essentially incorporated in the WP8 refactoring methodology through the close day-by-day interaction among developers and HPC experts. Re-introducing the refactored codes back into the scientific community ensured that users are aware of the newly developed software features and that they are ready to adopt them as soon as they are available, maximising the impact of WP8. This was accomplished by means of personal contacts, participation and presentation of the WP8 achievements at conferences and workshops, and by publishing scientific and technical papers in specialised journals. Furthermore, WP8 implemented a specific programme to progressively consolidate the engagement of the scientific community, based on periodic face-to-face workshops. Finally, on-going and accomplished work, rewritten codes and libraries were documented and published on a dedicated wiki web site.

1 Introduction

Work Package 8 (hereafter WP8) was designed with the idea of supporting science by enabling numerical applications and simulation codes for coming generations of High Performance Computing (HPC) architectures.

Today, all successful development projects for sustained petaflops applications have in common that major portions of application codes have to be rewritten (software refactoring) and algorithms have to be re-engineered for the applications to run efficiently and productively on emerging architectures. This principle is not limited to the high-end of supercomputing, but applies to all tiers of the HPC ecosystem.

The effort involved in software refactoring can be substantial and often surpasses the abilities of individual research groups. Hence, WP8 promoted and created a close synergy between scientists, code developers and HPC experts, the first two contributing with their deep comprehension of the research topics and of the algorithms, and the latter providing the necessary skills and competencies in novel hardware architectures, programming models and software solutions, and driving the refactoring program in order to optimally map applications to coming supercomputers.

In addition to the service role that is inherent to this approach, an implicit dissemination objective was pursued. Application developers have been familiarised with programming environments, supercomputer architectures and innovative programming models, laying the foundations to the implementation of algorithms that map onto supercomputers well after the PRACE-2IP project is completed. More importantly, codes will likely map to Tier-1 and Tier-0 systems, and it will be the particular requirements of the users of these community codes to determine whether a project can be run on a Tier-1 system or should be scaled up to a Tier-0 supercomputer.

The work package has been organised into inter-disciplinary teams around the community codes. Participating HPC centres contributed programming environment and architecture experts. The scientific community committed software developers and domain expertise. Appropriate collaborative tools, day-to-day interactions, periodic meetings and workshops, and contributions to international conferences have represented the main instruments by which WP8's synergies have developed and strengthened. Details are given in Section 2.

The codes subject to WP8 refactoring were selected according to the *performance modelling* methodology, as described in deliverables D8.1.1-4 ([2][3][4][5]). The following table lists the name of the selected applications, their respective scientific domains and the PRACE-2IP partner leading their development:

Code name	Scientific Domain	Responsible partner
RAMSES	Astrophysics	ETH
PFARM	Astrophysics	STFC
EAF-PAMR	Astrophysics	UC-LCA
OASIS	Climate	CEA
I/O Services	Climate	ICHEC
ICON	Climate	ETH
NEMO	Climate	STFC
Fluidity/ICOM	Climate	STFC
ABINIT	Material Science	CEA
QuantumESPRESSO	Material Science	CINECA
SIESTA	Material Science	BSC
EXCITING/ELK	Material Science	ETH
PLQCD	Particle Physics	CASTORC
ELMER	Engineering	VSB-TUO
CODE_SATURNE	Engineering	STFC
ALYA	Engineering	BSC
ZFS	Engineering	HLRS
Coupled FVM Solver	Engineering	HLRS

Table 1: List of the codes selected for refactoring (left column), corresponding scientific domain (central column) and responsible PRACE partner (right column).

A preliminary overview of WP8's outcomes can be already found in deliverable D8.2 [6], together with the description of the WP8 wiki web site, http://prace2ip-wp8.hpcforge.org,

where information on on-going activities and open issues together with up-to-date results are collected and published.

The update of the results and achievements of WP8 is given in Sections 3 to 7. In these Sections the status of integration of each code in the respective developers' and users' community will be provided, in order to show the effectiveness of the adopted methodology in producing tools and software ready for scientific applications.

Finally Section 8 will summarise the content of this document and will draw some conclusions.

2 Working methodology and re-introduction procedure

The working methodology adopted by WP8 has been carefully designed in order to effectively coordinate the effort of a large number of developers and researchers, to blend broad and heterogeneous skills and expertise, and to maximise the impact of its outcomes on the scientific community.

Figure 1 shows the main features of such methodology. The work plan consists of three Tasks each characterised by a number of steps, some of them performed in parallel.

At the beginning of Task 1, scientific domains have been selected according to a set of criteria, the most relevant being their impact on science and society, the relevance of supercomputing to the scientific cases, and the willingness of the involved communities to contribute to the project and invest in software refactoring and algorithm re-engineering. According to the adopted criteria, Astrophysics, Climate Science, Material Science, Particle Physics and Engineering have been selected as the five scientific domains to start a joint and synergic action with the PRACE-2IP project. A strong involvement of scientists and community codes developers in the work package was key to:

- Addressing codes refactoring to the needs of the corresponding community;
- Involving the communities in the codes validation procedure;
- Facilitating the re-introduction of the re-implemented codes in the communities.

During Task 1, the community codes have been selected through the adoption of the performance modelling methodology, identifying the performance-relevant software components and libraries. Representative parts of these software components served as prototypes to test out possible performance improvements due to software refactoring and algorithm reengineering. Plans to rewrite relevant code kernels, along with strategies for regression testing and benchmarking (against relevant parts of existing codes), have been developed.

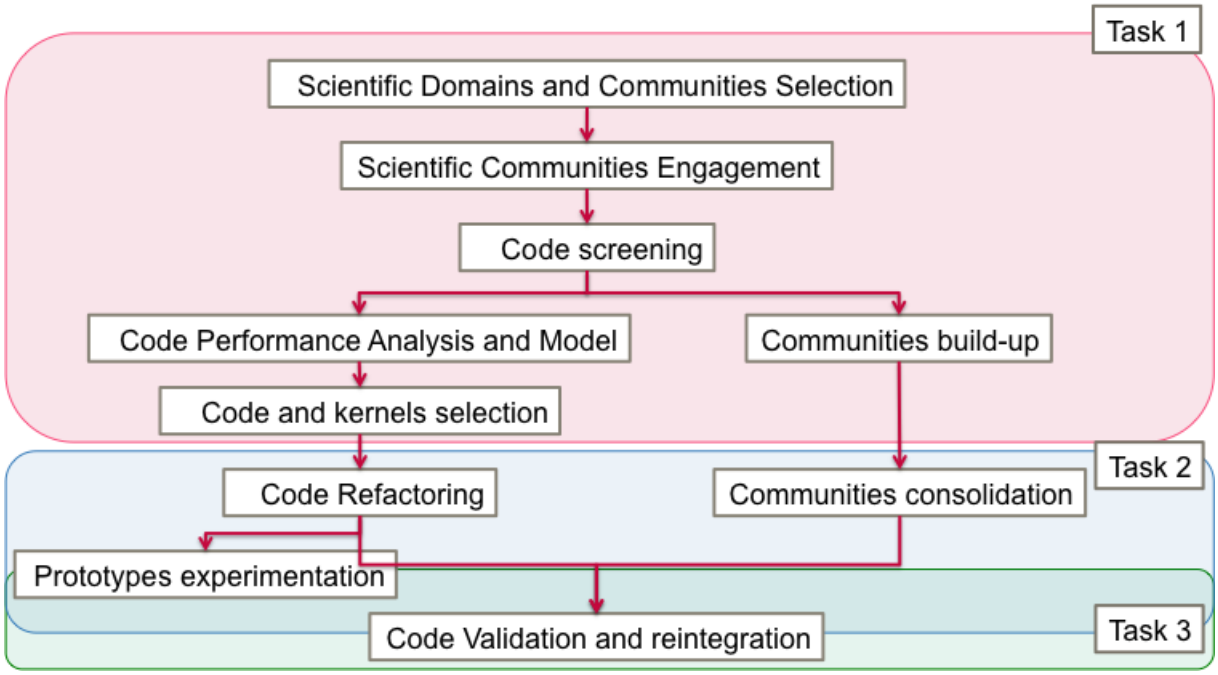

Figure 1: Sketch of the working methodology adopted for WP8.

In Task 2, performance-critical portions of the community codes were rewritten according to the plans setup in Task 1. Specific groups were formed to work on each of the selected code, involving both project partners and community members. Due to the different features, needs and targets of the codes, each group could adopt a different working methodology, strategy and schedule in order to reach the envisaged objectives in the WP8 time frame. Accomplished work, rewritten codes and libraries were documented and published, using the WP8 HPC-Forge and wiki web sites (https://hpcforge.org/ and http://prace2ip-wp8.hpcforge.org), together with systems already in use by the applications communities. Several codes served also as benchmarking software for the prototype hardware deployed in Work Package 11.

Besides the direct contribution of the community software experts, the involvement of the scientific community was addressed by continuous interaction of HPC experts with their scientific counterparts by means of personal contacts, by participation and presentation of the WP8 achievements at conferences and workshops, and by publishing scientific and technical papers in specialised journals. Furthermore, WP8 implemented a specific program to progressively consolidate the engagement of the scientific community, based on periodic Face to Face (F2F) workshops, where scientists, software developers and HPC experts could meet and discuss the WP8 achievements, as well as broader topics related to scientific computing and HPC technologies. More specifically, the workshops had the following targets:

- Summarise, verify and present the work accomplished in WP8;
- Present innovative HPC hardware and software solutions;
- Collect and analyse the requirements of each community;
- Assess, for each scientific domain and code, the working strategy.

Four workshops were organised:

F2F Workshop 1 – Lugano (CSCS), October 18, 2011.

This first workshop had the main objective of initiating the interaction between the HPC and the scientific community members. During the meeting, a number of codes were proposed and their features and the desirable improvements discussed. The methodology to be used for the code selection, the evaluation of the performance and the identification of the algorithms to be subject of refactoring, was assessed and identified with the “Performance Modelling”

approach, which provides a quantitative and objective methodology to evaluate and improve the software.

F2F Workshop 2 – Barcelona (BSC), February 2-3, 2012.

The second workshop was held at the end of Task 1 over two days. The main objective was to introduce and discuss the results of the code performance modelling. During the first day, the outcomes of the performance analysis and the conclusions for the algorithmic kernels refactoring were presented in a plenary session. In the second day, each of the five scientific domains had a dedicated session (which were in parallel), so that the experts in the field providing the necessary feedback to the HPC experts could conduct specific discussions. Furthermore, for each code, all the details of the re-design and refactoring workplan were defined.

F2F Workshop 3 – Paris-Saclay (Maison de la Simulation), June 11-13, 2012.

This workshop represented a first checkpoint for codes development. Furthermore, a *cross-cutting* session, providing an overview on innovative topics in the field of HPC, was organised. It focused on the GPU architecture and the its programming models (CUDA [24], OpenCL [25], OpenACC [26]). As for the Barcelona meeting, the first day was dedicated to the presentation of the status of the work for each code, the preliminary outcomes and the discussion of the coming deadlines and milestones. The morning of the second day was dedicated to the cross-cutting session, while the afternoon and the third day consisted of parallel sessions specific to each scientific domain. These sessions had several key objectives. First, present the details of the accomplished work. Second, collect updates from the scientific communities representatives on their needs and expectations for the next generation of high-end numerical simulations. Finally, discuss how the codes should be re-designed and the work plan adapted in order to try to match such requirements.

F2F Workshop 4 – Lugano (CSCS), March 6-8, 2013.

The last F2F workshop was organised to present the (almost) final outcomes of the code refactoring process and to facilitate their validation by the users' communities by means of a direct interaction with the developers. This was accomplished in the parallel sessions held in the afternoon of the second day and in the third day of the meeting. The first day, as for the previous two F2F meetings was dedicated to the presentation of the status of the work and the discussion of the remaining steps to the end of WP8. The second day started with two cross-cutting sessions. The first session introduced the Intel Xeon Phi [23] processor architecture and programming models. This was followed by the presentation of a few successful experiences in exploiting innovative HPC systems, given by Filippo Spiga (Cambridge University, Material Science), Giovanni Aloisio (representing ENES, Climate) and Bartosz Kostrzewa (Humboldt-Universitaet zu Berlin, Particle Physics). The second cross-cutting theme was represented by the "Big Data problem". This topic, which is of increasing interest in most of the scientific fields, was addressed with talks given by experts in the fields of Astrophysics, with Baerbel Koribalsky (ATNF Sydney, Geophysics), Peter Danecek (INGV Rome Material Science), and Nicola Marzari (IPFL Lausanne), who described successful examples of how the problem of effectively and efficiently managing huge data volumes is being tackled in specific areas. Finally, Giuseppe Fiameni, from CINECA, introduced the EUDAT project, one of the most ambitious FP7 EU funded projects, which addresses the "data deluge" problem. The workshop was closed by a final round table, where feedback, suggestions and remarks were gathered and discussed in order to provide guidelines for the final months of WP8.

Once the code refactoring was completed, Task 3 took over. In principle, Task 3 was expected to re-integrate the implemented kernels into the community codes and to validate them.

However, as described above, this process overlapped with, and blended in with, that of code refactoring. Hence in many cases it did not required a specific action, with details changing among different codes (see the following sections). The residual effort could hence be allocated to further algorithmic tuning and performance optimisation.

3 Astrophysics

This section summarises the achieved results and the re-integration status for three astrophysical codes: RAMSES, an application for cosmological and galaxy evolution simulations, EAF-PAMR, describing the magnetohydrodynamic behaviour of the interstellar medium and PFARM, a suite of programs for the calculation of electrons-atoms interactions in the inter and intra-galactic gas.

3.1 RAMSES

3.1.1 Overview

The RAMSES code was developed to study the evolution of the large-scale structure of the universe and the process of galaxy formation. RAMSES is an adaptive mesh refinement (AMR) multi-species code, describing the behaviour of both the baryonic component, represented as a fluid on the cells of the AMR mesh, and the dark matter, represented as a set of collisionless particles. The two matter components interact via gravitational forces. The AMR approach makes it possible to get high spatial resolution only where this is actually required, thus ensuring a minimal memory usage and computational effort.

The performance modelling procedure applied to RAMSES led to the refactoring of the main kernels, namely Hydro and Gravity, in order to exploit hybrid architectures equipped with multicore CPUs and accelerators. This was done not only to improve the performance, but also for optimal memory usage, avoiding data replication, which limits the maximum size of the simulated mesh, and restricting the usage of MPI message exchanges to internodes communication, allowing efficient local data access instead.

The two kernels were re-implemented with a hybrid OpenMP+MPI approach with specific care given to preserving data locality (even inside the shared memory) and effectively handling concurrent data accesses. The implementation of the GPU version of the hydro kernel was based on OpenACC [26], a novel standard for parallel computing supporting parallel programming of heterogeneous CPU/GPU systems by means of compiler directives. Several solutions have been explored to optimise the performance.

The work on RAMSES was carried out in collaboration between CSCS and the Institute for Theoretical Physics University of Zurich, involving the research group of Prof. Romain Teyssier, the creator and the principal developer of the code.

The main achievements are summarised below:

- Full shared memory OpenMP implementation of the hydro kernel, with good scalability up to 32 cores per CPU;
- Full shared memory OpenMP implementation of the gravity solver. Due to the intrinsic characteristics of the problem (long range forces), leading to an intense memory usage, scalability is limited. However, it represents an effective solution for memory bound problems;
- GPU refactoring of the hydro kernel, based on OpenACC. Two implementations were developed following different approaches. The first led to limited performance improvement, the second is in an advanced stage of development.

3.1.2 Results

3.1.2.1 Hybrid implementation

RAMSES has been re-designed to exploit hybrid multi-node/multi-core architectures, taking advantage, in particular, of the large shared memory available per node, that allows to

decomposition of the computational domain between nodes instead of between cores, thus dealing with larger data chunks, with intra-node direct memory access. The communication overhead is consequently reduced and load balancing is easier to optimise. In order to exploit shared memory, the Hydro and Gravity kernels had to be re-implemented with a hybrid OpenMP+MPI approach. Special care must be devoted to managing the access to shared memory. Therefore, the OpenMP implementation has been developed to preserve data locality even inside the shared memory, and to effectively handle concurrent data accesses.

The main code kernels that have been re-implemented with OpenMP are:

- Hydro kernel
- Conjugate gradient gravitational solver
- Multigrid gravitational solver

The performance obtained depends on the algorithm. The memory access patterns proved to be particularly crucial. In the following sections we present the results obtained using two HPC systems: the Intel based Tier-0 system CURIE, available at CEA (France [21]) and the AMD based system Monte Rosa available at CSCS (Switzerland, [22]).

Hydro kernel

The hydrodynamic solver, based on a MUSCL approach, is purely local, hence memory access is not particularly demanding. 6x6x6 cells domains are built for each cell, collecting at that stage all the information necessary to complete the time update of the cell. Then all the calculation is completely local. This is reflected in very good scalability of the hydro part, comparable to that obtained via MPI, as shown by the strong scaling curves presented in Figure 2 and Figure 3, that shows the results for a pure hydrodynamic test

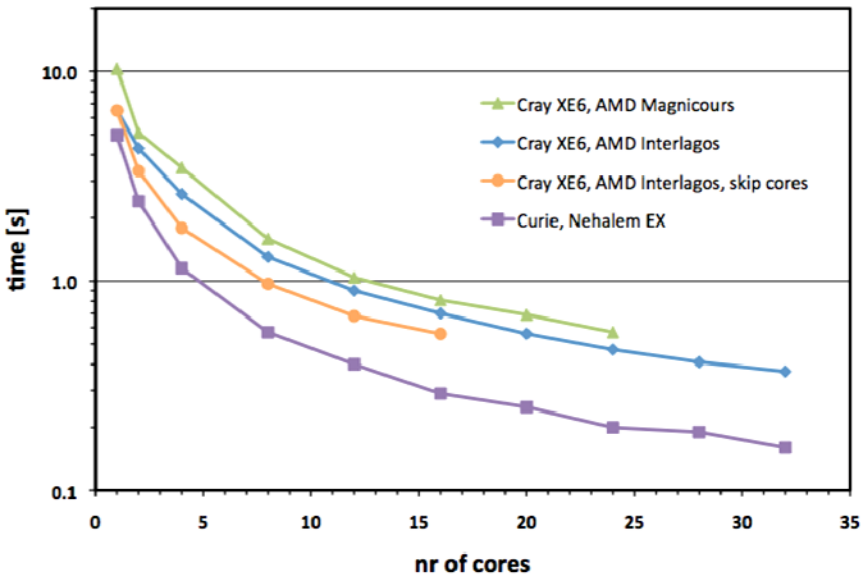

Figure 2: Computing time of the hydro kernel for a 3D "Sedov blast wave" test (no gravity)

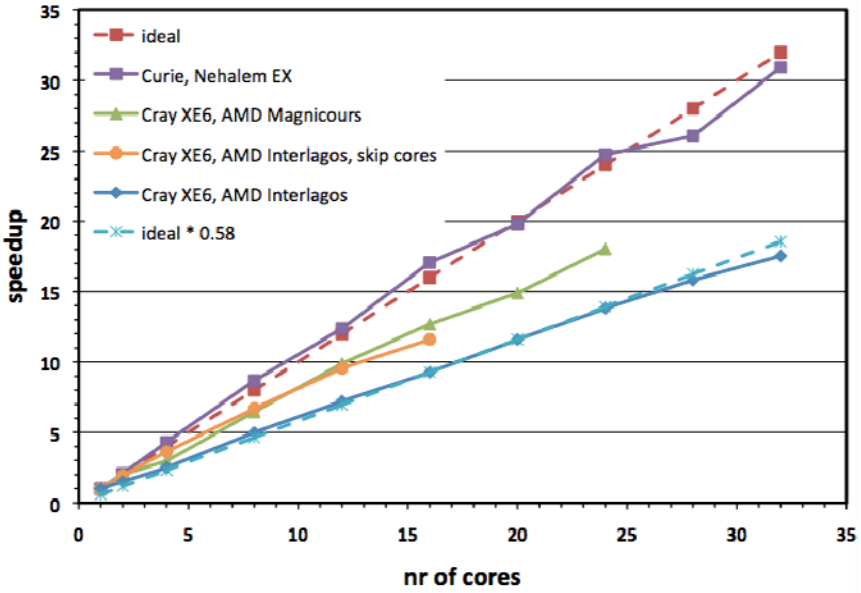

Figure 3: Speed-up of the hydro kernel for a 3D "Sedov blast wave" test (no gravity)

The speed-up is calculated as:

Speedup = t_1 / t_N

where t_1 is the computing time on one core and t_N is the corresponding computing time on N cores. The obtained results show linear scalability for any number of cores, on both systems. The performance, however, depends on the characteristics of the hardware. It is interesting to note that different systems produce speed-up curves with different slopes. This reflects the different architectural features of the various adopted processors.

Conjugate Gradient gravitational solver

Calculation of the gravitational field is accomplished by solving the Poisson equation. Whatever numerical approach is adopted, non-local memory accesses are needed, since gravity is a long-range force. This leads to very unstructured and time-consuming memory usage. On the other hand, the number of operations per accessed byte is small, leading to algorithms whose efficiency is low, especially on a shared memory, multi-threaded system, where concurrent memory accesses are frequent. The shared memory algorithm can be highly optimised on regular computational meshes, with Adaptive Grid Refinement (AMR) switched off, as shown in Figure 4. The obtained scalability is almost linear (even superlinear, thanks to cache effects on the Intel processor) in three of the four cases (the last one being not properly optimised), although the performance on AMD processors is rather poor, due to some known issues of the Interlagos architecture and related OpenMP support.

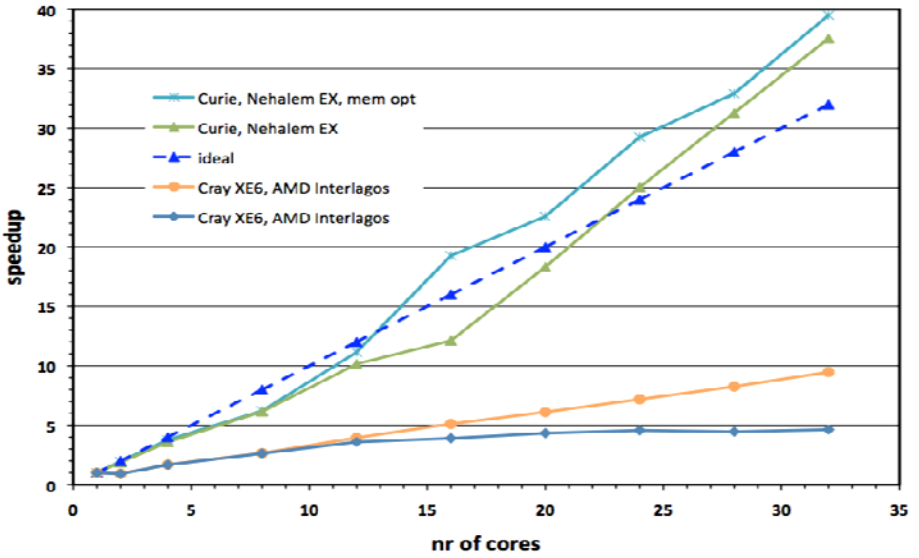

Figure 4: Speed-up of the conjugate gradient Poisson solver.

The linear scalability is more difficult to achieve when AMR is switched on, since memory access patterns become much more complex, and memory usage more difficult to optimise. In Figure 5 we show the results obtained on the Intel processor. Different optimisations have been tested, but an acceptable scalability cannot be achieved above 8 cores. In this case, the optimal approach consists in the usage of 4 MPI tasks per node, each spawning 8 OpenMP threads.

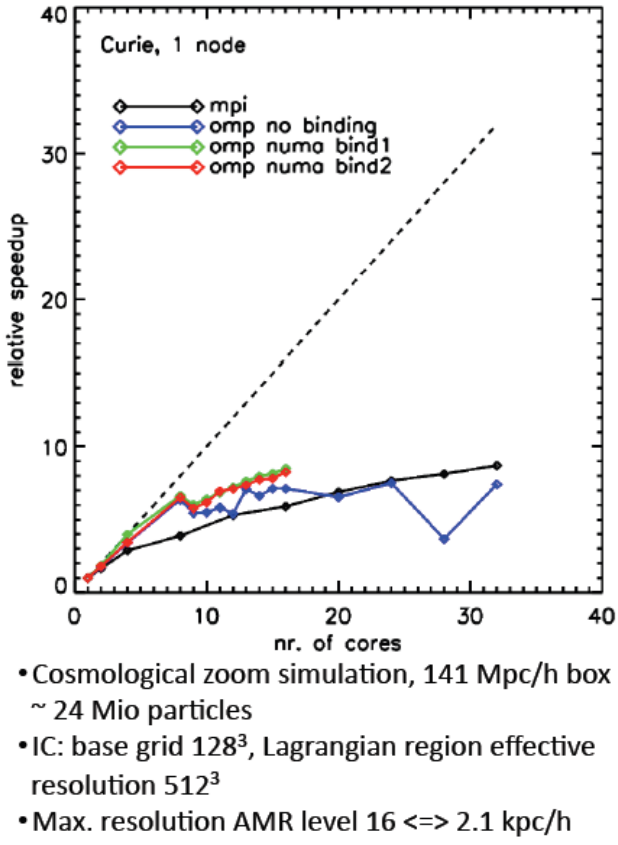

Figure 5: Speed-up of the conjugate gradients gravity solver for a AMR simulation

Multigrid gravitational solver

The multigrid solver shared memory implementation is challenging. The memory access complexity is comparable to that of AMR, and in some situations it can become even more complex. Many routines are computationally lightweight leading to a further loss in efficiency per accessed byte. The obtained speed-up is poor. A more detailed analysis of the behaviour of the various multigrid routines is shown in Figure 6.

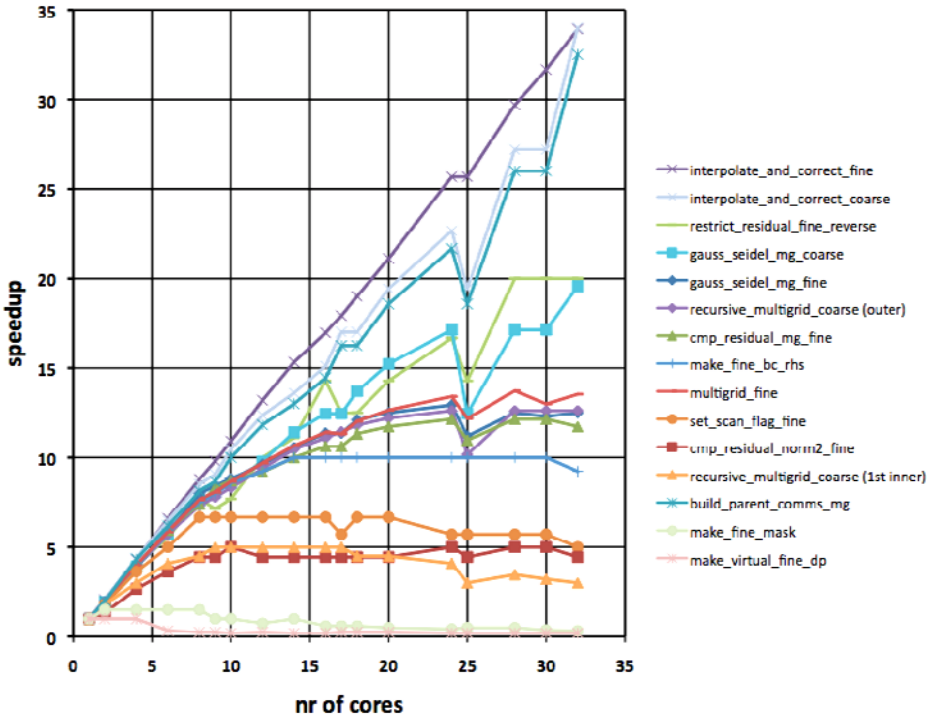

Figure 6: Speed-up of the different components of the multigrid gravity solver. The red line represents the overall speed-up

Linear scalability on the full range of cores cannot be achieved. However, the refactoring of most of the multigrid functions has resulted in acceptable scalability up to 8 cores allowing, also in this case, effective adoption of a solution with 4 MPI processes per node and 8 OpenMP threads per MPI process.

3.1.2.2 Hybrid implementation

The implementation of the GPU version of the hydro kernel was based on OpenACC [26], a standard designed to support parallel programming of heterogeneous CPU/GPU systems by means of compiler directives that specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator.

For the code development and testing the TODI CRAY XK7 system ay CSCS was used. This system is composed of 272 computing nodes, each with a 16-core AMD Opteron CPU, 32 GB DDR3 memory plus one NVIDIA Tesla K20X GPU with 6 GB of GDDR5 memory, for a total of 4352 cores and 272 GPUs.

In order to get good performance from the GPU code, different solutions have been explored.

Approach 1

In a first implementation, the idea was to move all the hydro kernel on the GPU. This idea is sketched in Figure 7, and consists in the following steps:

- Off-load all the necessary data (hydro and gravitational forces variables) to the GPU;
- build the 6x6x6 patches needed to solve each single cell on the GPU;

- solve the hydro problem for the cell using the patch data on a different GPU streaming processor;
- collect the updated hydro values in the result array;
- as soon as all cells are solved, move the result array back to CPU for the next steps.

The algorithm was implemented and optimised adopting different configurations of OpenACC loops. Typical results obtained with the best possible tuning for the classical "Sedov Blast Wave" test are shown in Table 2.

ACC	NVECTOR	Ttot	Tacc	Ttransf	Eff. Speed-up
OFF - 4 Pes	10	25.71			
OFF - 1 Pe	10	94.54	0	0	
ON	512	55.83	38.22	9.2	2.01
ON	1024	45.66	29.27	9.2	2.66
ON	2048	42.08	25.36	9.2	3.06
ON	4096	41.32	23.2	9.2	3.29
ON	8192	41.19	23.15	9.2	3.30

Table 2: Results for the GPU implementation of the hydro kernel with approach 1.

where NVECTOR represents an internal Ramses tuning parameter. The speed-up, defined as the time-to-solution for the hydro kernel on the GPU compared to that of a core, is limited to between a factor of 2 and 3.3.

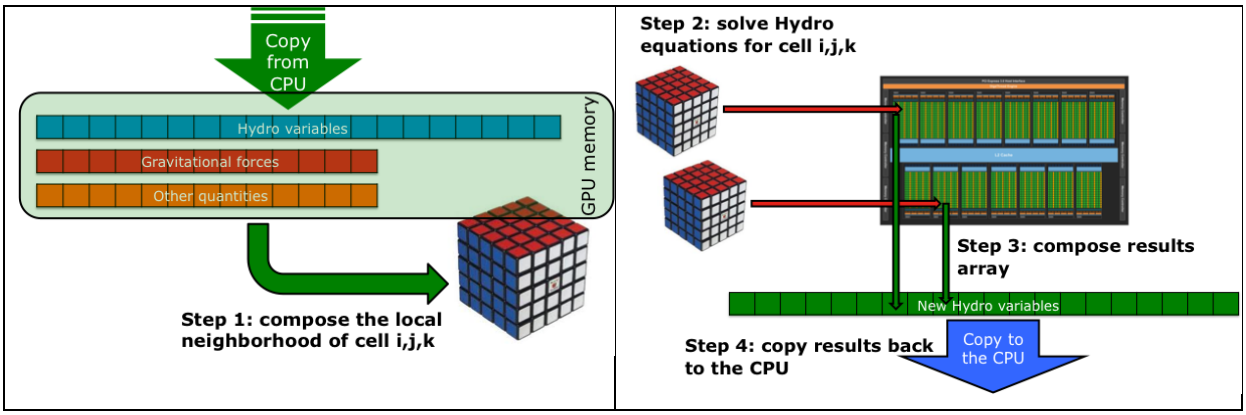

Figure 7: GPU implementation: approach 1

We identified a number of factors limiting the performance achieved with this first approach:

- Complex data structure, with irregular data distribution, which leads to an intensive usage of the GPU global memory with non-optimal patterns that prevent any kind of coalescence;
- Amount of transferred data from CPU to GPU and vice-versa;
- Low flops per byte ratio;
- Asynchronous operations not permitted, due to the way the algorithm is implemented: this prevents any kind of overlap between data transfer and GPU work.

Approach 2

In order to circumvent part of the performance limiting factors found with the first approach, we have adopted a different strategy. With this approach memory is reorganised in order to be accessed more efficiently and data arrays are properly split in order to support the overlap of data transfer (between CPU and GPU) and work. This solution is sketched in Figure 8.

In this case, the algorithm consists in the following steps:

- compose large, regular, rectangular patches of contiguous cells at the same refinement level on the CPU;
- off-load the patch and integrate it on the GPU;
- at the same time, using asynchronous operations, composes the next patch on the CPU and start copying it to the GPU;
- once the patch is updated, move it back to the CPU and update the original data structure.

With this solution, a much more effective usage of the GPU memory is guaranteed. At the same time, the off-load overhead is hidden thanks to the usage of asynchronous operations and the overlap with the work performed by the GPU cores. Furthermore, the large-patches re-composition overhead should finally be hidden. This is important since the patch-re-composition stage is not present in the original algorithm, so it represents an overhead due to the usage of the GPU.

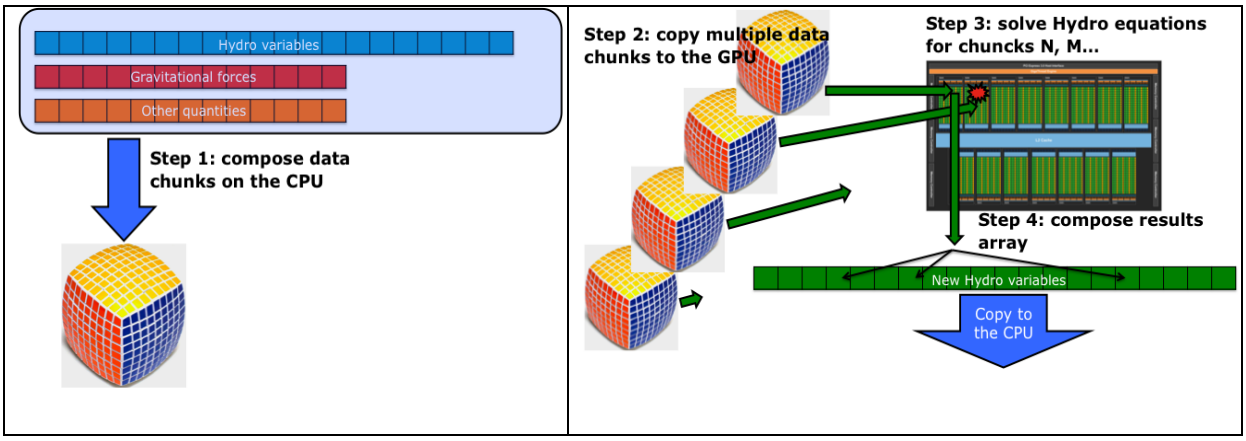

Figure 8: GPU implementation: approach 2

The main drawback for this kind of approach is its complexity, especially due to the partial support to asynchronous operations provided by OpenACC in the current release (version 2.0). This has slowed down the implementation process, which is in its debugging phase. Hence at the moment, no tests and benchmarks can be presented.

3.1.3 Re-introduction status

Thanks to the involvement of the University of Zurich, the way RAMSES was re-designed and re-implemented in WP8 was discussed, planned and engineered step-by-step between the CSCS experts and the main code developers, who also progressively verified the results of each implementation stage. In this way, the kernels implemented in the work package became part of the standard development process, without requiring special acceptance procedures. The source code can be downloaded from the RAMSES SVN repository at https://svn.physik.uzh.ch/repos/itp/ramses (see Deliverable 8.2 [6] for further details).

The work on RAMSES was introduced to the community by means of:

- WP8 F2Fs, where the technical progress and achievements were progressively described both to the broad interdisciplinary audience and, in specific parallel sessions, to the experts in the field of Computational Astrophysics.
- Papers and presentations at conferences and workshops (for instance the poster [13] at "GPU Technical Conference 2013" [12], and the presentation "Cosmology on the GPU" REF at the 3rd HydroSim Workshop in Trieste – January 10-11, 2013).

- Presentations at RAMSES users community workshops (the most recent held in Zurich on June 10-14, 2013 [14][15]).
- The project web site at: http://hpcforge.org/plugins/mediawiki/wiki/ramses/index.php/Main_Page, collecting all the details of the work and the relevant documents and links.

3.2 EAF-PAMR

3.2.1 Overview

The EAF-PAMR OCL-MODULE is a three-dimensional hydrodynamic simulation code specially tailored for astrophysical applications. It is a shock capturing grid based code that uses the 3rd order interpolation algorithm Piecewise Parabolic Method (PPM [64]) to solve the Riemann problem between neighbouring cells when solving the Euler equations. The PPM drastically increases the accuracy of the simulations compared with other popular algorithms, especially in problems where the evolution of shocks is very important to the dynamics of the problem, e.g., supernova driven shocks and the dynamics of the interstellar medium in star-forming galaxies.

The main goal proposed for the work done during the WP8 was the adaptation and refactoring of the EAF-PAMR code adopting the OpenCL paradigm, to enable the user to use CPUs, GPUs and also, thanks to the OpenCL support to heterogeneous architectures, other kinds of accelerators such as the new Intel Xeon Phi. However, once the refactoring work began and the intricacies of the chosen platform became clear, it became obvious that it would be much more productive and would lead to a better final result if, instead of refactoring an existing code, an entirely new program was created. In that sense the major goals were adjusted to:

1. Implementation of two first order methods (Roe and HLL solver) in one dimension.
2. Augmentation of the methods above using the PPM method.
3. Adaptation of the above algorithms to three dimensions using dimensional splitting.
4. Preparation of the program to work in several independent computational devices simultaneously by doing the division of the computational domain.
5. Extending the code to solve magnetohydrodynamical (MHD) problems.
6. Implementation of self-gravity calculation. Such work would imply implementation of a numerical solver for the Poisson equation in OpenCL.

From the above goal we can state that the following items were successfully implemented:

- Three dimensional PPM method using either a Roe or HLL solver (Goals 1-3 in above list).
- Domain decomposition (Goal 4 in above list).
- Implementation of a TVD MHD method (Goal 5 in above list).

3.2.2 Results

As noted above most of the goals were accomplished during the WP8. The main portion of that time was dedicated to the implementation of the PPM methods in OpenCL. This was due to both the complexity of the algorithm in itself, as well as the difficulties associated with using a complex platform such as OpenCL where most of the methods and paradigms usually employed when developing software such as this either do not work or are very inefficient. Thus, many times there was the need to implement new approaches. The following topics address the main stages of the work done when developing this code.

Implementation of a 3D PPM algorithm

In order to adapt a PPM algorithm that was efficient when using OpenCL several factors were weighed up. First the problem had to be expressed in a way that could take advantage of the structure of modern GPUs that consist of thousands of processors. This meant dividing the problem in a way such that all of those processors were busy and had enough work assigned to them so that the overhead times were negligible when compared to the calculation times. Another important aspect is memory management, in terms of reads/writes in each individual thread, in order to avoid bottlenecks where two or more processors try to access the same memory locations at the same time, or even worse, where the values in each cell are altered before each processor has time to read the value it requires.

Luckily one of the most commonly used methods to create multi-dimensional codes, dimensional splitting, managed to address both of these potential issues. By using dimensional splitting the problem is effectively reduced to a series of one-dimensional problems, with each one of those problems assigned to one processor. In this way, as each processor only calculates one "row" of the problem, and because each of those "rows" is independent of all the others, there is no risk of memory collisions or overwrites and, also with this method, each processor is assigned a large amount of work.

Memory structure

One important aspect of optimizing a HPC code is the way the memory is structured and accessed. When using GPUs that concern is especially important since the time required to transfer the data between the host computer and the device can be an important bottleneck.

In order to minimize that problem the information is uploaded to the GPU at the beginning of program execution, and, since all the calculations are done on the GPU, there is no further need to transfer that information back and forth to the host computer, except when output to physical media is required. In the case that modifications must be made to the data (e.g., insertion of an energy outburst in order to simulate a supernova explosion) only the relevant cell (or cell interval) needs to be downloaded to the host, altered and re-uploaded to the GPU. This same logic applies to the implementation of the boundary conditions, where only the relevant data intervals are downloaded from the GPU, modified (if necessary) and uploaded to the GPU in the appropriate place.

In terms of how the memory is structured, the chosen method was to pack all relevant variables in a structure, so that all the relevant information about each cell is in adjacent memory locations and the fetching process when running the program is substantially accelerated. In Table 3, we show the total run times of the parallel section of the code (in short the OpenCL run time) for the case where the variables are packed in a structure (aligned memory), and the case where each variable is allocated independently (memory not aligned).

	1 CPU core	4 CPU cores	GPU
Aligned memory	170.71 s	26.45 s	1.47 s
Memory not aligned	99.66 s	45.43 s	2.31 s
Ratio	1.71	1.71	1.57

Table 3: Total runtime of the parallel section of the code (in seconds) when the variables are aligned in memory and when they are not. The bottom row gives the ratio of the two results. (GPU: AMD HD7970 with 2040 stream processors; CPU: Intel core2 Q6600 2.4GHZ)

There is a substantial reduction of the run time when the memory is structured so that the fetching time is minimized. Surprisingly this effect is smaller when considering GPUs, but this is due to the fact that most of these devices use DDR5 memory that is substantially faster than the typical RAM memory in traditional computers.

Host code optimization and domain decomposition

Once a stable PPM code was obtained the next step was to enable the capability to use the code with several computational devices simultaneously. This is a necessity since most simulations with scientific relevance can easily need more than 30 GB of memory and most GPUs have, at best, 6 to 8 GB of memory available. To that end, a method to divide the domain was implemented so that each computational unit receives only the information for a portion of the total problem and calculates only the evolution of that same portion.

This led to a new problem, where this division created a new bottleneck in the host code (the portion of the code that sets up and controls the computational units) and suggests that additional use of another parallelization tool, such as MPI or OpenMP will be needed to complement the OpenCL implementation. In Figure 9, one can see that when using two GPUs to run the code we obtain only an increase of 35% relative to the same problem on only one GPU. A very simple implementation of OpenMP directives improves that value and brings it up to 63%. Although this is a very good initial result it makes it obvious that it requires more work in order to optimize this section of the program.

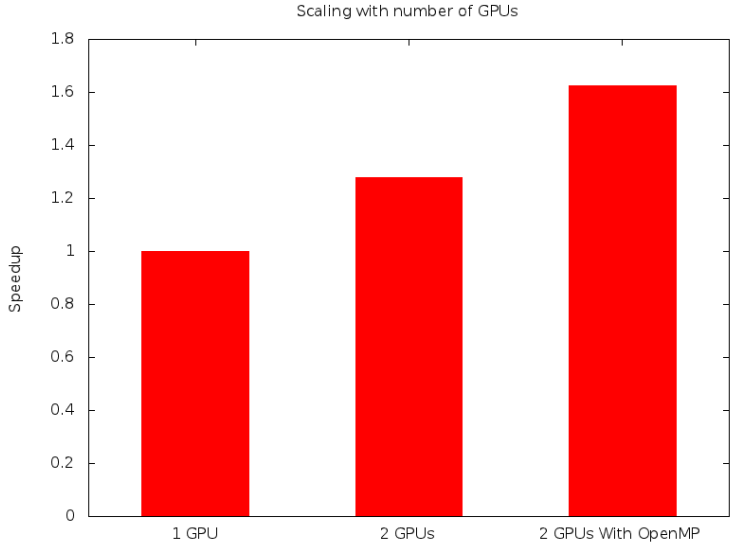

Figure 9: Speedup obtained when using two GPUs with and without host code parallelization with OpenMP.

Performance results

Although further testing is under way, preliminary results of the effectiveness of the HD PPM code can be seen in Figure 10.

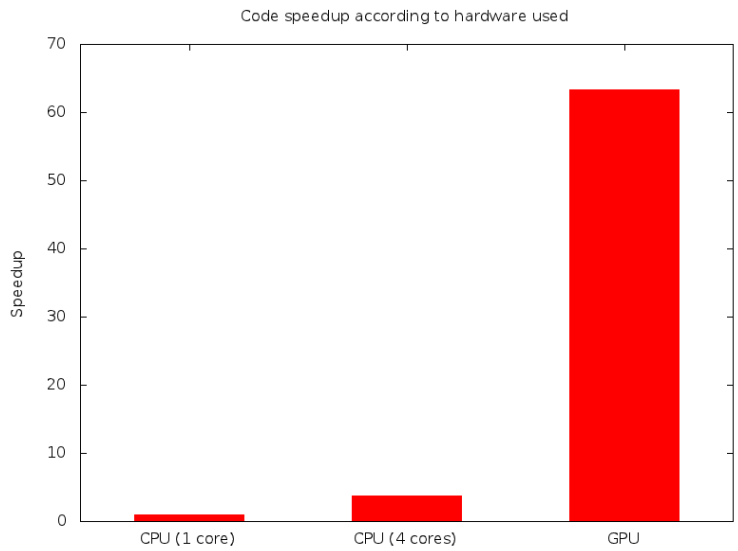

Figure 10: Speedup obtained when comparing the parallel code execution time obtained when using a GPU and a 4 core CPU with the time a single core of that same CPU took to run the program. (GPU: AMD HD7970 (2040 stream processors), CPU: Intel core2 Q6600 2.4GHZ).

When comparing the execution times of the parallel section of the code when using a quad-core CPU or a GPU with that same value obtained when using only 1 core of the CPU, we can see that we can obtain a speedup of about 63 times when using the GPU. Also of relevance is that the acceleration obtained when using all the four cores of the CPU the parallel code runs about 3.7 times faster, a value that is very close to the theoretical maximum of 4.

3.2.3 Re-introduction status

The EAF-PAMR OCL-MODULE can be used in two ways: (i) it can run as an independent hydrodynamical code or (ii) it can be incorporated into any general-purpose gas-dynamics software. In the latter case appropriate calls have to be made to the OCL-MODULE that in turn takes charge of all the time-dependent hydrodynamical calculation and the CPU-GPUs accesses and usage. Such use of the OCL-MODULE will allow the reduction of the computing time needed in production runs of gas-dynamical problems (not only in Astrophysics, but also in other fields). This in turn will allow for an increase of resolution without the need to use adaptive mesh refinement. The increase in resolution has several consequences, for example a) better convergence of solutions, b) resolving the cooling scales (it is known that a increase of resolution affects the cooling of the gas), c) better describing the dissipation scales of compressible turbulence, and therefore the fractal dimension of the most dissipative structures and their link to the density of the medium, d) in the case of structure formation, allowing the use of a gas-dynamical model to describe galaxies with better resolutions, just to name a few. As can be seen, the impact in science is quite large if the OCL-MODULE is used by researchers. While still under testing, the OCL-MODULE has already been used to carry out large-scale simulations of the interstellar medium by the Computational Astrophysics Group and its solutions have been compared to previous published results. After several improvements in the code and completion of the tests, the OCL-MODULE will be made public on the Community software pages of WP8.

3.3 PFARM

3.3.1 Overview

PFARM is part of a suite of programs based on the 'R-matrix' ab-initio approach to the variational solution of the many-electron Schrödinger equation for electron-atom and electron-ion scattering [28]. Relativistic extensions have been developed and have enabled much accurate scattering data to be produced. The package has been used to calculate electron collision data for astrophysical applications (such as the interstellar medium, and planetary atmospheres) with, for example, various ions of Fe and Ni and neutral O, plus other applications such as plasma modelling and fusion reactor impurities. PFARM performs the 'outer region' calculation extending from the R-matrix sphere boundary to the asymptotic region in which scattering matrices and (temperature-dependent) collision strengths are produced [28]. The code has recently been adapted to form a compatible interface with the UKRmol suite of codes for electron (positron) molecule collisions [29], thus enabling large scale parallel outer-region calculations for molecular systems as well as atomic systems.

PFARM divides outer region configuration space into radial sectors and solves for the Green's function within each sector using a basis expansion: the BBM method [30]. The parallel calculation takes place in two distinct stages, with a dedicated MPI-based program for each stage. Firstly, a domain decomposition approach is used in EXDIG, where parallel sector Hamiltonian matrix eigensolutions are performed. The energy-dependent propagation (EXAS stage) across the sectors is then performed using systolic pipelines with different processors computing different sector calculations. Each of the main stages of the calculation is designed to take advantage of highly optimised numerical library routines. Hybrid MPI / OpenMP parallelisation has also been introduced into the code via shared memory enabled numerical library kernels [31]

The main goal of the work in WP8 was to enable very large-scale electron-atom and electron-molecule collisions calculations by exploiting the latest high-end computing architectures. Performance analyses of the code in the early stages of this project identified three main bottlenecks that needed to be addressed in order to prepare PFARM for the next generation of HPC architectures: the limited scalability of parallel eigensolver methods in EXDIG, the sequential input of data to the process pipelines in EXAS, and ineffective load-balancing of the EXAS code on petaflop architectures.

Following on from this analysis, four main objectives were identified for enabling work:

- Analysis and subsequent utilisation of the latest numerical library routines (EXDIG and EXAS).
- Investigation of potential use of computational accelerators (EXAS).
- Efficient (parallel) I/O (EXDIG and EXAS)
- Performance tuning and efficient load-balancing of EXAS on new architectures

3.3.2 Results

The enabling work accomplished in the project is summarised in the listing below.

1. Utilisation of the latest parallel eigensolvers for PFARM (EXDIG). These new solvers have recently been made available as part of the ELPA (Eigenvalue SoLvers for Petaflop-Applications [32]) project through a collaboration of several German centres and IBM. The other location in the code where new numerical library routines [31] have been introduced into the code to improve performance has been for the Singular Value Decomposition factorisation that takes place as part of the K-matrix calculation in EXAS. In order to improve further the parallel performance of EXDIG, ELPA has

also been used to calculate each Hamiltonian sector parallel eigensolve concurrently on MPI sub-groups of tasks. The MPI sub-groups need to be of sufficient size to accommodate the Hamiltonian sector matrix plus associated overheads but small enough to maintain an advantageous communications/computation ratio. This strategy avoids distributing computational loads too thinly when problem sizes are relatively small and parallel jobs involve many thousands of cores. Figure 11 summarises the performance improvements on the IBM Blue Gene/Q for EXDIG associated with this approach.

2. Parallelisation of the input routines for surface amplitude data to the multiple process pipelines in EXAS. This approach significantly reduced the set-up time for the propagation stage. Parallel output has also been introduced to EXDIG as an integral feature of the MPI sub-group eigensolver approach.
3. New algorithms for load-balancing the EXAS stage of the code to reflect computation/communication ratios on the latest HPC architectures, such as the IBM Blue Gene/Q.
4. Performance analyses for the core computational kernels in EXAS from new computational accelerator technologies - NVIDIA GPU and the Intel Xeon Phi. Optimisation strategies such as auto-offloading and native execution using new routines from CuBLAS [33], Intel MKL [34] and MAGMA [35] have been investigated. A near complete port of the EXAS code to GPUs has also been achieved.

A detailed report of the software enabling effort, including many more sets of results can be found on the HPCforge PFARM [36].

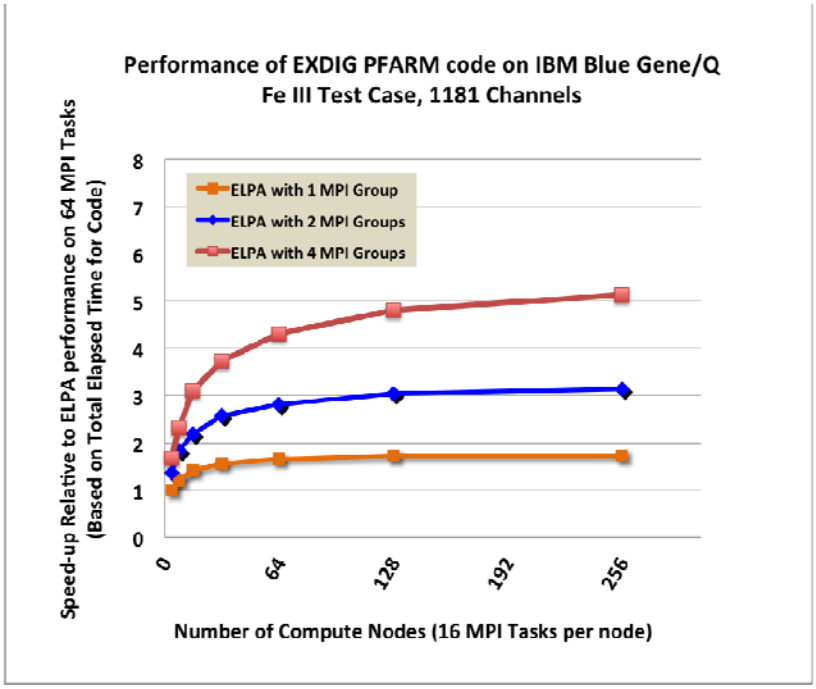

Figure 11. Parallel performance impact of using different MPI sub-groups for the ELPA-based sector eigensolves on the IBM Blue Gene/Q

The cumulative effect on parallel performance of all the software enabling work, undertaken to-date in PRACE WP8 is shown in Figure 12

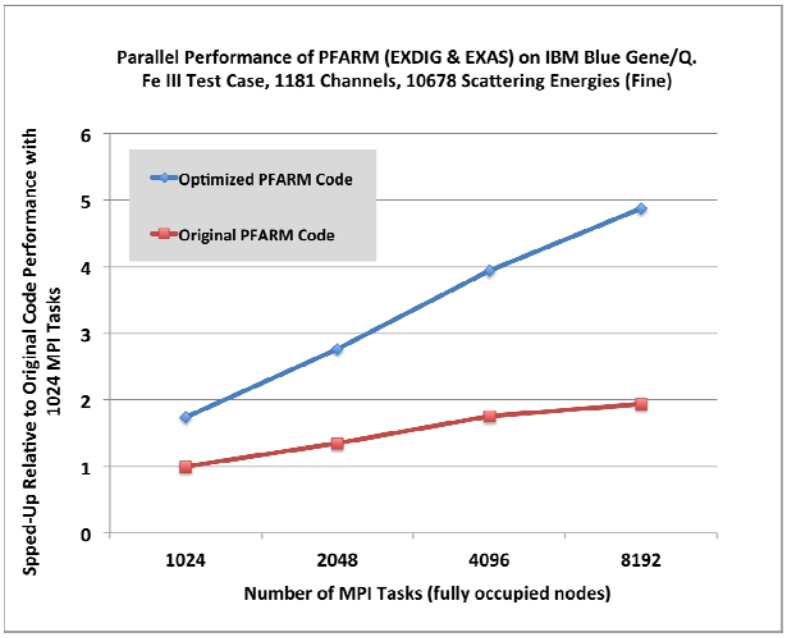

Figure 12. Overall impact of the software enabling work on the parallel performance of PFARM.

3.3.3 Re-introduction status

The PFARM codes are registered as projects on CCPForge [37], a collaborative software development environment tool with overall administration and assistance by the software engineering group (SEG) of the UK Science and Technology Facilities Council [38] CCPForge provides an online repository for the codes, allowing version control via Subversion [39], user and developer forums and mailing lists, bug reporting, tracking tools and more. The PRMAT (atomic inner region codes and PFARM for the outer region) suite can be downloaded from CCPForge in both development versions and as tested 'releases' (releases are currently unofficial but this will change), with associated test suites of example calculations being added to the site. The development of official test suites has enabled testing of ported code, verification of new code and validation of backwards compatibility.

Two examples of research groups that use PFARM include (i) the electron (positron) molecule collisions consortium UKRmol (University College London (UCL), The Open University, University of Edinburgh and various international collaborations) and (ii) the Atoms for Astrophysics group at Queen's University Belfast (QUB). Details of these collaborations, including details of the latest specific cases being investigated with PFARM follow below:

1. PFARM is fully compatible with the molecular R-matrix package UKRmol [29] and is currently being used for large-scale calculations of electron-methane scattering at UCL. These are the first accurate calculations of these cross sections and will allow more detailed studies of the properties of this system. Methane is common in the atmospheres of extrasolar planets: once known, the collision data can be subtracted from line lists, allowing other species to be more easily detected [40]. It is also of importance in many other hot astronomical environments, particularly brown dwarfs and cool stars. Various hydrocarbon simulations also involve methane, for example to model the human body in medical simulations. A separate project is adapting the UKRmol resonance-finding code to incorporate PFARM parallelism.
2. PFARM is being used at the Centre for Theoretical, Atomic and Molecular Physics (CTAMOP) at QUB [41]in an ambitious set of calculations of intermediate energy e-Fe+ collisions producing data for astrophysical spectra. A

particular on-going calculation involving relativistic coupling and partial waves with quantum numbers 2J = 6 or higher (J 'pi' recoupled from LS 'pi' with relativistic target states), involving several thousands of channels and tens of thousands of scattering energies, is only possible with the new PFARM code owing to the demands on parallelisation and memory. The CTAMOP staff will be providing feedback evaluation of the PRACE developments. The new PFARM is also to be used in new ambitious calculations of electron collisions with ionised tungsten relevant to JET / ITER fusion reactor research [42].

4 Climate

This section is dedicated to summarise the work accomplished in the Climate domain: OASIS, a software interface between different climate models, ICON, a general circulation model, NEMO, a modelling framework for oceanographic research, and Fluidity-ICOM, a three-dimensional non-hydrostatic parallel ocean model. Furthermore, the work on a common high performance "I/O services" module is presented. The re-integration status of the different codes is also presented.

4.1 Couplers: OASIS

4.1.1 Overview

In Europe, OASIS is the most widely employed coupler to set-up a full climate model on the basis on individual realm components (ocean, atmosphere, sea-ice, etc.). It is used in six of the seven Earth System Models (ESMs) involved in ENES to exchange and interpolate the coupling information (the coupling "fields") between their individual components. The widely used OASIS-3 version of the coupler offers only a limited field-by-field "pseudo-parallelism", and has become a central bottleneck for high resolution ESMs running on massively parallel platforms. Initial work in documented in D8.1 had shown bottlenecks in scaling beyond 256 MPI processes (of the IFS atmosphere component of EC-Earth) in OASIS3 on the Stokes cluster at ICHEC.

New climate models under development are now using unstructured grids, which OASIS3 does not support. A new version of OASIS, OASIS4 was designed to handle these, and to be properly parallel, but was seen to suffer conservation problems. Hence it was decided to investigate a new development on OASIS3, OASIS3-MCT, and to investigate other solutions to conservative interpolation in unstructured grids.

4.1.2 Results

- A new version of OASIS, OASIS3-MCT, was examined and tested against OASIS4 with pre-computed weights. OASIS3-MCT was seen to scale better and be a more reliable codebase; this was then tested and optimised for use, with EC-Earth used as a test model.
- Scaling at ICHEC discovered bottlenecks during initialisation at 120 nodes locally on Stokes, due to file handling. These have subsequently been removed by code development at CERFACS. A further bottleneck at 1000 MPI processes due to array distribution was discovered and removed (Released in July 2013), and the code now scales to 3000 MPI processes at ICHEC. On T799 resolutions, speedups were flat up to 3000 MPI processes (without OpenMP) on dual processor Westmere, though setoasis() execution time rose from 2 to 7%. It was discovered that the optimal ratio of IFS to NEMO processes moved from 3:1 to 2:1 as the process count exceeded 2000 processes: doing so reduced time spent in getoasis() to below 20%. Increasing the ratio of IFS MPI processes is likely to be necessary at higher core counts. This needs to be investigated, as it will likely be the bottleneck on larger core counts: this is under further investigation as part of extended work at ICHEC.
- EC-Earth v3.1 with OASIS3-MCT was ported to Hermit for testing on low-memory nodes as this was seen to be a bottleneck in D8.1; scaling work here is being completed at ICHEC. Testing will include hybrid mode (MPI+OpenMP) at high core count.
- Work on developing the "next generation" coupler, targeting models with icosahedral or unstructured grids (e.g. ICON, the Fluidity-ICOM ocean model, etc.) was undertaken by Joel Chavas of CEA/GENCI and "La Maison de la Simulation". For

this Open-PALM (developed at CERFACS and ONERA) was chosen, and work was undertaken to implement the necessary conservative interpolation in this coupler. Joel Chavas has produced a new version of the SCRIP library used in OASIS and Open-PALM couplers to implement conservative regridding when grid edges are unknown, as is needed for Open-PALM [65].

4.1.3 Re-introduction Status

OASIS3-MCT bugs and fixes were submitted back to CERFACS during the development of OASIS3-MCT v2.0 and are present in the July 2013 release. OASIS3-MCT changes to EC-Earth v3.1 are on the http://dev.ec-earth.org development portal. The current development branch of EC-Earth contains the OASIS3-MCT changes. Currently this code is available to EC-Earth consortium members only, but when the IFS code is made available via OpenIFS, it is hoped that the code will become open-source.

The conservative interpolation of Joel Chavas was included in the modified SCRIP library in the August 2012 release of OASIS3-MCT, which is available at http://www.cerfacs.org.

4.2 Input/Output: CDI, XIOS

4.2.1 Overview

As described in D8.1.4, a common "I/O services" module is being developed within the ENES community, for use by all climate models. As part of this, components of the proposed "I/O services" module were designed and implemented in PRACE-2IP WP8. In D8.1.4 it was agreed that:

- I/O services will implement a writer service, reading data from the model nodes (those nodes running the climate model itself) via RDMA / single-sided communications. This enables the model to continue while I/O services handles the parallel write (read is not an issue for global models).
- The I/O services implement parallel writes to a number of potential formats, principally netCDF [66] and GRIB [69], using the CDI library and XIOS libraries.
- The I/O services are implemented using separate I/O nodes, to enable buffering of I/O in memory. This balances the large transient communications internal to the compute cluster (e.g. 250 GB/s) to the typically smaller but sustained I/O bandwidth; then I/O scaling becomes a matter of adding additional nodes for I/O.
- Post-processing is then handled on the fly within the I/O services, based on the XIOS model from IPSL. This work is done in collaboration with IPSL (XIOS developers), MPI-M (CDI developers), in parallel to the G8 "ICOMEX" dycore initiative and IS-ENES efforts.
- To enable the compute nodes to proceed while I/O services are writing and doing post-processing, it is, in some configurations, necessary to add buffer ("memory proxy") nodes. It was decided to extend the XIOS code to add such buffer nodes.

Within PRACE, ICHEC and CSCS implemented an initial template version of the common I/O services based on the ScaLES CDI, in comparison to the existing PIO developed at NCAR. This implements the API, in which the post-processing services will be implemented in parallel by ENES partners (IPSL). CDI interfaces to both the XIOS and the IFS atmosphere model for EC-Earth (which has its own native I/O) was required to implement the I/O services completely.

4.2.2 Results

A new CDI interface has been implemented for IFS, within the EC-EARTH framework. This has been tested using GRIB format and version 1.6.0 of the libCDI interface, and is on a development branch in the EC-Earth development portal. Testing under the parallel version with PIO_FPGUARD mode of libCDI is starting under the project extension at ICHEC. I/O writes on the CDI interface occur at bandwidth-limited rates from the nodes in serial mode; the results have been tested for correctness vs. the native GRIB mode in IFS.

The XIOS ("XML I/O Server") from IPSL has been extended to add "memory cache" nodes. These enable the offloading of model data from compute nodes at large transient communication rates (e.g. 250 GB/s over Infiniband, seen under the ScaLES project within IS-ENES) to buffer writing by the I/O nodes at smaller but sustained I/O bandwidths (e.g. 30 GB/s). This is essential to enable the I/O services module to work on memory-constrained systems, e.g. BlueGene or Cray-based systems. These "memory cache" nodes operate transparently, enabling the "XIOS" (XML I/O Server) subsystem to scale limited by I/O bandwidth rather than node memory. In current configurations, a fixed number of "memory cache" nodes are enabled, typically a multiple (2× or 3×) of the number of XIOS "I/O" nodes (this number depending on the degree of post-processing work XIOS undertakes). The memory caches then work transparently, requiring just a parameter set in the XIOS configuration file.

The memory cache nodes work at Infiniband-limited rates on the Stokes cluster, but need testing on large core counts, and the placement of memory cache nodes has not been investigated yet. This is the subject of extended work at ICHEC.

Work on adding a CDI-XIOS mapping layer, to enable this "memory cache" system to be used with CDI was postponed due to staffing shortages, and the work of testing the XIOS changes was then tested using the XIOS test suite and NEMO ocean model. This work is being done currently in a six-month extension of the project at ICHEC.

The CDI-XIOS layer is now being implemented in extended work at ICHEC. Within IS-ENES, work continues on the "PIO" version of the CDI writer library. At CEA, interpolation has been added to the post-processing layer within XIOS.

4.2.3 Re-Introduction Status

XIOS changes are available at the XIOS Subversion repository http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS; IFS changes for CDI at the EC-Earth Subversion repository https://dev.ec-earth.org.

Changes are continuing as the IFS code and XIOS codes are tested. A workshop is planned with the IS-ENES community for October 2013 on merging the components of the I/O services and interpolation systems used. The CDI-XIOS interface is planned to be complete by this time and all the components available for merging into a common I/O services library. It is expected that the PIO_FPGUARD mode of the CDI library, currently on development branch at the DKRZ subversion repository will be released by October, enabling the merge and full-scale testing of the CDI interface within the planned extension of the project at ICHEC.

4.3 ICON

4.3.1 Overview

The Icosahedral Non-Hydrostatic (ICON) model is a climate model jointly developed by the German Weather Service (DWD) and the Max Planck Institute for Meteorology (MPI-M). It is intended to replace the popular ECHAM model in the next several years. Broadly speaking, ICON consists of a non-hydrostatic dynamical core (NHDC), which calculates the motion of the atmosphere, and physical parameterisations ("Physics"), which calculate the influence of phenomena occurring on a sub-grid scale. ICON employs an icosahedral mesh (and its hexagonal-pentagonal dual mesh) as well as static grid refinement to enhance resolution in geographical areas of interest.

The projects goals were to

- Implement several single-node GPU prototypes of the NHDC to determine which programming paradigms were most promising for the project.
- Accurately model the performance of a new multi-node distributed memory implementation.
- Implement a multi-node distributed memory version of the code based on the ICON Domain Specific Language (DSL) testbed.
- If possible, to integrate the physical parameterisations which were enabled for GPUs within the COSMO (Consortium for Small-scale Modelling) model.
- To propagate the achievements of this task in the wider climate and computational science communities.

The main achievements were,

- Two prototypes – one with OpenCL and one with CUDAFortran – were implemented, both with appreciable speedups on GPUs with respect to current CPU technology
- A multi-node distributed memory GPU-enabled version of the ICON DSL was implemented using the OpenACC industry standard for accelerator directives. The resulting performance – roughly 2 times speedup on the GPUs compared to equal numbers of dual Intel Sandybridge nodes for real execution scenarios – appears to validate the OpenACC strategy.
- Results of the both the prototype and multi-node implementation were presented at several workshops and conferences.

4.3.2 Results

The results of the ICON task were documented on the PRACE WP8 site at http://prace2ip-wp8.hpcforge.org, whose contents are described in detail in WP8 deliverable D8.2. The performance of the two single-node prototypes is depicted in Figure 13 for three different resolutions ("R2B3", "R2B4", and "R2B5") each one requiring roughly four times as much computation as the previous one.

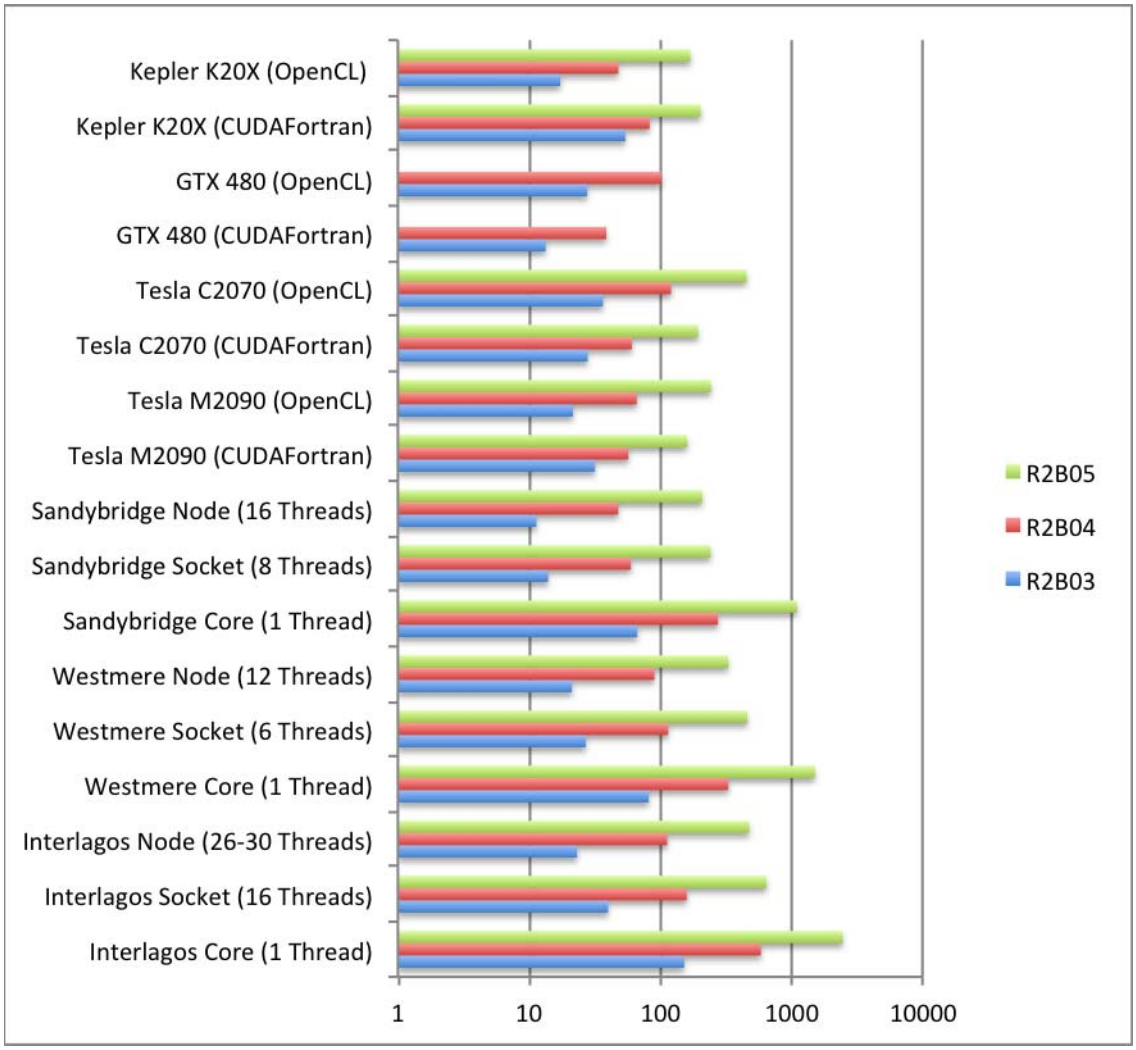

Figure 13: Execution time in seconds of the two single-node prototypes for three different resolutions "R2B3", "R2B4", and "R2B5". Each successive resolution increases computation by roughly 4x.

While the performance results of the single-node prototypes can be considered a success, feedback from the community indicated limited interest in the programming paradigms. It was felt that OpenCL, although a standard, and the required C wrappers, representing too large a change from the existing Fortran programming style. CUDAFortran, on the other hand, was seen as a more natural extension to Fortran, but it would bind developers to one vendor (Portland Group). In view of this, both paradigms were rejected for integration into the community development, and the relatively new OpenACC standard for accelerator directives was chosen for the multi-node implementation.

The multi-node implementation presented new challenges, in particular the relatively new OpenACC standard and its vendor implementations, as well as the challenge of successfully porting the communication (halo-exchange) to the GPUs. Both of these were addressed, and the resulting performance for several resolutions (Figure 14) indicates a speedup of roughly 2 times for the configurations of interest, where the problem size first fits into system memory (left-hand) side of graphs.

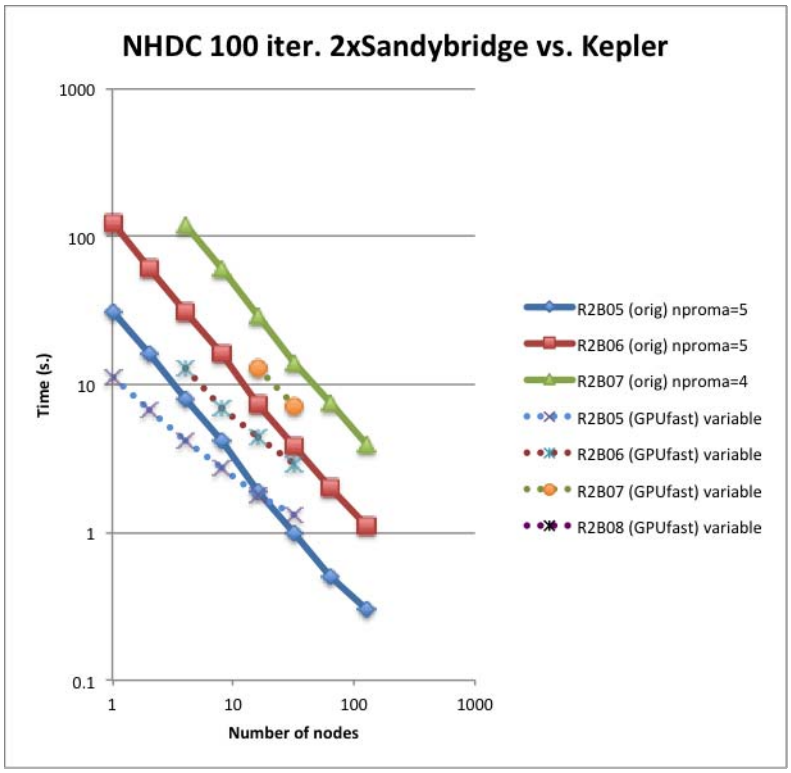

Figure 14: The execution time (in seconds) of the OpenACC multi-node ICON testbed code is depicted for dual-socket Intel Sandybridge nodes (solid lines) and single Kepler K20x GPU nodes (dotted lines) for three realistic global resolutions (R2B05, R2B06 and R2B07). The scenarios of interest are those where the problem just fits into memory (leftmost points). For these cases the GPU version performs roughly 2x faster than the CPU versions. As more nodes are added, the GPU version scales less well than the CPU version, due to limited GPU occupancy and increased message-passing latency.

4.3.3 Re-introduction status

As the OpenACC implementation was performed directly in the ICON DSL testbed, its re-introduction was straightforward. It was necessary to merge in modifications from the ICON DSL development team in order to keep the development versions synchronised. The resulting GPU-enabled ICON DSL was made available to the community through the corresponding PRACE WP8 hpcforge.org site. Knowledge transfer with the ICON DSL developers took place at MPI-M on June 3-4, 2013. This exchange was crucial, since the MPI-M team would like to maintain the OpenACC version as it continues development on the ICON DSL.

During the IS-ENES Workshop on HPC and Climate Models (Jan. 30 – Feb 1, 2013) the feeling was expressed that the OpenACC version should be incorporated into the ICON development trunk. While this was not one of the original project goals, this seemed like a sensible follow-on objective. Work on re-introducing the OpenACC version into ICON was started in April 2013. On July 23-26, 2013, CSCS developers travelled to DWD to work with ICON developers on the integration.

One project objective was not achieved: the integration of the physical parameterisations ("Physics") from the HP2C COSMO project. That project had intend to merge the COSMO Physics with the ICON Physics, after which the OpenACC-enabled parameterisations from HP2C COSMO could have been combined with the OpenACC implementation of the non-hydrostatic dynamical core. As that merge was delayed in HP2C COSMO, it was not possible to take this step in this project.

4.4 Ocean Models: NEMO and Fluidity-ICOM

4.4.1 NEMO

4.4.1.1 Overview

NEMO (Nucleus for European Modelling of the Ocean) is a modelling framework for oceanographic research, operational oceanography, seasonal forecasts and climate studies. It is of great strategic importance for the UK and European oceanographic communities.

The code itself is written in Fortran90 and parallelised using MPI. Portability and stability of the code base are of great importance. The parallelisation is done via geographic domain decomposition in the horizontal, but not vertical, dimensions. The fundamental equations are solved using a finite-difference scheme with a simply-connected, curvilinear grid. The model deals with both 2- and 3-dimensional fields, which are represented as Fortran arrays. These arrays are declared with haloes, and field values on the haloes must be exchanged frequently, which involves nearest-neighbour, point-to-point communications.

The MPI parallelisation of NEMO has served it well for many years. However, the trend for increasing core counts on the compute nodes of high-performance computers (*e.g.* IBM's BG/Q can support 64 processes per node) means that it is gradually becoming ill-equipped for making efficient use of such machines. Current best practice in programming for these computers is to use OpenMP across the cores in a node and MPI between nodes – the so-called 'hybrid' or 'mixed-mode' approach.

In WP8, STFC has explored the performance implications of various approaches to introducing OpenMP threading into the code. In doing this we have also experimented with the choice of array index ordering used for the 3-dimensional fields. NEMO's ancestor, OPA, was developed with vector architectures in mind and thus NEMO itself has the *x* or longitude grid index as the leading dimension for all of its arrays since this is typically the largest of the three grid dimensions. In contrast, codes developed more recently (such as WRF and POLCOMS) for scalar MPP architectures have tended to favour having the z grid index as the leading array dimension. By experimenting with two different kernels taken from the NEMO code, we aim to discover which array-index ordering is best for NEMO as well as the best way to introduce OpenMP.

These conclusions will then be used to inform the work of Italo Epicoco and Silvia Mocavero of the Centre for Mediterranean Climate Change (a member of the NEMO consortium) as they tackle the considerable task of introducing OpenMP throughout the NEMO code base.

4.4.1.2 Results

The main page for the NEMO section on the WP8 web pages can be found at: http://hpcforge.org/plugins/mediawiki/wiki/nemo/index.php/Main_Page

This page contains a brief introduction to NEMO and links to pages describing the 'Work Plan, Status and Results', 'Documents' and 'Downloads'. In the first of these we describe the work undertaken in more detail and the results achieved. Results are mainly given for the Intel Sandy Bridge chip, although some results for the BG/Q and Intel Xeon Phi are also shown. We also discuss the implications of the choice of array-index ordering for the performance of the code at the strong-scaling limit with the current domain decomposition for MPI.

The Documents page contains a link to the official NEMO documentation and the presentation prepared for the face-to-face meeting at CSCS, Lugano in March 2013. The Downloads page contains a link to the code tarball containing the kernels developed during the work.

4.4.1.3 Re-introduction status

Most of the activity since the previous deliverable has focused on continued liaison with the NEMO community.

The impact of this work on NEMO must be assessed in context. NEMO is a large, well-managed community code being actively developed by a consortium of members from several European countries. Most of the development effort in recent years has gone into new and/or improved scientific functionality. Performance and scalability concerns have not been neglected, but — for good reasons — the emphasis has been on developments that can be achieved in an incremental, evolutionary manner. The move towards asynchronous, parallel I/O strategies through the introduction of the XIOS I/O server into NEMO is a good example. Historically, I/O has been one of the main inhibitors to scalability in NEMO; the development of XIOS is alleviating the I/O scalability bottleneck, without disrupting other scientifically-important development activities.

In the UK, the NERC funded the National Oceanography Centre (NOC) to assess the development of ocean model dynamical cores required to meet operational and research needs on the timescale 2015-2025. This Ocean Roadmap Project identified in its final recommendations "the need to maintain and improve the efficiency of the models on evolving computational architectures" as one of the three main challenges facing the ocean modelling community [45].

The big question facing the NEMO consortium today is whether the incremental, evolutionary approach to performance improvements is fit to meet the challenge posed by establish trends in HPC. These trends include: (1) the decline of vector computers in favour of commodity, cache-based processors; (2) energy limitations and the many-core revolution, and (3) the arrival of accelerators.

Somewhat serendipitously perhaps, NEMO, despite being designed for vector computers, has stood up well to the challenge from commodity scalar processors. This is because the layer-oriented idiom of NEMO, developed for and tuned to vector computers, has benefited from increasing SIMD capabilities in commodity scalar processors in recent years. However, it is not clear that SIMD capabilities will continue to increase, given that many classes of application struggle to benefit from them. Despite the surprising longevity of NEMO's current level-last index ordering, there are still good arguments in favour of a level-first ordering: working sets tend to stay in cache better, halo exchanges are more efficient, and there are better options for eliminating redundant computations on land.

The many-core revolution has brought a fundamental change. Cores are no longer getting faster, and memory per core is already decreasing. Each generation of computer comes with more, but less capable, cores than the previous, and developers must discover ever more concurrency in their applications. We are driven towards hybrid programming paradigms in which thread parallelism (e.g. OpenMP) is used within shared-memory regions, and message passing (usually MPI) is used between them. For NEMO, as with many climate codes, this is more easily said than done. Its flat profile means that it is not enough to parallelise a few routines with OpenMP; the change is therefore pervasive, and hardly incremental; moreover, the overhead of managing, maintaining and testing a large code in the face of overlapping contributions from various quarters, not always from OpenMP experts, should not be ignored.

Whether accelerators could have a significant impact on NEMO is still an open question. Our early findings in the GNEMO project [51] were disappointing for accelerators, due to the low computational intensity of NEMO and the overhead of transferring data between host and device. On the other hand, NVIDIA's recent port of the NEMO core to GPUs using OpenCL is more encouraging; in this case, they were able to keep the main model fields resident on the device, and benefit from the high bandwidth within the device itself; however, the port is only

partial, and investigations are on-going.

The NEMO consortium is in the process of developing a strategy for mid-term NEMO development (on a 5-10 year timescale). These plans will be detailed in the "NEMO Perspectives" document, which will be published by the end of 2013. A set of white papers has been provided by the NEMO consortium partners, and presented to the NEMO community in a meeting of the NEMO Enlarged Developers' Committee (Paris, 18-19 June 2013). Stephen Pickles represented PRACE-2IP at this meeting. The papers from this meeting are available at: https://forge.ipsl.jussieu.fr/nemo/wiki/2013WP/EnlargedDevComJune2013.

The NEMO consortium must decide whether to continue with incremental improvements to performance and scalability, or to embark on wholesale restructuring and a pervasive programme of modifications that will touch most of the source code. This decision must be tensioned against requirements for continued scientific development, in the context of a limited pool of developer effort.

The work described here, along with related, complementary work funded outside PRACE-2IP, is informing the decisions facing the NEMO consortium.

In previous work, STFC developed a version of the NEMO core with a build-time option to switch between level-last and level-first array index orderings, with a view to enabling the complete elimination of redundant computation on land [50]. On-going work in the FINISS (Further Improvements to NEMO In Shallow Seas) project is extending this to eliminate redundant computations beneath the sea bed, and is quantifying the efficiency savings that would follow from a complete implementation of this strategy. These developments have been committed to branches of the NEMO source repository, but with the exception of dynamic memory allocation and a few bug fixes, the changes have not been incorporated into the trunk. The reason is that the scale of these changes, coupled with the pace of science-driven developments impacting the same source files, means that the merge and subsequent validation would require a prolonged, concerted effort from the NEMO core development team, and a long-term commitment from the NEMO consortium to their maintenance. Nonetheless, this branch provides a platform for investigation of NEMO's potential for scalability, and a reference for a possible future implementation should the NEMO consortium so decide.

Work by CMCC on a hybrid MPI+OpenMP implementation of their NEMO-based operational model is also ongoing. The urgency of a hybrid implementation is particularly acute for CMCC, because achieving best performance from the IBM BlueGene/Q platform on which their operational model runs requires the ability to oversubscribe hardware cores with software threads.

Given that two pervasive, disruptive changes to the NEMO trunk are being proposed (i.e. level-first index ordering, and MPI+OpenMP hybridisation), it is important to understand the interactions between the two. The present study of the effect of the choice of array-index ordering on OpenMP scalability meets this goal. This work was presented at the conference "EASC2013: Solving Software Challenges for Exascale" in Edinburgh, April 2013, and a paper has subsequently been prepared for publication [51].

Meanwhile, it is recognised by the NEMO consortium that the NEMO code is carrying an unwieldy overhead of obsolete build-time options that needs to be reduced [47]. There is also a growing recognition that NEMO, like many traditional climate codes, has very limited separation between the concerns of the natural scientist (who defines the algorithms used by the model) and the computational scientist (who deals with the issues of parallelisation, performance, scalability and so on). The NEMO consortium is therefore interested in the concerted attempt by the GungHo project to separate these concerns as it designs the

infrastructure for the UK Met Office's next generation atmospheric model [48].

Collectively, this work has directly informed both the UK Ocean Roadmap [45][46], and the NEMO perspectives process through the Met Office and NOC white paper, and presentations and contributions to discussions at the NEMO Enlarged Developers' Committee Meeting. The developed software has been committed to a branch of the NEMO source repository. Its long-term impact cannot be foreseen until the NEMO perspectives process is complete.

4.4.2 Fluidity-ICOM

4.4.2.1 Overview

Fluidity-ICOM is built on top of Fluidity, a general-purpose adaptive unstructured finite element multiphase CFD code that is capable of modelling a wide range of fluid phenomena involving single and multiphase flows. It has been applied in the sphere of oceanographic modelling as a next generation model, robust in its application to coupled scales from global to coastal to process. Together with innovative treatment of bathymetry using surface conforming finite elements, the use of unstructured adaptive meshing in parallel allows the method to optimally reveal physical details with the aid of user defined error fields. The mesh and hence computational resources are concentrated only when and where required by the physics whilst being relaxed elsewhere. Fluidity-ICOM uses state-of-the-art third party software components whenever possible. These libraries include for example NetCDF (I/O) [66], PETSc (sparse iterative solvers) [63], and Zoltan (parallel data-management services) [68].

One of the main objectives of this work was to optimise Fluidity-ICOM for modern supercomputers with NUMA nodes. Hybrid OpenMP/MPI offers new possibilities for optimisation of numerical algorithms beyond pure distributed memory parallelism. For example, scaling of algebraic multigrid methods is hampered when the number of subdomains is increased due to difficulties coarsening across domain boundaries. The scaling of mesh adaptivity methods is also adversely effected by the need to adapt across domain boundaries.

4.4.2.2 Work plan

The work plan prepared for WP8 intended to continue work on parallelisation of other assembly kernels for different equations with different methods, e.g., CG (Continuous Galerkin) and CV (Control Volume), based on benchmark test cases such as GYRE, 3D Backward Facing Step, and Collapsing Water Column. We also planned to optimise the threaded HYPRE library and the PETSc OpenMP branch usage for linear preconditioners/solver for large core counts. These developments/optimisations have the capability to transform Fluidity-ICOM into a fully hybrid code.

4.4.2.3 The main outcomes of WP8 work on Fluidity-ICOM

- Fluidity-ICOM matrix assembly kernels have been fully threaded.
  - The performance results indicate that node optimisation can be done mostly using OpenMP with an efficient colouring method.
- Benchmarking results with threaded HYPRE and the PETSc OpenMP branch show that the mixed mode MPI/OpenMP version can outperform the pure MPI version at high core counts where I/O becomes a major bottleneck for pure MPI version.
- Fixed threaded HYPRE compiling and running issues.
- BoomerAMG performance in HYPRE has been investigated and compared with Fluidity's own MG preconditioner both in respect to within-node performance and performance at high core counts.

- The current threaded PETSc branch with Fluidity has been optimised and benchmarked on up to 32K cores.

4.4.2.4 Performance Results

The benchmarking and performance tests were carried out on the UK National HPC platform HECToR, which is a Cray XE6 system. It offers a total of 2816 XE6 compute nodes. Each compute node contains two AMD 2.3GHz 16-core processors giving a total of 90,112 cores, offering a theoretical peak performance of over 800 Tflops. There is presently 32 GB of main memory available per node, which is shared between thirty-two cores, making a total machine memory of 90 TB. The processors are connected with a high-bandwidth interconnect using Cray Gemini communication chips. The Gemini chips are arranged on a 3 dimensional torus.

4.4.2.4.1 Performance consideration of NUMA architecture

One of the key performance considerations for achieving performance on ccNUMA nodes is memory bandwidth. In order to optimise memory bandwidth, the following methods have been employed to ensure good performance:

- First-touch initialisation ensures that page faults are satisfied by the memory bank directly connected to the CPU that raises the page fault;
- Thread pinning to ensure that individual threads are bound to the same core throughout the computation.

Thread pinning has been used through Cray *aprun* with all benchmark tests.

4.4.2.4.2 Matrix Assembly Performance results and analysis

After applying the first touch policy, the wall time has been reduced from 45.127 seconds to 38.303 seconds using 12 threads on Cray XE-Magny Cours. However, even after applying a first-touch policy, there is still a sharp performance drop from 12 threads to 24 threads. The problem appears to be connected to the use of FORTRAN automatic arrays in the matrix assembly kernels for support of p-adaptivity. There are many such arrays in the matrix kernels. Since the compiler cannot predict their length, it allocates the automatic arrays on the heap. In order to keep memory allocation thread safe, memory allocation by all threads will be effectively serialised by waiting on the same lock. Thread-Caching malloc (TCMalloc [43]) resolves this problem by using a lock-free approach. It allocates and deallocates memory (at least in some cases) without using locks for synchronisation. This gives a significant performance boost for the pure OpenMP version, which performs better than the pure MPI version within a compute node (the details and performance figures can be seen on hpcforge [44]).

4.4.2.4.3 Sparse linear preconditioner/solver performance results and analysis

Fluidity-ICOM uses PETSc for solving sparse linear systems. Many other scalable preconditioner/solvers can be called through PETSc interface, eg: HYPRE. Previous studies have already shown that Fluidity-ICOM spends the majority of its run time in sparse iterative linear solvers. This work-package focuses on BoomerAMG from HYPRE as a preconditioner as it has already been fully threaded.

We have experimented with several versions of HYPRE, but have found that only HYPRE-2.9.1a works correctly with Fluidity. We have compared the BoomerAMG preconditioner with Fluidity's own Multigrid preconditioner for the pressure Poisson solver of the lock exchange test case. The results show that BoomerAMG outperforms Fluidity's own multigrid pre-conditioner.

We use a separate PETSc branch where a number of low-level matrix and vector operators in the Mat and Vec classes have been threaded using OpenMP. In particular, task-based parallelism and an explicit thread-level load-balancing scheme are utilised in this branch to optimise the parallel scalability of sparse matrix-vector multiplication. However, the Krylov Subspace Methods and preconditioners implemented in the KSP and PC classes have not been threaded explicitly. These components, such as the KSP classes for CG and GMRES, are based on functionality from the Mat and Vec classes and thus inherit the thread-level parallelism implemented at the lower levels. Moreover, certain frequently used preconditioners, such as Symmetric Over-Relaxation (SOR) or Incomplete LU-decomposition (ILU), have not been threaded due to their complex data dependencies. These may require a redesign of the algorithms to improve parallel efficiency.

We have shown the threaded performance results for solving the pressure Poisson equation using the preconditioner HYPRE BoomerAMG and solver Conjugate Gradient. It indicates that pure MPI performance starts to degrade from 64 HECToR nodes (2048 cores) onwards. Here the hybrid mode begins to outperform pure MPI (the details and performance figures can be seen from [44]).

We have also demonstrated that I/O has benefitted from mixed mode parallelism. In order to simplify the complexity, all writes have been switched off. Therefore, the only I/O involved is the reading of input which includes the flml file and mesh files. The performance results showed that significant I/O efficiency has been achieved by using mixed mode parallelism. For example, in the mixed-mode implementation only four processes per node are involved in I/O (with pure MPI potentially 32 processes per node are performing I/O under current Phase 3 of HECToR). This reduces the number of metadata operations on large numbers of nodes, which may otherwise hinder overall performance. In addition, the total size of the mesh halo increases with the number of partitions (i.e. number of processes). For example, it can be shown empirically that the size of the vertex halo in a linear tetrahedral mesh grows proportionally as $O(P)$, where P is the number of partitions. Overall, the use of hybrid OpenMP/MPI has decreased the total memory footprint per compute node, the total volume of data to write to disk, and the total number of metadata operations based on the files-per-process I/O strategy. More details and performance figures can be seen on [44].

Combining our previous efforts in developing thread-level parallelism in finite element matrix assembly kernels and the efforts in WP8, we are now able to run Fluidity-ICOM in full MPI/OpenMP mixed-mode. Figure 15 shows the total Fluidity-ICOM run-time and parallel efficiency. Clearly pure MPI runs faster up to 2048 cores. However, due to the halo size increasing exponentially with number of MPI tasks, the cost of MPI communication becomes dominant from 4096 cores onwards, where the mixed mode begins to outperform the pure MPI version.

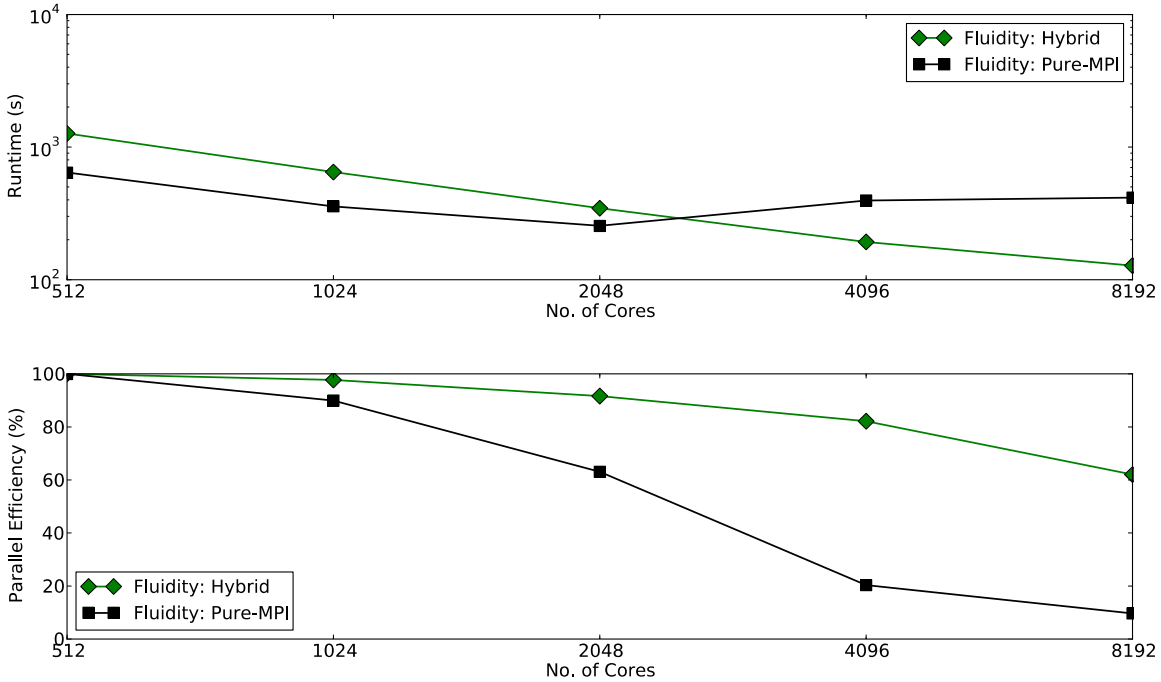

Figure 15: Strong scaling results for the whole Fluidity-ICOM up to 256 XE6 nodes (8192 cores). All hybrid modes use 4 MPI ranks per node and 8 threads per rank

4.4.2.5 Re-introduction status

Fluidity-ICOM uses the Buildbot tool to automate code verification. In the tests directory, there are more than a thousand test cases supplied for a wide range of problem sizes. Any commits, before being merged into main trunk, are required to complete all validation tests in the test directory.

Buildbot checks out and builds on various platforms with several different compilers. Any errors are relayed back to developers. If a failure is detected in a test problem, the developers are notified details via email. Statistical information about code quality is automatically collected from the newly validated code. This allows for the monitoring of performance. Results are made available via a web interface. This modern approach to software engineering has yielded dramatic improvements in code quality and programmer efficiency.

All our contributions from PRACE 2-IP WP8 for Fluidity-ICOM have already passed through all validation tests and have been merged into Fluidity-ICOM main trunk on Launchpad [54].

Due to the software enabling work undertaken in WP8, the overall performance of Fluidity-ICOM is very much improved. The parallel performance of the code now scales up to more than 32 thousand cores. Better scaling means that we can now reduce the time to solution by increasing the number of cores beyond what was previously possible. Indeed, aside from other scaling benefits, hybridisation has greatly reduced I/O, allowing us to immediately start running on an order of magnitude more cores than before. Fluidity-ICOM users are now realising that previous limits have been significantly alleviated and are looking to take advantage of this new capability across a wide range of applications in geophysical fluid dynamics. Figure 16 shows how the hybrid MPI/OpenMP version of Fluidity-ICOM can be used to model UK west coast domain (Courtesy Dr. Alexandros Avdis from Applied Modelling and Computation Group (AMCG), Imperial College London).

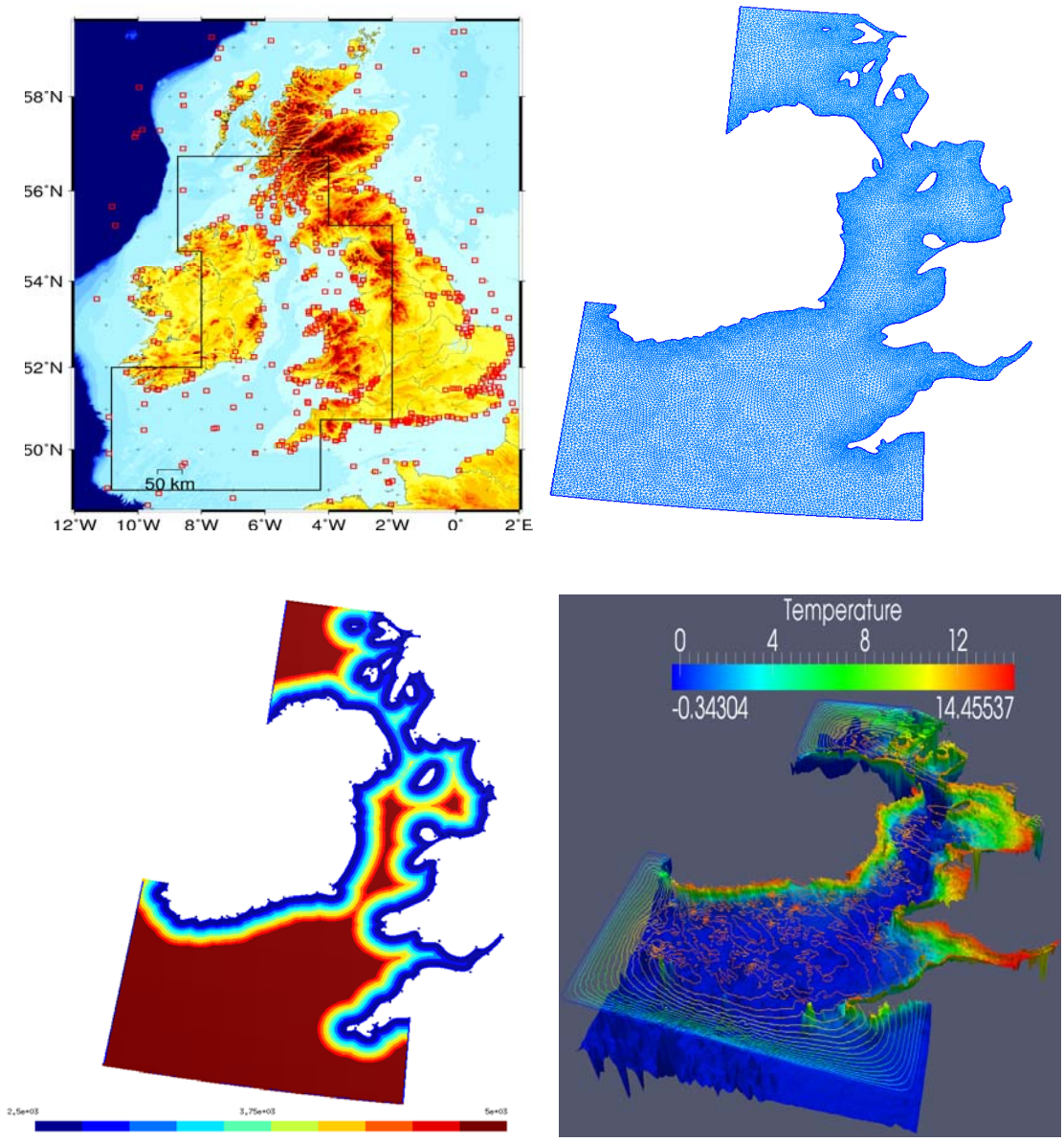

Figure 16: UK West Coast Domain Map (top-left), The Mesh for UK West Coast Domain (top-right), The domain with Mesh Size (bottom-left), UK west coast domain fine resolution results (bottom-right)

5 Material Science

In this section, the results and re-introduction status for the Material Science domain are presented. Four applications, devoted to electronic-structure calculations and materials modelling at the nanoscale, have been the subject of WP8 work. The first two, ABINIT and Quantum ESPRESSO, are large packages that implement a number of different algorithmic solutions (density-functional theory, plane waves, and pseudopotentials). The other two codes, Siesta and Exiting/ELK, adopt more specific approaches to describe the systems under investigation.

5.1 ABINIT

5.1.1 Overview

ABINIT is a package, available under the GNU General Public Licence (GPL), whose main program allows one to find from first principles the total energy, charge density, electronic structure and miscellaneous properties of systems made of electrons and nuclei (molecules and periodic solids) using pseudo-potentials and a plane-wave or wavelet basis. The basic theories implemented in ABINIT are Density-Functional Theory (DFT), Density-Functional Perturbation Theory (DFPT), Many-Body Perturbation Theory (MBPT), and Time-Dependent Density Functional Theory (TDDFT).

Historically, ABINIT uses plane-waves to describe the electronic wave functions; in recent years, a development of wave functions on a wavelet basis has been introduced. Before WP8, ABINIT parallelisation was performed using the *MPI library*. Some code sections of the ground-state part were also ported to GPU.

Most of the performance tests mentioned in the following have been performed on *TGCC-CURIE PRACE* French supercomputer. They all have been performed on a 107 gold atoms simulation cell.

Goals of the work in WP8

The performance analysis phases [3] [4] [5] were divided in four sections:

1. ground-state calculations using plane waves,
2. ground-state calculations using wavelets,
3. excited states calculations,
4. linear response calculations.

The work in WP8 has been driven by CEA-Bruyères-le-Châtel (GENCI) and involved four groups of developers:

1. *CEA – Bruyères-le-Châtel (Paris, France)* : Ground-state + plane waves
2. *CEA – INAC (Grenoble, France)* : Ground-state + wavelets
3. *UCL (Louvain-la-Neuve, Belgium)* : Excited States, Linear Response
4. *BSC (Barcelona, Spain)* : Linear Response

The main goals of the work were:

- Introduction of a shared memory parallel approach
- Improvement of the process distribution
- Improvement of parallel I/O
- Automation of process distribution (help tool for user)
- Automatic optimal code generation

Overview of main achievements

- *Ground-state*, *Excited states* and *Linear Response* parts of the code can now benefit from shared memory parallelism (OpenMP*)*,
- Load balancing of the *Ground State* part has been improved (band and FFT distributions),
- Parallel Input/Ouput has been accelerated in the *Excited state* and *Linear Response* parts by the use of MPI-IO library,
- Linear and Matrix Algebra performance has been improved by using external optimised libraries,
- Distribution of processes over 6 levels of parallelisation can now be done via an automation tool,
- A first set of tools to automatically generate optimised code has been tested for *Ground State* calculations using *wavelets*.

5.1.2 Results

Shared memory parallelism

The shared memory parallelism has been introduced in the following code sections:

1. Linear and matrix algebra
2. Fast Fourier Transforms
3. Non-local operator application

For points 1 and 2, it is possible to use OpenMP versions of the routines delivered in the package, or to use — only on Intel architectures — the multi-threaded MKL library.

For the application of the non-local operator (point 3), a partial rewriting of the routines was necessary. Now, two versions of these routines exist, one without any OpenMP directives, the other one for the calculations using several threads. In order to benefit from the maximum efficiency and to favour vectorisation, the multithreaded version implements the loops in an order which is not necessary optimal when running sequentially. The multi-threaded parallelisation is fully compatible with all other parallelisation levels: ABINIT now implements a complete hybrid parallelism solution.

An example of the performance of the *ground state* code section obtained on TGCC-Curie using the MKL library is shown in Table 4. For the test case presented here (107 gold atoms), 32 MPI processes have been used.

The generalisation of these results is not an easy task. The performance strongly depends on the load repartition in the different sections of code: a heavy material (*f* electrons) makes demands mainly on the non-local operator; a light material uses the Fast Fourier Transforms more. Also, the linear algebra load is very sensitive to the system size. Figure 17 shows the performances of the linear response part of ABINIT.

peedup ➔ # threads ↓	Total	Linear Algebra	Local Hamiltonian (FFT)	Non-local Hamiltonian
1	1.0	1.0	1.0	1.0
2	1.9	2.1	1.5	1.9
4	4.0	4.9	2.9	3.9
8	5.3	6.1	3.5	6.9

Table 4: Speedup of the hybrid MPI/openMP ground-state part of abinit. The test is the computation of a 107 gold atoms system on 32 MPI processes. Speedups greater than the theoretical speedup can be explained by memory effects.

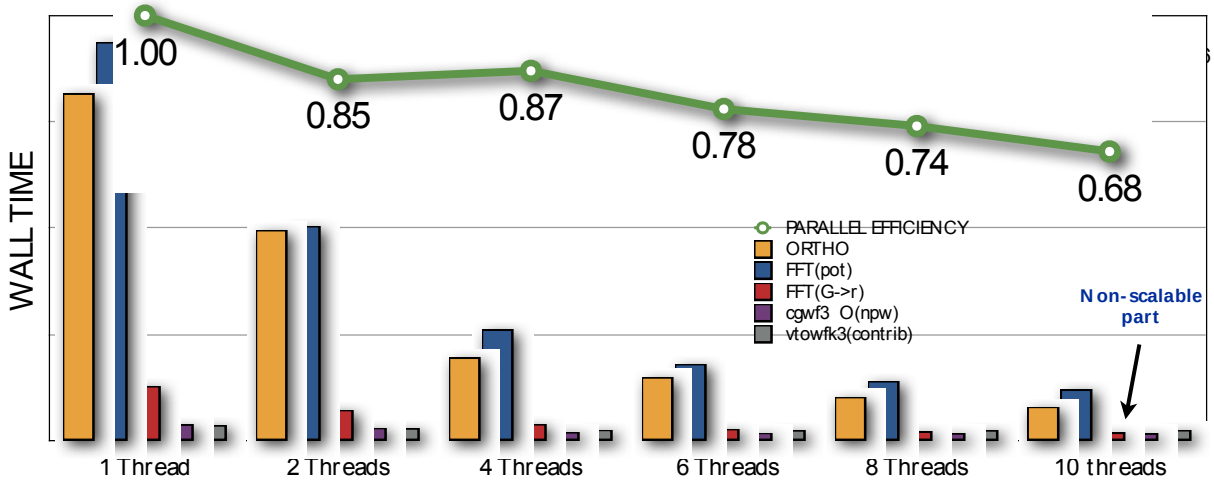

Figure 17: Efficiency of the multithreaded *linear response* part of ABINIT. The test is the computation of a 29 $BaTiO_3$ atoms system on 1 MPI process.

Load balancing

The load balancing of the ground-state code is now improved (two load balancing issues have been detected during the "performances analysis" phase).

The load imblance due to band distribution was easy to correct. A better repartition of bands over the processes has been implemented. It is mainly efficient when the number of processes does not divide the number of bands. As indicated in Figure 18, the waiting time before each orthogonalisation step has now decreased (see blue area).

The load balancing due to plane-wave distribution required more work. The previous implementation was based on a plane-wave selection with a "modulo" function; *(x,y)* planes were attributed to each MPI process. A new "dummy" distribution has been implemented making the code more modular. It has not been possible to avoid a small communication before each FFT. Figure 18 shows the new behaviour of the code.

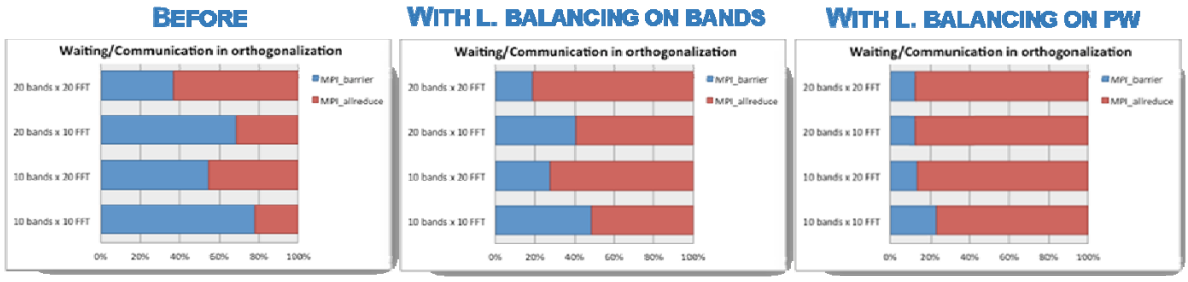

Figure 18: Repartition of waiting and communication time in ABINIT before each orthogonalisation step. On the left: before the modifications; in the middle: with the new distribution over bands; on the right: with all the modifications. The test is the computation of a 107 gold atoms system on 32 MPI processes.

Parallel I/O

In some sections of the code, especially for *linear response* and *excited states* calculations, a bottleneck was clearly identified at the level of Input/Output. The reading of wave functions (*inwffil* routine) was not performed with a parallel library. This has been corrected and the bottleneck has been partly reduced. A comparison of the time repartition before and after the present work is shown in Figure 19.

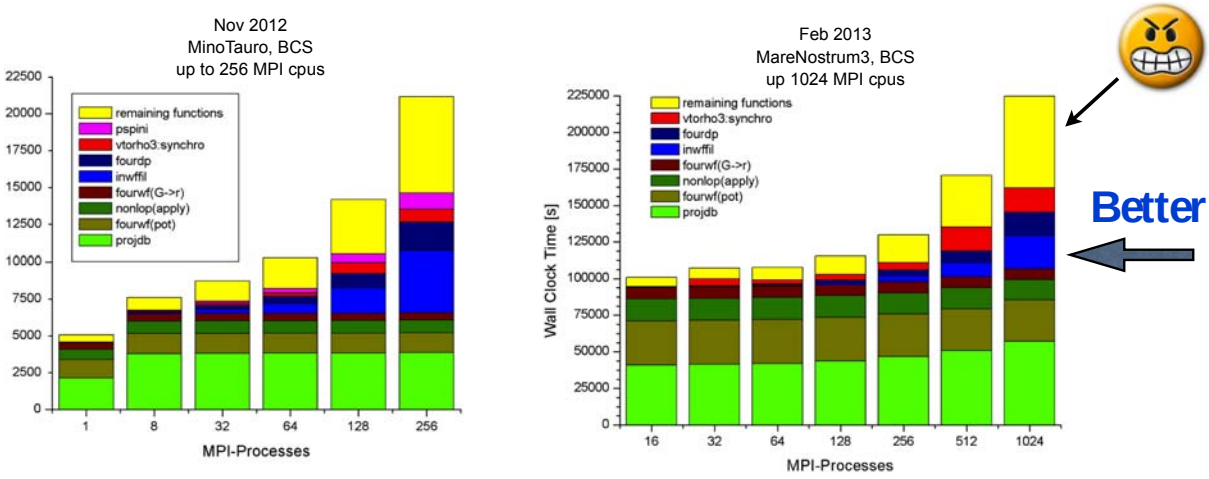

Figure 19: Repartition of time for a *linear response* calculation (29 $BaTiO_3$ atoms). The "blue" section has been reduced. On the left before the modifications; on the right after the modifications. Note that the new version has been tested on more MPI processes (the horizontal range is enlarged).

Linear and Matrix Algebra libraries

The data-structures related to the interface of ABINIT with the Linear and Matrix Algebra routines have been totally reorganised. The goal was to facilitate (in the future) the link with new external libraries. Also, the memory performance has been improved by avoiding several copies of arrays.

In order to find an alternative to the ScaLapack library, ABINIT has been interfaced with the ELPA library (Eigenvalue soLvers for Petaflops Applications [16]). The performance obtained is presented in Figure 20. For the standard case (107 gold atoms), the final gain is about 10% to 20%; but for other computations, on larger systems, a speedup of 50% can be observed. This is the case, for instance, for the computation of the *ground state* properties of a 512 Nickel atoms system (2500 electronic bands).

ABINIT has been also linked to the PLASMA library (Parallel Linear Algebra for Scalable Multicore Architectures [17]) for a future use on multicore architectures. Under development,

this library did not include the computation of eigenvectors at the time of the implementation. For that reason it has not been fully tested.

Np_slk	Wall MKL SLK	Wall MKL Elpa
1	239,5	239,5
2	255,2	235,5
4	243,9	255,7
8	249,4	201,7
16	224,7	208,9
36	243,5	242,8
72	258,3	258,3
144	287,2	240,7

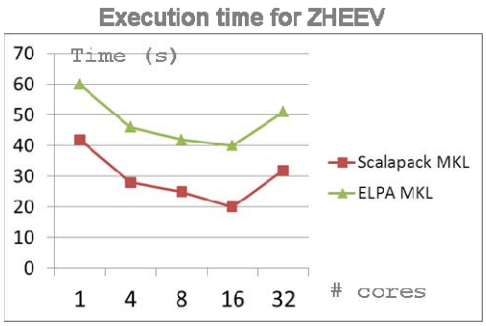

Figure 20: Matrix Algebra benchmark for ABINIT (107 gold atoms system): Intel MKL library vs ELPA library. "np_slk" stands for the number of MPI processes sent to the ScaLapack routines. A shown in the figure, ELPA and MKL scaling factors are equivalent. ELPA appears as better in terms of pure performance.

Automation of process distribution

A complete automation solution is now available for the ABINIT user. In practice, users do not need anymore to give any parallelisation keyword in the input file. The code starts in any case, using a process distribution determined by a set of simple rules. On demand, ABINIT can perform a series of small benchmarks in order to refine the distribution of processes according to the system size and the computer architecture (hardware and software).

Three levels of automation can be used:

1. A simple heuristics based on empirical rules
2. A series of benchmarks of the diagonalisation/orthogonalisation routines to refine the prediction of the simple heuristics
3. A series of benchmarks to optimise the parameters of the Matrix Algebra library (number of processes, use of GPU…)

Automatic code generation

As mentioned above, ABINIT uses historically plane-waves to describe the electronic wave functions; more recently, a development of wave functions using a wavelet basis has been introduced (for the ground state calculations), using wavelet transforms and a specific Poisson operator in real space. The implementation of wavelets has been achieved in the project named *BigDFT,* which is an inseparable part of the ABINIT project.

The efficiency of the wavelet transform strongly depends on the computer architecture. Even the simplest convolution routine shows different behaviours on several types of computer.

A generic approach to optimise application code has been proposed and applied to the convolution code included in the *BigDFT* library. The idea is the following: in a given application, optimising a frequently used operation or function will automatically benefit application performance. However, code optimisation depends on the target hardware architecture, namely the number of cores, their characteristics, the way they are interconnected, the cache levels, the memory distribution, etc. In consequence, in order to choose the best optimisation configuration, developers should be able to perform the following actions:

1. Identify the possible types of optimisations for the frequent operation,
2. Identify the parameters that characterise these optimisations,
3. Be able to produce combinations of the possible optimisations and lastly take into account the architecture parameters in order to select the best.

The purpose of the BOAST (Bringing Optimisation through Automatic Source-to-Source Transformations) project is exactly to automate this process. It is a new parametrised generator that can optimise *BigDFT* convolutions and generate multiple versions of a reference convolution, according to architecture dependent optimisation parameters. It has been tested on the cluster Tibidabo (MontBlanc project).

Simple	0.3 GFLOPS
Hand unrolled	3 GFLOPS
Hand vectorised	6.5 GFLOPS

Table 5: Comparison of performances of several implementations of a convolution operation.

5.1.3 Re-introduction status

Integration in the distributed version

The work done by the ABINIT group in the framework of the WP8 has been part of the *Continuous Integration* of the code through the server of the developer community.

The server is available through the distributed version control system *bzr* ("bazaar" [18]). Each developer can modify his own "branch" which is regularly merged into the "trunk" by the ABINIT final maintainer.

The *Continuous Integration* workflow relies on:

1. Extensive test suites (more than 800 automatic tests are provided in ABINIT package).
2. Daily reviewing of contributions; this obviously needs automation. For that purpose, the *buildbot* [19] package is used.
3. A "computer farm" management.

This ensures the possibility of a continuous feedback and a fast way to find eventual bugs. As soon as the modifications on the code are synchronised with the main *bzr* trunk, they are made available in the development version of the code. Regularly (each 6 months), the development version is fully checked and made available to the user community.

Figure 21 shows a sketch of the *Continuous Integration* workflow of ABINIT. More details about it have been described in a publication [20].

A mentioned above, the *Continuous Integration* of ABINIT code relies on a test suite: a set of automatic tests is executed on a "farm" of computing architectures. A computing architecture is made of the association of a slave (the hardware) and a builder (the software). The ABINIT test farm is made up of 20 slaves and 30 different builders. More than 800 automatic tests are executed each night, on each computing architecture in order to check each new submitted development.

In the frame of the WP8, several new capabilities have been added to the test suite:

- New automatic tests have been added in order to test the new functionalities developed for the WP8: unit tests for Matrix Algebra and FFT libraries, tests of the new automation tool for process distribution, and tests of the new "band" and "plane wave" distributions.
- A new slave has been bought: it allows running MPI parallel jobs on 32 MPI processes (or more).

- New builders have been added to the test farm: one builder to test the new multi-threading capabilities, one builder to test ABINIT in parallel on more than 32 MPI processes.

On March 28, 2013, ABINIT production version 7.2.2 was released. It contains all the new capabilities implemented in WP8.

Based on the previously described integration workflow, it can be confirmed that all these new developments have successfully passed the quality checks. They are all documented in the official online documentation of the code, meaning that they can be used by anyone downloading the code.

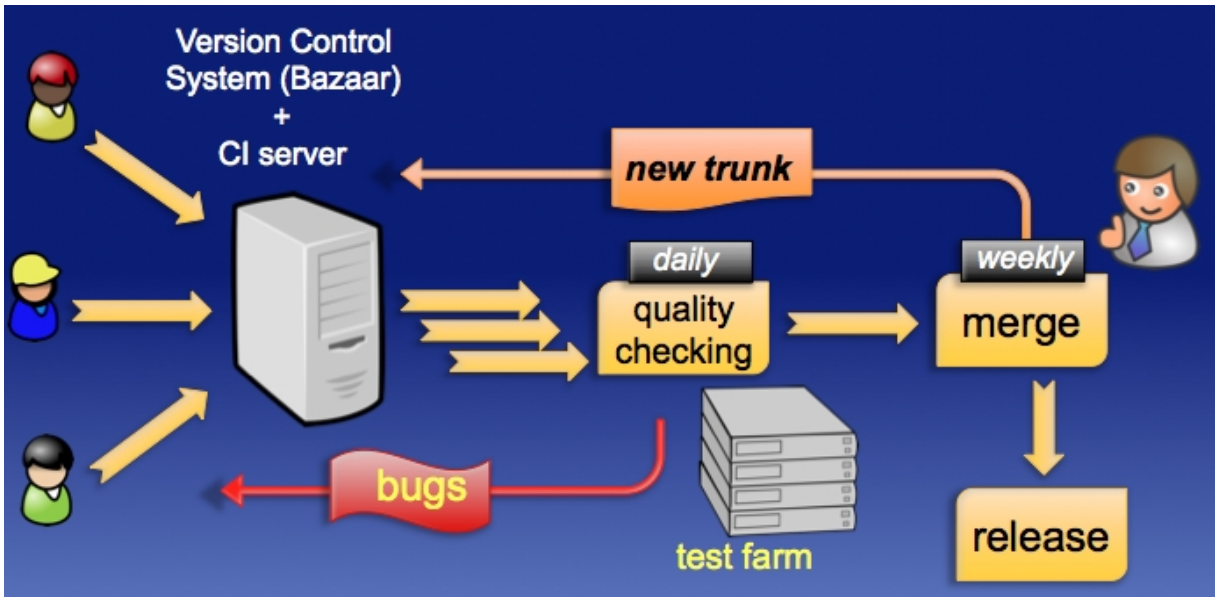

Figure 21: ABINIT *Continuous Integration* Workflow

Dissemination, impact for the user community

All the new developments have been presented during the "6th *International ABINIT Developer Workshop*" that was held in Dinard (France), March 15 to 18, 2013. The presentation is available on the public website of the workshop [70].

During a CECAM tutorial — "*Atomistic and molecular simulations on massively parallel architectures*" — a day was devoted to the presentation of ABINIT new capabilities. A "hands-on session" has been organised to help the participants experiment the automatic parallelisation and the hybrid parallelism new functionalities.

Other tutorials/schools are already planed for 2013 and 2014. One will be organised at the French TGCC and will be devoted to the exploration of the new parallel capabilities of ABINIT.

The new production version of ABINIT (v7.2.2) – including all contributions from WP8 – has been installed on several PRACE supercomputers. On TGCC-Curie, the measured performances are good: the hybrid version allows access to 8 times more CPU cores (above 8 threads, the scaling factor decreases because of architecture issues). The ELPA library shows relatively good performance.

This production version has also been installed on other supercomputers: Tier-1 GENCI computers (TGCC-Airain and IDRIS-Jade), and Stampede (Intel Sandy Bridge + MIC) at Texas University.

5.2 QuantumESPRESSO

5.2.1 Overview

QuantumESPRESSO is an integrated suite of computer codes based on density-functional theory, plane waves, and pseudo-potentials. The acronym ESPRESSO stands for opEn Source Package for Research in Electronic Structure, Simulation, and Optimisation.

The main goals of the work in WP8 were:

1. to enable state-of-the-art materials simulations.
2. to foster methodological innovation in the field of electronic structure and simulations by providing and integrating highly efficient, robust, and user-friendly open source packages containing most recent developments in the field.

The QuantumESPRESSO distribution (licensed under GP-GPL) offers users a highly portable and uniform installation environment. The web interface, qe-forge, provides to potential developers an integrated development environment, which blurs the line separating development and production codes, and engages and nurtures developers by encouraging their software contributions.

Work in WP8 has been driven by CINECA and involved ICHEC, IPB, and University of Sofia. Each partner contributed to the refactoring and improvement of the QuantumESPRESSO package. In particular, both CINECA and ICHEC were involved in the low-level modifications, implementing new levels of parallelisation, improving the linear algebra kernel and refactoring the general structure of the code. ICHEC, in addition, also focused on the refactoring of the Phonon module. IPB concentrated their effort on the EPW module, while University of Sofia contributed with testing of the new EXX implementation.

As a summary, the list of the accomplished activities is:

- OpenMP parallelisation (where lacking) and optimisation.
- Implementation of a level of parallelisation on energy bands.
- Parallelisation of the Phonon module.
- Implementation of the ELPA libraries for linear algebra calculations.
- Parallelisation of the EPW module.
- Testing and validation of the EXX calculations using the G2 test set.
- Testing and porting to new architectures.

5.2.2 Results

The OpenMP parallelisation scheme was already introduced in several QuantumESPRESSO modules before the WP8 work was started. However this implementation was partial and in some cases very naive and not optimised. This was due to the nature of the code, coming from a scientific community where some of the users were not experts in parallelisation techniques. WP8 tested this implementation, made it more homogenous inside the code, and, finally, fixed various bugs. Both CINECA and ICHEC contributed to the achievement of this task.

The parallelisation on bands relies on the fact the many do-cycles inside the QuantumESPRESSO code are intrinsically independent on the energy band index. This permits to include, in addition to those already implemented, a new parallelisation level. This new level can be exploited with success when the number of levels increases and takes great advantage from the multi-core systems now widely available. The band parallelisation has been implemented in three of the most used modules of the QuantumESPRESSO distribution: CP (Car-Parrinello, see Figure 22), PW (DFT with plan-waves) and GIPAW (NMR calculations). This work was conducted by CINECA and ICHEC.

The Phonon module allows the calculation of the linear-response properties of materials inside the QuantumESPRESSO framework. This module, however, lacked a satisfactory parallelisation that could run larger systems on current available HPC machines. Together with the implementation of a hybrid MPI/OpenMP parallelism, an OpenACC directive approach was also introduced. This was done to test the possibility of exploiting accelerators using PGI compilers on different architectures. This work was mainly conducted by ICHEC.

In QuantumESPRESSO, the scalability of the linear algebra calculations was related to the exploitation of the ScaLAPACK library. However, typical plane-wave calculations do not involve very large matrices and this can be a hard limit for ScaLAPACK both in terms of performance and scalability. ELPA libraries recently emerged as a way to (partially) overcome this kind of limitations. One-stage diagonalisation with ELPA is now possible inside QuantumESPRESSO. This obtains better results in terms of performance (see Figure 23) even on small calculations. This task was conducted by CINECA.

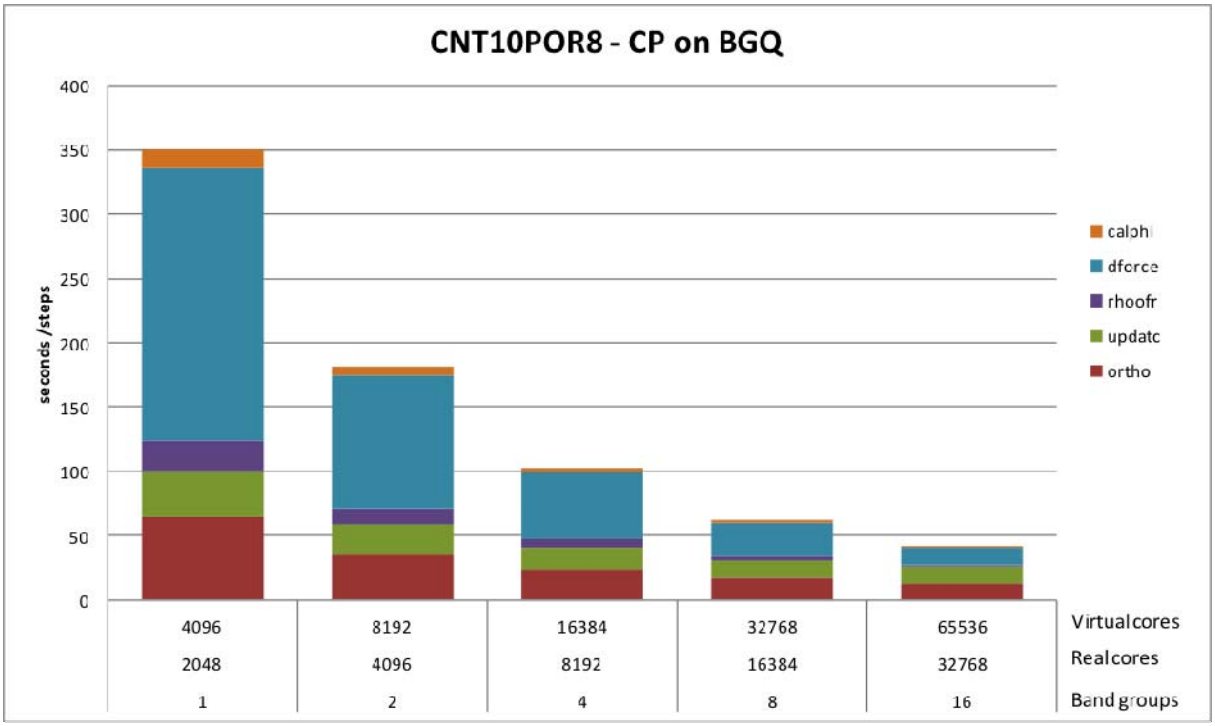

Figure 22: Car Parrinello simulation on a CdSe system (~1200 atoms). Using band parallelisation it is possible to reach a satisfactory scalability up to more than 6500 virtual cores

```
SCALAPACK+ELPA :
    Called by *egterg:
    h_psi        :   8413.42s CPU   4262.96s WALL (   18516 calls)
    s_psi        :    406.45s CPU    203.68s WALL (   18516 calls)
    g_psi        :     22.61s CPU     11.56s WALL (   17270 calls)
    cdiaghg      :   2715.00s CPU   1411.47s WALL (   17530 calls)
    cegterg:over :    614.68s CPU    311.11s WALL (   17270 calls)
    cegterg:upda :    621.48s CPU    316.24s WALL (   17270 calls)
    cegterg:last :    334.76s CPU    168.89s WALL (    1766 calls)
    cdiaghg:chol :    162.37s CPU     83.82s WALL (   17530 calls)
    cdiaghg:inve :    179.72s CPU     93.09s WALL (   17530 calls)
    cdiaghg:para :    663.38s CPU    363.62s WALL (   35060 calls)
```

SCALAPACK:

```
    Called by *egterg:
    h_psi        :   8605.96s CPU   4358.72s WALL (   18619 calls)
    s_psi        :    406.90s CPU    203.88s WALL (   18619 calls)
    g_psi        :     22.98s CPU     11.57s WALL (   17369 calls)
    cdiaghg      :   4198.63s CPU   2236.85s WALL (   17629 calls)
    cegterg:over :    626.34s CPU    317.64s WALL (   17369 calls)
    cegterg:upda :    704.67s CPU    364.05s WALL (   17369 calls)
    cegterg:last :    345.32s CPU    174.68s WALL (    1770 calls)
    cdiaghg:chol :    163.26s CPU     83.22s WALL (   17629 calls)
    cdiaghg:inve :    179.15s CPU     93.06s WALL (   17629 calls)
    cdiaghg:para :    861.79s CPU    465.61s WALL (   35258 calls)
```

Figure 23: Profiling of a PW run on a Graphene system (60 atoms). Above there are timings of calculation with ELPA. Below, data obtained using ScaLAPACK only. Comparison on the diagonalisation routine (cdiaghg) show the improvements obtained with ELPA

The work on the EPW code within WP8 was focused on introduction of additional levels of parallelisation that enabled the use of the code for larger systems, as well as better scaling of the code. EPW is the code for interpolation of electron-phonon coupling constants obtained on coarse grid of electronic k and phonon q points in the Brillouin zone, to a much denser grid of k and q points. Original implementation of the interpolation procedure in EPW contained the parallelisation over k-points only. For this reason, the usage of the code was limited to small systems only (up to ~5 atoms), since memory issues would arise for larger systems. Within WP8, the parallelisation over all other indices of electron-phonon coupling constants was introduced. The new implementation was tested to give the same results as the original implementation. It was shown that the new code can access quite large systems for this type of calculation (~30 atoms) on the machines with more than 3000 CPU cores (preliminary results are shown in Figure 24). Weak scaling tests showed an $\sim N^3$ scaling, as expected for this type of calculation. This work was conducted by the IPB team in collaboration with Prof. F. Giustino from Oxford University.

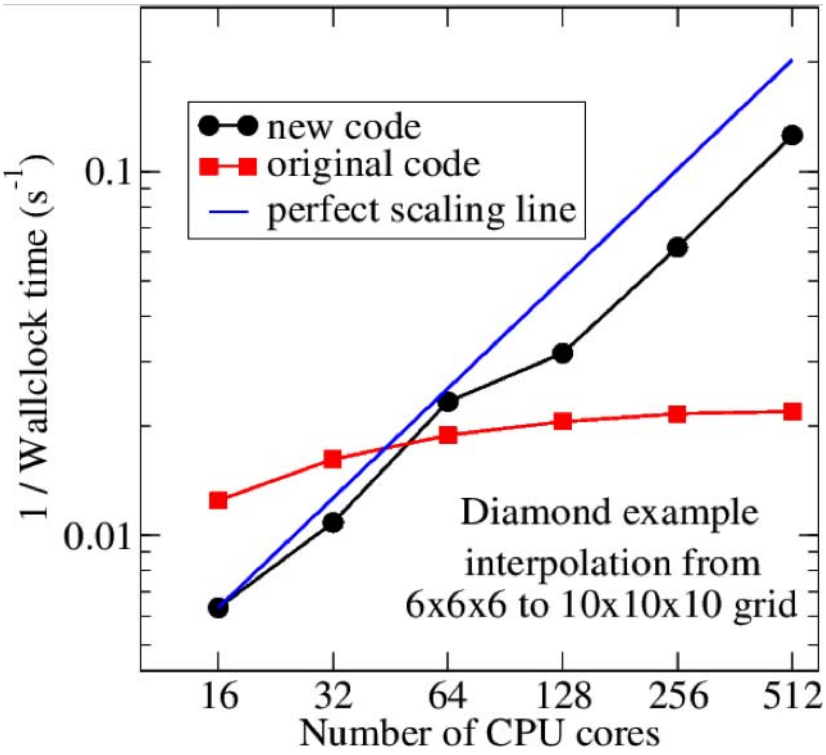

Figure 24: Preliminary test obtained with the EPW code showing the improved scalability with respect to the original code.

Together with the optimisation and the development in the code, an important part of the WP8 work was dedicated to the testing and validation of the functionalities of the code. In this area, the contribution produced by the University of Sofia was very important. They used the G2 test set to assess and validate the exact-exchange functionals recently introduced into the QuantumESPRESSO distribution. Different molecules have been tested, comparing results obtained with the B3LYP functionals with the G09 results. The good agreement found shows that the work done has been validated. Documentation produced in this task is available on PRACE WP8 hpc-forge website.

One of the aims of the QuantumESPRESSO community is to make this distribution as far as possible ready to be run on very different kinds of architectures, from home computers to large HPC facilities. This attitude was extended to GPGPUs and, more recently to MICs. The idea, pursued through the refactoring activity in WP8, was to make the code ready to be used with upcoming devices and, at the same time, easily maintainable. For GPGPUs, this aim was achieved through the implementation of the PhiGEMM libraries. In collaboration with WP11 'Prototyping', we tested the CP module on the Intel MIC platforms, gathering information to plan future developments of the QuantumESPRESSO distribution. This work was done by CINECA.

5.2.3 Re-introduction status

Unlike other codes, the accomplished work is part of the continuous integration of QuantumESPRESSO through a SVN server open to every user and developer. This ensures the possibility of continuous feedback and is a fast way to find bugs. As soon the modifications of the code are synchronised with the main SVN trunk, they are made available through the Quantum ESPRESSO website.

As well as continuous integration achieved through the SVN server, it should be mentioned that CINECA, as leader of the QuantumESPRESSO task, and DEMOCRITOS, the leading community behind the QuantumESPRESSO project, promoted a continuous exchange of

information and feedback. This activity drove the work on QuantumESPRESSO to where the needs of the communities were most urgent. Moreover, the dissemination work done in the WP8 framework by CINECA and partners was very useful in achieving a consolidated link between the PRACE and the community of condensed matter scientists, with particular attention to DEMOCRITOS.

Finally, the continuous feedback obtained from the scientific community showed that the obtained results during WP8 are valuable for the users and will permit them to obtain good scientific results using software of improved quality.

5.3 SIESTA

5.3.1 Overview

SIESTA (Spanish Initiative for Electronic Simulations with Thousands of Atoms) is both a method and its implementation as a computer program, to perform electronic structure calculations and ab-initio molecular dynamics simulations of molecules and solids.

For larger systems the most costly step is the diagonalisation, currently done with ScaLAPACK, a widely used and robust method. But it has two main drawbacks:

- It does not take advantage of one of the most important features of SIESTA: sparsity. SIESTA uses a local atomic orbital basis, resulting in sparse matrices. In particular systems featuring a lot of vacuum in the unit cell can be treated more efficiently than with other methods. To use ScaLAPACK, the matrices are filled with zeros and treated as dense matrices.
- Its scaling with the number of processors is limited. When increasing the number of processors for a given problem, the efficiency drops more and more, until at a certain point the time to solution even starts to grow again. The bigger the problem, the more processors can be used efficiently, but at the same time the computational effort grows with the third power of the system size ($O(N^3)$). This limits the size of systems that can be treated in reasonable time.

Practically the number of processors that can be used is limited to a few thousands, which is much fewer that current supercomputers provide to the scientific community: with future computers the situation will be even worse.

Thus the main goal within WP8 was to find other algorithms that are suitable for massive parallelisation and ideally take advantage of sparsity. To achieve this, the following has been done:

1. Implementing a prototype and testing the Sugiura Sakurai method
2. Testing the FEAST library
3. Implementing the eigenvalue solver of MAGMA to take advantage of GPUs
4. Testing a PEXSI prototype and developing it towards the needs of SIESTA
5. Implementing the PEXSI library into SIESTA

Unfortunately, due to the character of the problem to solve in SIESTA, the Sugiura Sakurai method and the similar FEAST library do not work as efficiently as needed for the massive parallelisation we aim for.

PEXSI, however, works fine within SIESTA, is particularly meant for sparse systems, and is able to use orders of magnitude more processors than ScaLAPACK. But its most powerful feature is that it reduces the computational cost depending on the system size N to

- $O(N^2)$ for 3D systems
- $O(N^{3/2})$ for quasi-2D systems

- *O*(N) for quasi-1D systems

Thus PEXSI will, with growing system size, always be faster than any O(N^3) method. But in contrast to other order-reducing algorithms,such as order-N codes, it does not use any simplifications limiting the range of applications or accuracy.

5.3.2 Results

PEXSI:
We compare PEXSI with diagonalisation on the basis of calculating the density matrix in one SCF step. While PEXSI calculates the density matrix directly, a solution by diagonalisation involves solving the generalised eigenvalue problem and calculating the density matrix from the eigenpairs. The matrix construction is not taken into account, firstly because it is basically the same for both solvers, secondly because for large problems it accounts for only a small fraction of thecomputation.

Figure 25 shows results for DNA strands of different lengths (1, 4, 9, and 16 revolutions). The number of processors used scales linearly with the problem size. For ScaLAPACK these numbers of processors are already relatively high, so that increasing them would not yield much better results. On the other side, PEXSI calculated all its 40 poles in parallel, so that for the matrix operations per pole only 4, 16, 36, and 64 processors were used. This could be raised while still being efficient. But even when comparing with diagonalisation, using the same number of processors, we see, that

1. PEXSI is 40 times faster than diagonalisation for the largest example
2. a problem with more than 10000 Atoms (~10000 orbitals) needs less than 2 minutes per SCF step with PEXSI
3. the advantage of PEXSI grows massively with system size (note the log scale!)

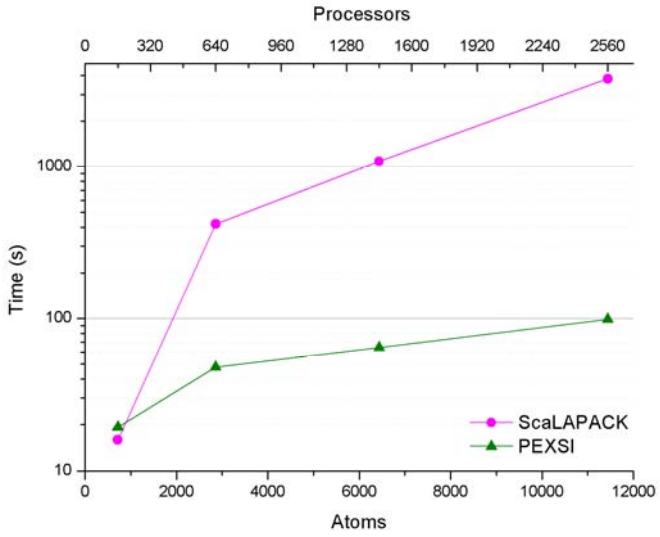

Figure 25: Performance of PEXSI-Siesta compared with ScaLAPACK. The test example is a DNA strand doing one revolution in a unit cell. Different problem sizes were generated by stacking unit cells. Here data for 1, 4, 9, and 16 unit cells are shown. The time displayed is the time for solving one SCF iteration (the matrix setup is not included).

SIESTA-PEXSI is already used for actual research on systems consisting of layers of graphene and BN. These subsystems feature slightly different interatomic distances, so Moire patterns show up. The periodicity of these patterns, which is the size of the unit cell needed, depend also on the relative orientation of the layers involved.

We investigated a series of systems with one graphene and one BN layer. The biggest unit

cell consists of about 13000 atoms and extends over 14 nm. Table 6 shows the results for two of these systems. This comparison does not show the full capability of PEXSI, since it takes only partial advantage of the very efficient parallelisation over poles. In these runs only 10 of 40 poles were treated in parallel, so one can multiply the number of processes by 4 and obtain nearly perfect speedup, while the efficiency of ScaLAPACK would decay rapidly. With a more efficient implementation of the computations per pole (work in progress), one could use over 100000 processes.

Atoms / orbitals	Solver	Processors	Time [s]
~8000 / ~100000	PEXSI	2560	1487
	Diagonalisation	2560	2751
~10000 / ~130000	PEXSI	3240	1528
	Diagonalisation	3240	3602

Table 6: Comparison of PEXSI and diagonalisation, showing the time for computing the density matrix in one SCF iteration.

MAGMA:

The range of applications for the MAGMA eigenvalue solver is quite limited: if the matrices are too small, the GPU does not work efficiently, if the matrix is too large, it does not fit into the memory of the GPU. Like LAPACK, it is meant for dense matrices, so a lot of memory is "wasted". Thus MAGMA cannot be a general solution, but it can boost the performance in the case of:

- a problem size of 500 – 1000 atoms
- using a single workstation for the computation.

As Figure 26 shows, the advantage over a computing node with 12 or more cores and optimised MKL library is not big, but a simple workstation with a few cores and usually less optimal LAPACK implementation can profit a lot from a GPU for a relatively small price.

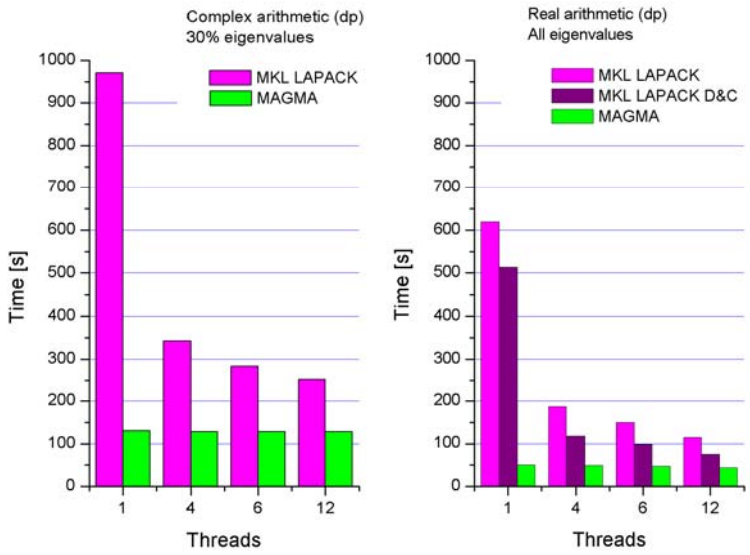

Figure 26: Performance of the MAGMA eigenvalue solvers for real and complex arithmetic compared to multithreaded LAPACK.

5.3.3 Re-introduction status

PEXSI has been fully integrated into SIESTA in the form of a library with a well-defined interface. Thus further developments of SIESTA and PEXSI can be done independently. Also the configuration of the PEXSI solver is integrated into the configuration of SIESTA in the

fdf format. Besides its good single-shot performance, PEXSI has proven to be robust regarding the total SCF loop for various problems of different character.

Ongoing work includes further improvements of the performance of several parts of PEXSI and heuristics for automatic configuration of some parameters, so that users need to set only a few options.

MAGMA has been implemented in a development version of SIESTA. The results shown in the previous chapters are based on this version.

5.4 Exciting/ELK

5.4.1 Overview

Exciting and Elk are two rapidly developing full-potential linearised augmented plane-wave (FP-LAPW) codes with a lot of common low-level functionality and a rich set of features. Both codes are used to study the electronic structure and excited state properties of the periodic solids (crystals) but can also be applied to atoms and molecules. The FP-LAPW method implemented in the codes is considered to be the gold standard among the all-electron methods for the electronic structure calculations of crystalline solids. The high accuracy of the method is achieved by the clever choice of the basis set, which captures both the atomic-like and nearly-free character of the electron wave functions in the different regions of the crystal.

The principal requests from the Exciting community for the code refactoring was to make it possible to run electronic structure simulations for large unit cells containing up to a thousand atoms. During the preparatory phase of the WP8 the major bottlenecks of the codes were identified and the decision was made to re-implement the basic low-level LAPW functionality in a standalone prototype library, which we have named SIRIUS. Our arguments for the library are the following:

- The number of changes that are necessary for a proper scaling and optimisation is quite significant and implementing these changes in Fortran 90/95 (and making the code still readable) is a tough job.
- The library encapsulates the architecture-dependent implementation from the scientifically relevant Fortran code.
- By having a common library we avoid a redundant work of optimising two codes (Elk and Exciting) simultaneously.
- Other LAPW-based codes may benefit from the common library.

In the time frame of the WP8 project we focused only on the refactoring of the ground state calculations, i.e. a path from the crystal structure input to the ground state charge density and magnetisation. We did not touch other parts of the code such as phonons, forces, linear response calculations, etc.

The main achievements of the WP8 work can be summarised in the following list:

- All the prototype functionality of the library, which is required to run ground state calculations at scale is implemented.
- Critical parts of the library are ported to GPU (using CUDA programming framework).
- Ground state runs with >1000 atoms in the unit cell have been demonstrated.
- The library is integrated back into the Exciting code and verified against the ‘canonical’ Fortran implementation.

5.4.2 Results

The emerging SIRIUS library is the main outcome of the WP8 effort of Exciting/Elk code refactoring. It re-implements basic low-level LAPW functionality and provides a universal solution for the scaling and optimisation of both codes. The SIRIUS library is written from scratch in C++ with the following features:

- MPI + OpenMP (+ GPU) parallel model. The coarse-grained parallelisation happens on the multi-dimensional (depending on the task) grid of MPI ranks, while the fine-grained parallelism is achieved using the OpenMP threads (and/or a GPU device).
- Efficient wave-function representation, which increases both the data locality and the number of dense linear algebra operations.
- Extensive use of external high-quality libraries: FFTW for Fourier transforms, HDF5 for large binary I/O, libJSON for parsing input JSON files, libXC for the comprehensive collection of exchange-correlation functionals and Spglib for finding and handling crystal symmetries. Optionally, ScaLAPACK and ELPA are used for distributed eigenvalue solvers and cuBLAS and MAGMA for GPU-enabled acceleration.
- Accurate numerical techniques. Analytical expressions are used wherever it is possible.
- Built-in testing and debugging capabilities: tests of individual components (if possible), run-time checks, stress tests of parallel execution (exactly the same ground state properties must be obtained with various MPI grid configurations, different number of threads and different eigenvalue solvers), verification and validation of the results.

The strong scaling of the new code is shown in Figure 27, which compares the ideal scaling (red) versus the actual scaling (blue) for the ground state calculation of Eu_6C_{60} on an AMD Interlagos 2×16-core platform with two MPI ranks per node and 16 threads per rank. The X-axis shows the dimensions of the MPI grid, used by the distributed eigenvalue solver ELPA.

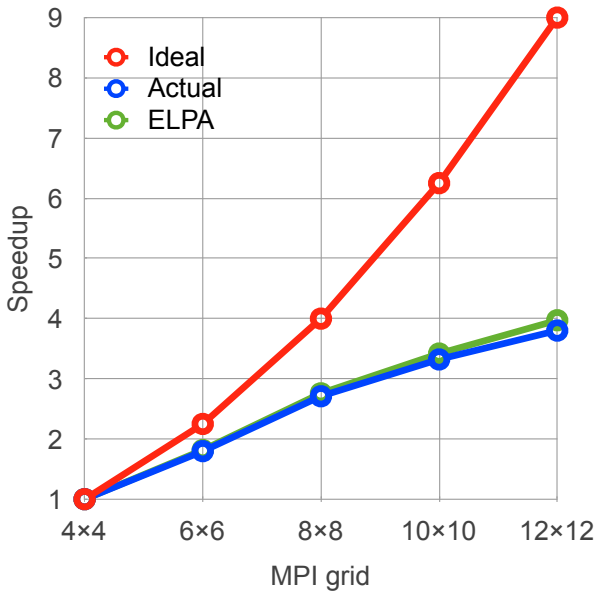

Figure 27: strong scaling of the developed code comparing the ideal scaling (red) versus the actual scaling (blue) for the ground state calculation of Eu_6C_{60} on an AMD Interlagos 2×16-core platform.

Figure 27 shows that the scaling of the code is limited by the scaling of the underlying generalised eigenvalue solver (green line). A new fast and scalable distributed eigenvalue solver is required in order to boost the performance.

5.4.3 Re-introduction status

We have started the work on the integration of the SIRIUS library into the Exciting community code. Together with the main Exciting developers we have formulated the two basic requirements for the library incorporation:

- Old (canonical) Fortran implementation must be intact.
- New library must not work as a 'black box' and should possess enough granularity to replace critical Fortran subroutines.

We use the preprocessor directives (#ifdef #else #endif) to control the compilation of the Exciting code. The example of the source code for the call to the exchange-correlation potential generation is below:

```
   if (input%groundstate%usesirius) then
#ifdef SIRIUS
     call sirius_generate_xc_potential(rhomt, rhoir, vxcmt, vxcir)
#else
     stop "Not compiled with SIRIUS library"
#endif
   else
     call potxc
   endif
```

The following solution allows compiling and linking the Exciting code with and without the SIRIUS library. If the code is linked with the library support, it can still be switched off by the input token. We gain a lot of flexibility in terms of code management for the price of some loss in code readability. The second item (API granularity) is being constantly improved during the re-introduction phase.

The Exciting+SIRIUS code is now available for the Exciting code developers for the revision and feedback. The following roadmap was drafted to track the progress in the SIRIUS library incorporation:

- Validate and verify ground state calculations for closed shell (spherical) atoms (He, Be, Ne, etc.).
- Validate and verify ground state calculations for open shell spherical atoms (H, Li, N, etc.).
- Validate and verify ground state calculations for open shell non-spherical atoms (B, C, O, F, etc.).
- Validate and verify ground state calculations for molecules, for example atomisation energies for the G2 molecules set.
- Verify forces and atomic relaxation.

For the validation of the ground state total energies of atoms and molecules we are planning to use two independent full potential codes: NWChem (Gaussian basis) and moldft (multi-resolution adaptive grid). Both codes agree to a μHa precision and can serve as a reference point for the total energy calculation. When the validation and verification phase is finished, the SIRIUS library may be released to the public domain for further tests and improvements. Together with this, the library will be ready for the presentation to the Elk code community.

6 Particle Physics

In the Particle Physics area, the focus is on Quantum Chromodynamics with the PLQCD code. Achieved results and re-integration status are described.

6.1 PLQCD

6.1.1 Overview

Lattice QCD is one of our forefront methods to understand the theory of strong interactions, known as Quantum Chromodynamics. This approach has a remarkable success in understanding and testing the strong force starting from the fundamental degrees of freedom, namely quarks and gluons. It is fair to say that this is not possible using other approaches known at the moment. Areas of success of Lattice QCD include computing the spectrum of hadrons, precision tests of the parameters of the standard model of particle physics, studying hadronic matter under extreme conditions such as very high temperature and density (relevant for the physics of the early universe), and physics beyond the standard model. Lattice QCD simulations are entering a new phase: the age of simulations at the physical point. What this means is that we are now able to simulate quarks with masses at or very close to their physical values and use small lattice spacing and large volumes to reduce discretisation and finite size effects. This has mainly been due to the availability of powerful new generations of supercomputers as well as great advances in the simulation algorithms. PLQCD refers to a collection of coding efforts under WP8 of PRACE for lattice QCD. The main motivation for this effort was to choose certain components of common community codes and improve their scaling and performance aspects in collaboration with the community. The code components selected represent tasks that are used most frequently in simulation codes that also consume the major portion of computing resources. Namely, we chose to work on the "hopping matrix" of the lattice Dirac operator, the iterative linear solver, Fourier accelerated Landau Gauge Fixing, and tuning of the Hybrid Monte Carlo Integrator. There are quite a few lattice formulations with different degrees of complexity as well as lattice codes used by the larger community. In this work, we focus on two widely used codes: the tmLQCD codes for the European Twisted-Mass Collaboration and the Chroma software suite by the USQCD collaboration. Both are publically available.

The PLQCD code has four main components:

- Implementation of the hopping matrix of the Wilson Dirac operator in which parallelisation is done using a hybrid MPI+OpenMP approach.
- Efficient linear solver based on the Incremental EigCG algorithm using the methods of deflation.
- Fourier accelerated Landau gauge fixing.
- Tuned integrator for the Hybrid Monte Carlo simulation algorithm.

The goal in WP8 was to focus on these aspects since they represent critical parts of the lattice simulation codes. Firstly we consider the "hopping matrix" of the Wilson Dirac operator. This represents the main kernel of all lattice calculations. This operator is called tens or even hundreds of thousands of times during the course of a lattice simulation. It is a memory bound operator with arithmetic intensity of about 1.4 and involves nearest neighbour interaction terms, thus requiring communication between nearest neighbouring MPI processes. Until recently, MPI was the main tool of communication used in lattice codes implementing the hopping matrix. It is expected that as we go to a large number of cores (tens or hundreds of thousands) MPI will lose its linear scaling feature because of the large cost of communication overhead. Since modern computer architectures have many cores on a single node that share memory in a UMA or NUMA fashion, one solution for the scalability of MPI could be the

hybrid approach, where cores on the same node use OpenMP for parallelisation while communication among nodes is done with MPI. Another expected advantage of this approach is the smaller surface to volume ratio of the local lattice on a single node leading to a smaller amount of data to be communicated than in the situation where every core has its own lattice with a larger surface to volume ratio. An additional feature of this approach is the reduced memory footprint of the hopping matrix because of the reduced communication buffer, thus allowing larger systems to be simulated on the same number of cores. In addition to implementing this hybrid parallelisation we also looked at other performance improvement techniques. The first improvement aspect was overlapping communications and computations by dividing the sites into bulk sites plus boundary sites and working on the bulk sites first while the communications were being done using non-blocking MPI send/receive then perform the calculations on the boundary sites. The second improvement aspect was the use of the recently available AVX instructions that allows SIMD on a quad of double precision numbers. Other improvement aspects include the use of compiler intrinsics instead of inline assembly, and using a compact representation of the link variables to reduce the memory traffic. In addition, we have tested compression algorithms for MPI. The FPC compression algorithm for MPI can speed the communication by compressing the data to be sent and decompressing them when received. The efficiency of these algorithms is high when there are some patterns in the data to be communicated.

The second part of PLQCD is to provide an implementation of one or more of the efficient iterative linear solvers used in lattice computations. This is an essential part of lattice codes that is needed to compute quark propagators on a background gluon field. This is the part where the Dirac operator gets called thousands of times. As we approach simulations at the physical point with small quark masses, standard iterative solvers such as Conjugate-Gradient converge very slowly. This phenomenon is known as critical slowing down. Recently, there has been a lot of progress in building efficient linear solvers. These include deflated Conjugate-Gradient and algebraic multi-grid methods. There is a large variety of new algorithms that are much faster than standard iterative methods. In this work we focus on two solvers, Incremental EigCG algorithm (for Hermitian Positive Definite Systems) and GMRES with deflated restarting. One reason for selecting these algorithms is also the fact that they work for Twisted-Mass fermions. Since tmLQCD is one of the codes that we focus on, we chose these algorithms to speed-up computations in tmLQCD.

The third part of PLQCD focuses on improving the parallelisation of the Fourier-Accelerated Landau Gauge Fixing. On the lattice, Landau gauge fixing is performed using a local optimisation method, such as Steepest Descent. Such methods suffer from critical slowing down: the number of iterations needed to solve the problem increase with V^z, where V is the size of the problem (in our case, the lattice volume). For gauge fixing, naïve procedures have an associated critical exponent z around 2. A Fourier-accelerated method has been proposed (with z~0), but it requires the use of Fast Fourier Transforms (FFT), whose parallelisation is far from being trivial. The standard and well-known FFTW package provides parallel routines that only parallelise along one dimension. In this work-package we implement the PFFT parallel FFT package that allows for parallelisation in three dimensions. This work will be done within the Chroma software suite.

The fourth part of PLQCD is tuning and optimisation of the integrator used in the Hybrid Monte Carlo (HMC) simulation code. HMC is the standard algorithm to generate lattice configurations with dynamical quarks. The most time consuming part of the HMC method is the molecular dynamics (MD) step, where we evolve the system using some approximate integrator. In general, the integrator contains free parameters, which can be tuned such that the acceptance rate is maximised, while keeping the step size as large as possible. The best way to do this is using Poisson bracket measurements. Depending on how well tuned the

default integration scheme is, the optimisation of the integrator allows to decrease the number of the inversions of the fermionic matrix needed by HMC. New integrator schemes can be tested (such as force-gradient integrators), hopefully providing further improvements. This work is done within Chroma software suite.

In WP8, the following has been achieved:

- Implementation of the hopping matrix for the Wilson-Dirac operator with MPI+OpenMP for parallelisation.
- Overlap of communication and computation in the hopping matrix.
- Using compiler intrinsics for SIMD parts of the hopping matrix.
- Developing a version of the SIMD parts of the hopping matrix using AVX instructions.
- Testing the FPC compression algorithm for MPI.
- Implementation of the Incremental EigCG solver within tmLQCD.
- Implementation of the GMRES-DR solver for tmLQCD (some work is still needed here that is expected to be done before the end of August 2013).
- Implementation of the Fourier accelerated Landau Gauge Fixing using the PFFT package within Chroma software suite.
- Tuning of the integrator used in HMC using Poisson brackets.

6.1.2 Results

In this section, we give some of the results obtained in WP8. A more detailed set of results will also be given on the project webpage hosted on www.hpcforge.org.

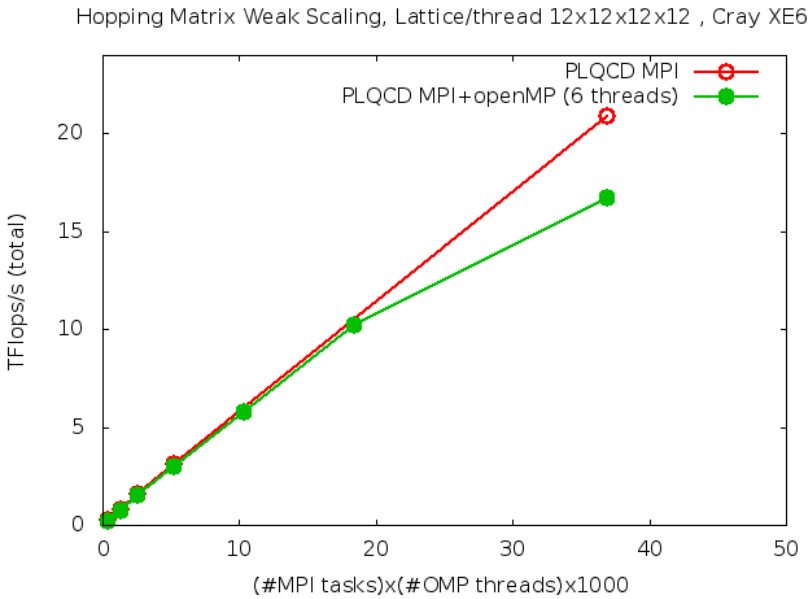

Figure 28: Weak scaling of the hopping matrix on a Cray XE6

In Figure 28, we show a weak scaling test of the hopping matrix performed on a Cray XE6 machine. The machine has 24 cores per compute node such that every 6 cores share the same memory. Thus, the most efficient use of OpenMP would be with 6 threads per MPI process. The results in this plot, as well as other tests, indicated that the hybrid approach gives a lower performance at high number of cores as compared to pure MPI. A possible explanation for this would be that MPI is less effective in sending large size messages than sending multiple small messages when we reach a large number of cores. Since this is a weak scaling test, the amount of data communicated is the same in the hybrid and the pure MPI cases. The

difference is that in the hybrid case, the number of messages is smaller but with a larger size. In this particular case, the message is 6 times larger in the hybrid case and there are 6 times fewer messages to be communicated.

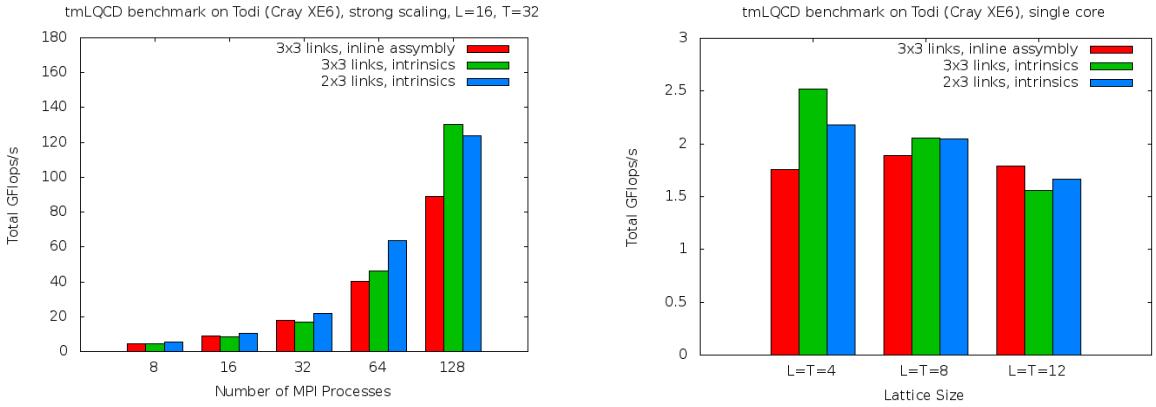

Figure 29: Comparison of the effect of using compiler intrinsics and reduced representation of the gauge links. Left: single core. Right: Strong scaling on multiple cores for a lattice with size L=16, T=32.

In Figure 29, we show the results from using compiler intrinsics for vectorisation instead of inline assembly. We show results on a single core as well as a strong scaling on many cores. The results indicate that intrinsics, besides being easier to code and debug, allow for a better optimisation by the compiler. This is especially true for small volumes. We also show the effect of using a reduced representation of the SU(3) matrices. These matrices are unitary which means that the third row can be reconstructed from the first two rows. This should reduce the memory traffic at the expense of extra floating point operations. Since the hopping matrix is memory bound, it is expected that this should improve the performance. The results indicate that using this reduced representation lead to a better performance. The gain however, is dependent on the volume. In the plot, these are labelled as 2x3 links.

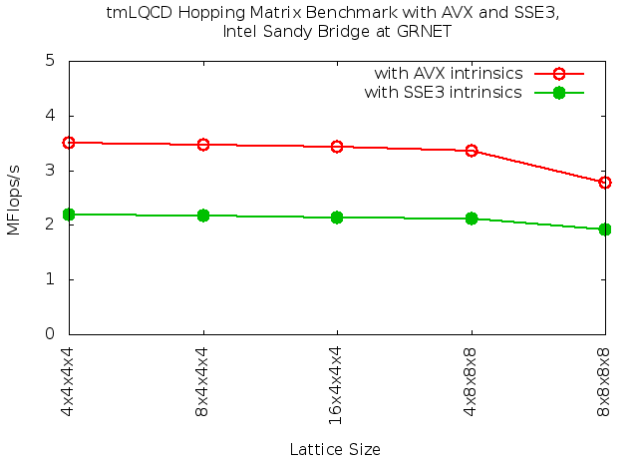

Figure 30: Comparison of performance with AVX instructions to SSE3

In Figure 30, we show the effect of using AVX instructions. A gain of about 1.6 enhancement is observed over different lattice volumes. In Figure 31, we show results from using the Incremental EigCG algorithm on a twisted-mass configuration. These configurations correspond to a large lattice with L=48, T=96 at a small lattice spacing and light quarks corresponding to a pion mass of 230 MeV. The ensemble has dynamical up, down, strange

and charm quarks. In this test 300 total eigenvectors corresponding to the smallest eigenvalues were deflated. The solution time for each of the successive 36 right hand sides is shown. This time include any overhead associated with deflation. The time is compared to a standard CG solver. As seen in the figure, a considerable speed up in solution time is observed. In addition to the Incremental EigCG algorithm, another linear solver that is under development is the GMRES-DR(m,k) algorithm (Generalised Minimal Residual with Deflated Restarting). This algorithm has a couple of advantages over Incremental EigCG. First it is designed for the non-Hermitian linear systems while CG and Incremental EigCG works for Hermitan positive definite systems. Given that the equations in Lattice QCD are non-Hermitian, there is no need to make them Hermitian by solving the normal equations as in Incremental EigCG. The second advantage is that the eigenvectors needed for deflation are computed from the first linear system instead of accumulating them incrementally as in the Incremental EigCG case. This means that the benefit of deflation is obtained starting from the second linear system. Finally, with GMRES-DR we can deflate BiCGStab. It is well known that BiCGStab, although an efficient algorithm for other lattice fermions such as Wilson clover fermions, fails to converge for Twisted-Mass fermions. In Figure 32, we show a sample result of this on a sample ETMC configuration. The first linear system is solved with GMRES-DR(50,30) or GMRES-DR(70,50). The next right-hand side is solved using Deflated BiCGStab (D-BiCGStab) or GMRES-Proj. We also note that BiCGStab did not converge in this case.

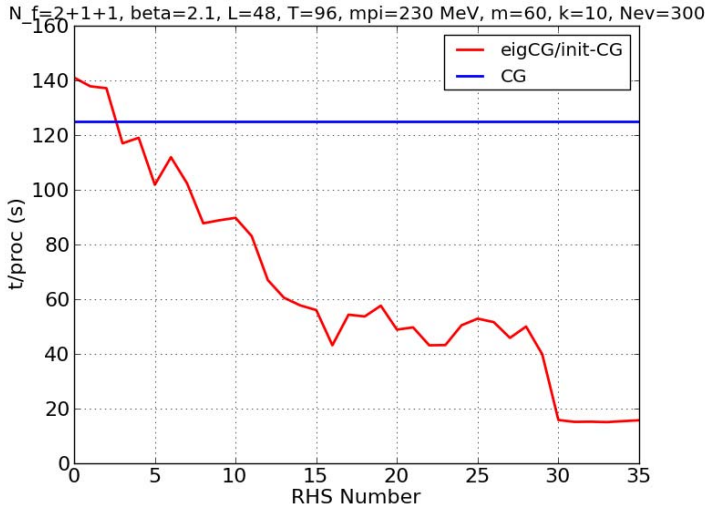

Figure 31: Time for solution of 36 linear systems with Incremental EigCG algorithm on a sample configuration of twisted-mass fermions with 2+1+1 dynamical flavours and L=48, T=96 lattice at beta=2.1.

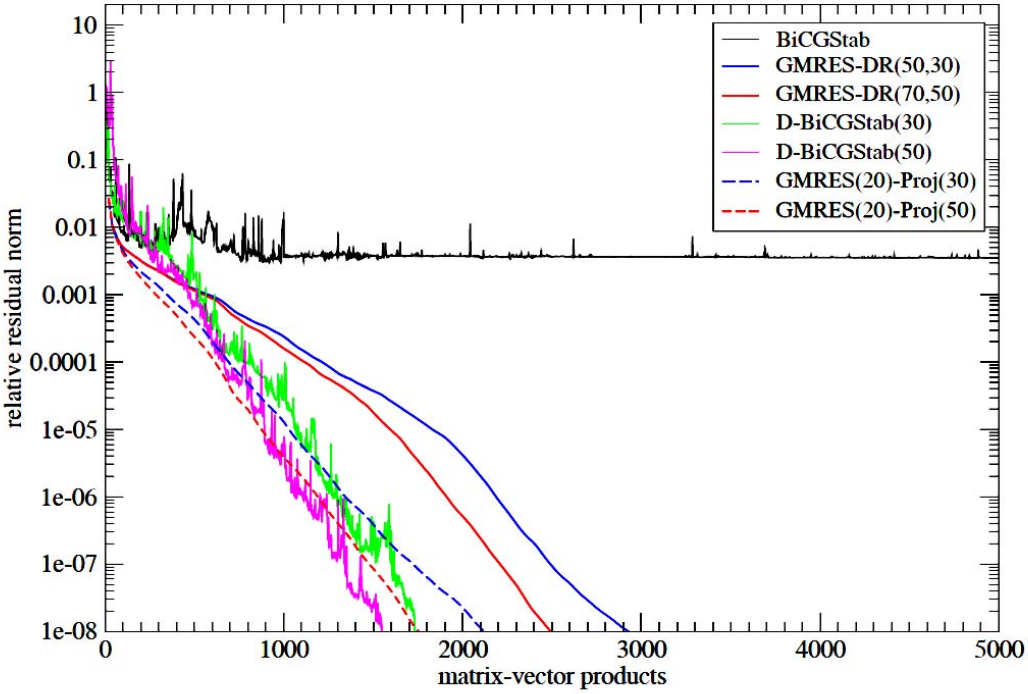

Figure 32: GMRES-DR linear solver for a configuration from the ETMC collaboration with 2 dynamical flavours on a lattice with L=24, T=48, and mass parameters kappa=0.160859, mu=0.004. First right-hand side is solved using GMRES-DR(m,k) and next right-hand side is solved using Deflated BiCGStab or GMRES-Proj(m-k,k).

As mentioned earlier, we looked at the efficiency of using compression algorithms for MPI. We tested the FPC algorithm that is used with floating point data. It turned out however that these algorithms are not efficient for lattice data since they tend to be very random and compression is not efficient in this case.

Next we show results for the use of the PFFT library to accelerate the Fast Fourier Transforms used in the Landau Gauge fixing. In Figure 33, we show a scaling plot of the codes with this library. As can be seen, a perfect scaling can be achieved.

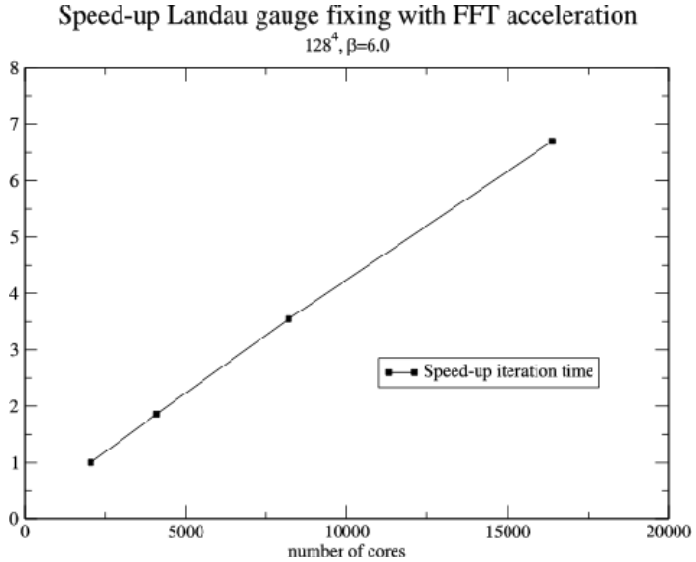

Figure 33: Speed-up Landau gauge fixing with FFT acceleration.

Finally, code has been developed for using a tuned integrator in the HMC algorithm. A sample result for the effect of this optimisation is shown in Table 7.

Integrator Scheme	Predicted Acc. rate	Time (sec)	Measured Acc. Rate	Cost
PQ4	88%	385	85%	452
PQ4	77%	389	81%	479
PQ3	84%	404	92%	441
PQ3	**74%**	**394**	**62%**	**638**

Table 7: effect of using optimised integrator parameter λ on the HMC integrator. PQ3 and PQ4 are two integrator schemes. Test is done on a lattice L=T=36 , β=5.6, κ=0.1575 Wilson quarks. Trajectory length=1 and the Cost is measured in terms of trajectory CPU time/acceptance rate

6.1.3 Re-introduction status

For the hopping matrix, a stand-alone library is provided which can be called by other codes. The library has a simple interface and it is hoped that it will be beneficial to the lattice community. The linear solvers are implemented directly in the tmLQCD software suite. Currently Incremental EigCG is fully implemented and tested. It has been used by members of the ETMC collaboration speeding-up many computations. For example, it has been used in three-point function calculations with multiple sink locations for the nucleon form-factor calculations. The Landau gauge fixing code is a stand-alone code that links against the Chroma library. This is because the code needs a specific layout provided by Chroma library to optimise the interface between Chroma and the parallel FFT library (PFFT). These codes will be very useful since now the FFT is parallelised in three dimensions as opposed to parallelising in a single dimension using FFTW. The code for Poisson bracket measurements is a modified version of the default Chroma library. Work will be done in the future to include Poisson bracket measurements for more lattice actions, and to integrate the code into the default Chroma library. Poisson bracket measurements will be very useful to the lattice community in order to reduce the computational cost of dynamical ensemble generation.

7 Engineering

The Engineering domain focused on several computational challenges, namely parallel mesh refinement, parallel mesh generation and solvers scalability, developing solutions for various codes, such as Elmer, Code_Saturne, Alya, ZFS and a custom Finite Volume solver. The accomplished work and the main results are summarised. The re-integration status is presented.

7.1 ELMER

7.1.1 Overview

Elmer is an open source multiphysical simulation software suite developed by CSC - IT Center for Science in Helsinki, Finland. Elmer includes physical models of fluid dynamics, structural mechanics, electromagnetics, heat transfer, acoustics, etc. These are described by PDEs, which Elmer solves by the FEM. Currently Elmer has more than 5000 worldwide users.

Elmer has shown excellent scaling on appropriate problems up to thousands of cores. Elmer's developers focused on implementation of more robust solvers, which enable further scaling of Elmer. The standalone tool ElmerSolver has implemented several types of solvers: time integration schemes for the first and second order equations, solution methods for eigenvalue problems, direct linear system solvers (Lapack & Umfpack), iterative Krylov subspace solvers for linear systems (GMRES, CG), multigrid solvers (GMG and AMG) for some basic equations, ILU preconditioning of linear systems, the discontinuous Galerkin method. Recently the Elmer code was extended by new FETI1 and TFETI domain decompositions.

Scalability of Elmer's internal solvers

For the comparison of Elmer's parallel solvers of large linear systems (scalar diffusion and linear elasticity problems), there are several methods and preconditioners available internally in Elmer and via interfaces to open-source libraries. After a preliminary selection, a candidate group of about 25 different combinations of solvers and preconditioners was selected (CG, BiCGStab, BiCGStab(l), FETI, multigrid and algebraic multigrid solvers from Elmer, Hypre and Trilinos packages; for preconditioners ILU0, ILU1, Parasails, multigrid and algebraic multigrid from Elmer, Hypre and Trilinos packages). Krylov methods without any preconditioning were also tried and they turned out to be surprisingly competitive in terms of wall-clock time. The benchmarks were computed on the Cray XC30 system Sisu.

Elmer recently incorporated an internal FETI solver which was implemented within WP8 by cooperation between CSC and VSB, and which can be used without any external library dependencies. There is also a possibility to use MUMPS as a solver to factorise the internal domains and to solve the local nullspaces algebraically, and to use MUMPS with several cores to solve the coarse problem. The resulting scalability graphs are shown in Figure 34 and Figure 35. More detailed report of numerical experiments and comparisons can be found at WP8 wiki pages.

FLLOP library for Elmer

FLLOP (FETI Light Layer On top of PETSc, [71]) is a novel software package developed at IT4Innovations at VSB-Technical University of Ostrava, Czech Republic, for solution of constrained quadratic programming problems (QP). It is an extension of the PETSc framework. PETSc is a suite of data structures and routines for the parallel solution of scientific applications modelled by PDE. FLLOP is carefully designed to be user-friendly while remaining efficient and targeted to HPC.

The typical workflow looks like this: a natural specification of the QP by the user, a user-specified series of QP transformations, an automatic or manual choice of a sensible solver, the solution of the most derived QPs by the chosen solver, and a series of backward transformations to get a solution of the original QP.

Additionally, any combination of equality, inequality and box constraints can be specified.. A QP transformation derives a new QP from the given QP. They allow use of efficient solvers but are themselves solver-neutral. Currently, these are: dualisation, homogenisation of the equality constraints, and enforcing of the equality using penalty or projector onto the kernel.

For the comparison of FLLOP parallel solvers of large linear system (linear elasticity of model cube and of engineering car engine), there are several solvers (conjugate gradient method, deflated conjugate gradient method, domain decomposition method, FETI type method). Numerical experiments (model elastic cube and engineering car engine) were computed on the Cray XE6 system HECToR and can be found on the WP8 wiki pages.

7.1.2 Results

Since the FETI methods implemented in FLLOP performed better than those implemented internally in Elmer, the Elmer-FLLOP interface was created. This array based interface is close to procedural programming, in that it uses no PETSc classes so that the libraries that do not know anything about PETSc can use FLLOP.

Interfacing FLLOP to Elmer improves scalability, performance and robustness of Elmer solvers**,** especially due to the FETI coarse problem solution by SuperLU_DIST running in parallel in subcommunicators (see Figure 34 and Figure 35). It leads to significantly better scalability for the coarse problem solution than MUMPS used internally by Elmer in one subcommunicator. Furthermore, there are on-going development of other ingredients improving the performance of FETI. For the numerical experiments the cube benchmark was chosen. On the WP8 wiki pages more realistic engineering problems can be found. The system used for these computations was Sisu (Cray XC30) at CSC.

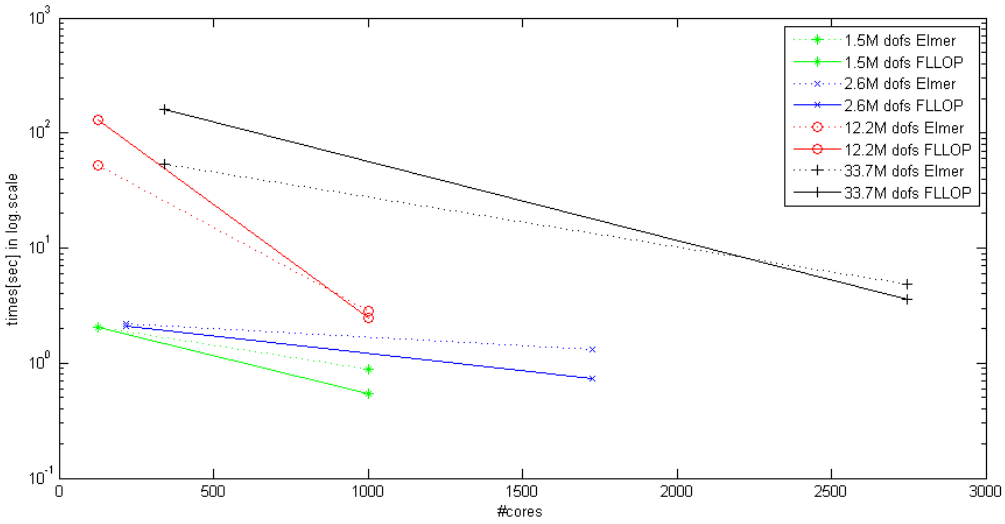

Figure 34: Strong scalability - Elmer-FETI/FLLOP-FETI

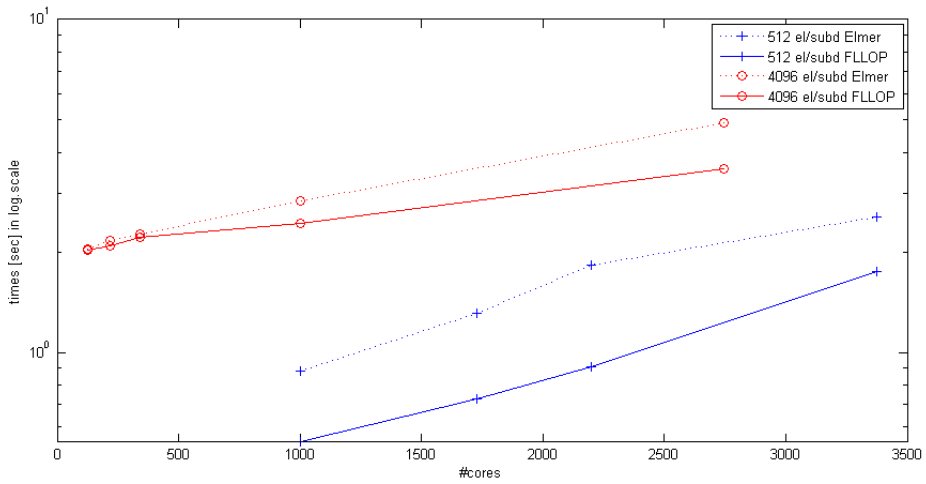

Figure 35: Weak scalability - Elmer-FETI/FLLOP-FETI

7.1.3 Re-introduction status

The Elmer is an open-source multiphysical simulation software developed at CSC. Most problems described by PDEs and solved by the FEM result into large systems of linear equations. By means of systematic testing in the "Code performance analysis" phase, the FETI type solvers were identified as the most efficient tool for their solution. Because of the effort to improve the scalability of the Elmer FETI solvers and the long term experience of VSB-TUO researchers with FETI methods developed and implemented in their FLLOP library, synergy was achieved - the Elmer-FLLOP interface was implemented in the phase "Code refactoring". To apply more efficient FETI solvers, Elmer users have to convert Elmer matrices and vectors into standard arrays, include "fllopaif.h" and then FLLOP solvers can be called immediately from Elmer. The final contribution is not only the scalability improvement of Elmer, but also functionality extension of Elmer via FLLOP enabling an efficient parallel solution of contact problems and other equality, inequality and box contrained QPs. The efficiency of the refactored code was first tested by teams of developers on the model cube linear elastic benchmark in the phase "Prototypes experimentation" on Sisu and then on several engineering problems in the phase "Code validation". The possibility to solve larger, more complicated problems faster will be certainly fully appreciated by Elmer users in the near future.

7.2 Code_Saturne

7.1.1 Overview

Code_Saturne is a multi-purpose CFD software developed by EDF since 1997 and open-source since 2007. The Finite-Volume Method is used to discretise the equations on meshes made of any type of cells. Code_Saturne is distributed under GNU GPL licensing and written in C, Fortran90 and python. MPI is used for communications and OpenMP features have recently been added to the software. Code_Saturne is highly portable and is one of the two CFD codes selected for PRACE 2-IP benchmark suite. The code can be downloaded from http://research.edf.com/research-and-the-scientific-community/software/code-saturne/download-code-saturne-80059.html and a general documentation is available from http://en.wikipedia.org/wiki/Code_Saturne where other links are also available. This information is displayed on the main page of the Wiki.

The goal of this engineering workpackage is to improve performance of codes focusing on mesh generation and linear solvers. For Code_Saturne, four different tasks are dealt with within WP8:

- mesh multiplication (MM) to efficiently generate extremely large meshes (VSB),
- code testing at scale (pushing the limit of the algorithms used in the code) (STFC),
- testing the effect of the order of the Neumann Preconditioner on the resolution of the pressure Poisson equation and then implementing a Chebyshev Preconditioner and testing its performance in Code Saturne (AUTH),
- implementing a deflated conjugate gradient to solve the pressure Poisson equation (STFC).

The main achievements in WP8 are:

- an efficient MM has been developed for hexa-, tetra-hedral, prismatic and pyramidal cells,
- the main algorithms of the code have been tested at scale (1 time-step has been simulated for a 105 billion cell mesh and a speed-up of about 1.5 has been observed going from 0.25M to 0.5M processors),
- a Chebyshev Preconditioner has been implemented, tested and compared to the existing Neumann Preconditioner,
- progress has been made in the implementation of the deflated conjugate gradient, though this has been delayed by the time required by the extensive tests at scale and by the size and complexity of the code.

7.3.2 Results

VSB has been working on a Mesh Multiplication capability, which has been developed, integrated in Code_Saturne and tested for hexahedral cell meshes up to 105 billion cells. All the tests, carried out on different machines (Cray, IBM Blue Gene/Qs and POWER7 cluster) have shown that generating the mesh does not take more than 1 minute. The best achievement was observed on MIRA, Argonne's Blue Gene/Q, where less than 9s were required to generate a 105B cell mesh from a 25M cell mesh on 0.25M and 0.5M processors. MM is now available for other type of cells (tetras, prisms and pyramids) and first tests carried out for tetras on a POWER7 cluster show that it takes only about 15s to generate a 470M cell mesh from a 0.9M cell mesh, on 96 processors.

The code has been ported onto STFC Daresbury Laboratory's (Blue Joule), Jülich's (JUQUEEN) and Argonne's (MIRA) Blue Gene/Qs and the performance of the main algorithms have been tested, with a special focus on the existing linear solvers. The current solver used for the pressure relies on an algebraic multigrid approach. The largest test has been run at Argonne for one time-step and a 105 billion cell mesh on 262,144 and 524,288 processors. A good speed-up of about 1.5 has been observed. Note that during this testing exercise on several Blue Gene/Qs, some issues concerning MPI_Allreduce have been observed with the current driver V1R2M0. MPI_Allreduce operations were about 4 to 10 times slower than with the older driver V1R1M2. This was reported to IBM, who recently fixed the problem. The performance observed for the 105 billion cell mesh was obtained with the non-fixed version of MPI_Allreduce. Figure 36 presents the performance of Code_Saturne on Blue Joule for meshes of different sizes.

In order to speed-up the pressure resolution, several preconditioners have been tested on two CFD cases, the former to simulate a flow in a bundle of square tubes and the latter to study thermal fatigue in a pipe. The Neumann Preconditioners were already implemented in the code and the Chebychev Preconditioners have been implemented there, using the power

method to get the dominant eigenvalue of the pressure matrix. First results show that for both types of preconditioners, the number of iterations reduces as the polynomial order increases, but also that more time is required for the whole calculation to complete, as more matrix-vector multiplications are needed.

Expecting the current multigrid solver not to be able to scale on a large number of processors, a deflated conjugate gradient solver, similar to the one existing in Alya, has been implemented in Code_Saturne. The Lapack library has been linked to the code in order to be used to directly solve the low frequency modes. Testing the whole algorithm is currently underway.

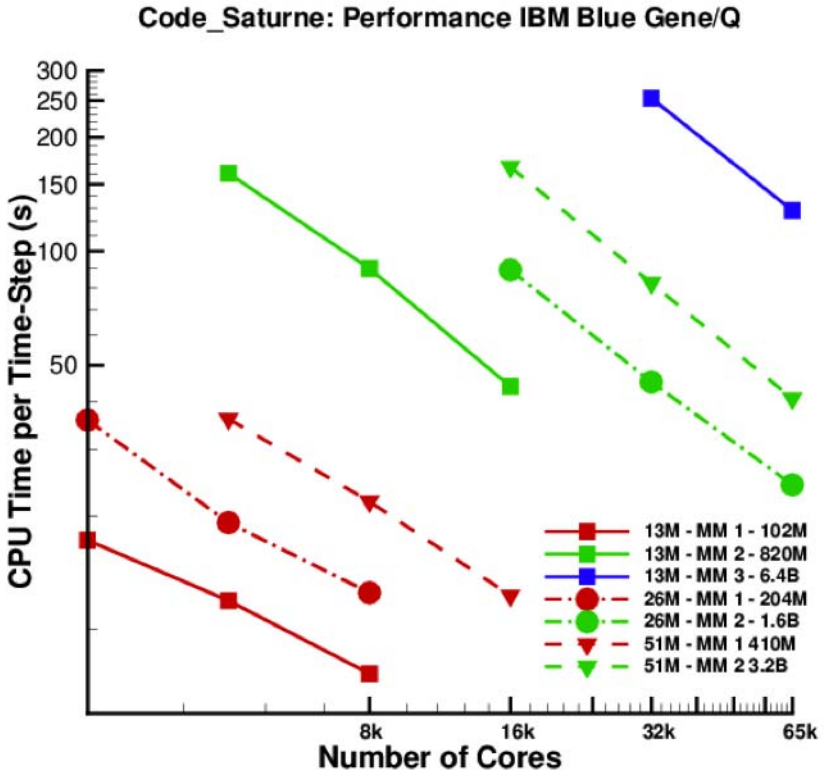

Figure 36: Code_Saturne performance IBM BGQ. Meshes are obtained by Mesh Multiplication.

7.3.3 Re-introduction status

New versions of the community code are released on an approximately 6 month-base, including a fully validated, 'long-term support' released version every 2 years. This latter one is a stable, validated version, declared fit for industrial use in the framework of the EDF R&D Quality Assurance system, which has gone through a 6 to 8 month full validation process. The latest stable version 3.0 has been released at the end of March 2013 and version 4.0 should be released by October 2014.

Due to this process, integration of the mesh multiplication feature, of the Chebyshev preconditioner and of the deflated conjugate gradient solver are not available to all users yet, but this will happen in the next releases of the code. The mesh multiplication (or global refinement) is now the base for developing a new local adaptivity strategy into the code. This new development has currently started at VSB in collaboration with STFC and EDF R&D.

The current work on mesh multiplication will have a large impact on the science delivered by the users as I/O time will be drastically reduced, allowing users to run more detailed simulations (for instance, 10 to 20 minutes are required to load a 812M cell mesh from a

Lustre system, whereas less about 2 minutes are needed for a 100M cell mesh to be loaded and the mesh multiplication to be completed).

Moreover, an introductory course to Code_Saturne has been delivered by STFC at VSB, where, among others, performance of the mesh multiplication algorithm has been presented. This course has been very well received and the code will be shortly installed on Czech Republic new National Supercomputer for any user of their cluster.

Finally, Code_Saturne is going to be used in PRACE 3-IP-WP7.2B, to enable MPI 3.0 one-sided communications, and the tests performed at scale will only be possible thanks to the MM strategy.

7.3 ALYA

7.3.1 Overview

The Alya System is a Computational Mechanics code with two main features. Firstly, it is specially designed for running with the highest efficiency standards in large-scale supercomputing facilities. Secondly, it is capable of solving different physics, each one with its own modelisation characteristics, in a coupled way. Both main features are intimately related, meaning that all complex coupled problems solved by Alya must retain the efficiency. Alya is organised in modules that solve different physical problems. Among the modules included in the code are: incompressible/compressible flows, non-linear solid mechanics, species transport equations, excitable media, thermal flows, N-body collisions, electromagnetics, quantum mechanics, and Lagrangian particles transport. The module involved in WP8 solves for incompressible flows.

The work carried out in WP8 was twofold. On the one hand, the performance of iterative solvers has been studied deeply and enhanced. On the other hand, the team has been developing a surface correction algorithm for the Mesh Multiplication algorithm (MM) implemented in Alya.

The main achievements obtained in the WP are:

Solvers. The team has implemented new solvers and worked on enhancing the performance of an existing solver, the Deflated Conjugate Gradient. Weak scalability tests were performed.

Surface correction for MM. A surface correction has been implemented to correct the shortcomings of the MM when interpolating linearly the surface mesh.

7.3.2 Results

Solvers

The team has implemented a new solver (Schur complement solver for the Poisson equation), new preconditioners (Block LU, one-level multigrid), renumbering techniques (postordering), and studied deeply the performance of an already existing solver in the code, the Deflated Conjugate Gradient. As far as this last topic is concerned, we focused on the following points: a sparse direct solver for the coarse problem; construction of the groups of the coarse problem; “weak” scalability.

- Preconditioners for symmetric/non-symmetric systems
    - Deflation for BICGSTAB/GMRES. The Deflated Conjugate Gradient (DCG) was implemented as a preconditioner to be used for non-symmetric systems, together with Jacobi and Gauss-Seidel smoothing. It was tested for Fluid and Solid mechanics problems. The conclusion is that a certain gain in number of iterations

can be obtained but the reduction in CPU time is low, due to the extra cost involved in the solution of the coarse problem.

- Block LU. The Block LU preconditioner has a more severe behaviour. In current implementation, blocks are directly inverted in each CPU. Obviously, convergence depends greatly on the number of CPU s. However, in the case analyzed, even though the gain in number of iterations is very high, the CPU cost is also much higher than a simple diagonal solver. However, this preconditioner can be useful for ill-conditioned matrices.

- Schur complement solver. A Schur complement solver was implemented for the symmetric problem. The solver was tested against the DCG on 1920 CPUs and performance was found much lower for the Schur complement solver with respect to classical CG.
- Deflated CG
  - Performance: a weak scalability test was performed for a 500M-element mesh, showing the dependence of the convergence and CPU time upon the number of groups.
  - Communication strategies: two different communication strategies were studied for communicating the RHS of the coarse problem. One is based on an AllReduce and the other on an AllGatherv. No noticable difference was found in the scalability of the solver.
  - Strategies for constructing groups: two strategies for constructing the groups were compared: the first one consists of constructing the groups using an advancing front strategy, independently of the subdomain partition; the other one mimics the partition, choosing one group per subdomain. The first option leads to better convergence while the cost per iteration is exactly the same.
- Direct sparse solver
  - Postrenumbering: a post-renumbering strategy was implemented when a sparse direct solver is required. This is the case of block LU preconditioner, Deflated CG and Schur complement solver.

Surface correction for MM

One inconvenience of the MM strategy is that it does not preserve curvature. Indeed, at each multiplication level, new nodes are added on edges and faces (and on centers for hexahedra) by interpolating linearly the coordinates of the previous mesh level. Therefore, a surface correction may be needed. Two options are available, based on geometry reconstruction techniques, or by using a fine STL surface description on which to project each level of the MM algorithm. We considered the latter technique, whose advantage is that it does not require user-based or arbitrary decisions, and is quite automatic. The drawback is that, as it does not recognise geometrical patterns (such as corners), the projection onto the surface can fail, as shown in Figure 37. The technique consists of the following steps:

1) Generate the coarse mesh (from CAD) and a very fine boundary/STL mesh of the geometry surface;
2) Carry out the MM in parallel;
3) On the multiplied mesh, compute the minimum distance d_{min} of the boundary nodes to the very fine STL mesh (using an SKD-Tree technique);
4) Carry out a Laplacian smoothing by using this minimum distance d_{min} as a Dirichlet condition. The Laplacian smoothing conserves the boundary layer by simply

weighting it by the element aspect ratio. The resulting system is solved using the DCG.

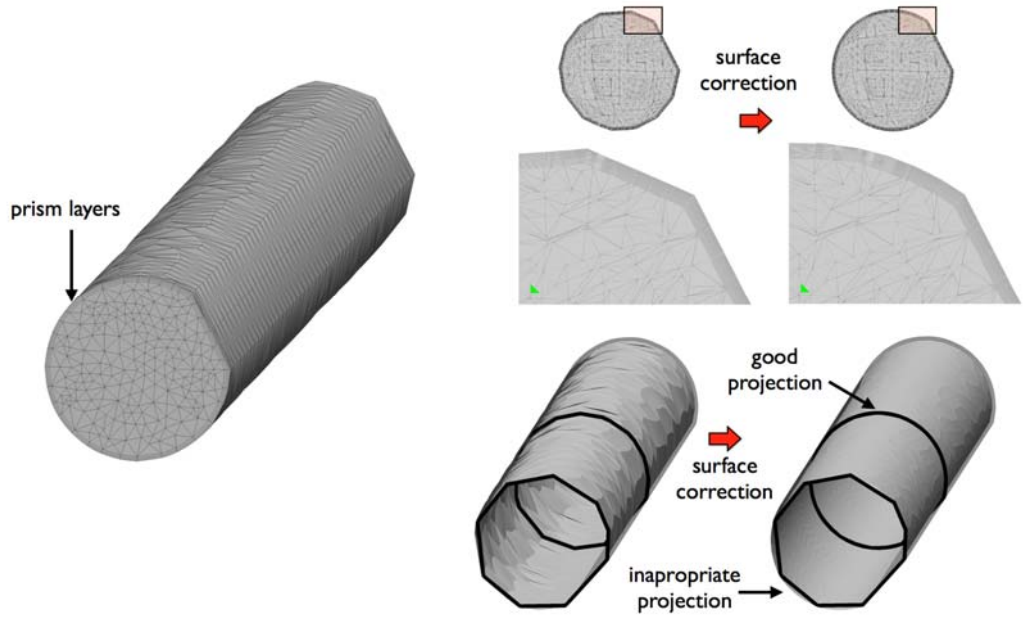

Figure 37: Shortcoming of the surface correction for the mesh multiplication

7.3.3 Re-introduction status

All the algorithms described previously are fully integrated in Alya and available to the users as new options. Some of them are currently used in production and are therefore being extensively tested. All the subroutines have been integrated in Alya sources, without any dependence on external libraries. Some algorithms were coded as Fortran90 modules that can be used by anyone as plug-ins (such as the direct sparse solver). However, due to the close dependence of the iterative solvers with the parallel data structure of Alya, it was found very difficult in general to have all the implementations in the form of modules without totally rewriting the solver library. On the other hand, the developers would be very glad to help any coder who wants to develop the aforementioned algorithms; this was in fact the case of the implementation of the DCG in Code_Saturne.

Some of the newly implemented options (Deflated Conjugate Gradient, together with the mesh multiplication algorithm) have been used in a PRACE Access project, to simulate the airflow in the human large airways. Here are some characteristics of the run:

- Mesh: 44M and 350M hybrid elements (generated in parallel with the mesh multiplication)
- Time integration: 60000 time steps
- Resources: 16000 CPUs on Fermi

Figure 38 shows a snapshot of the results obtained.

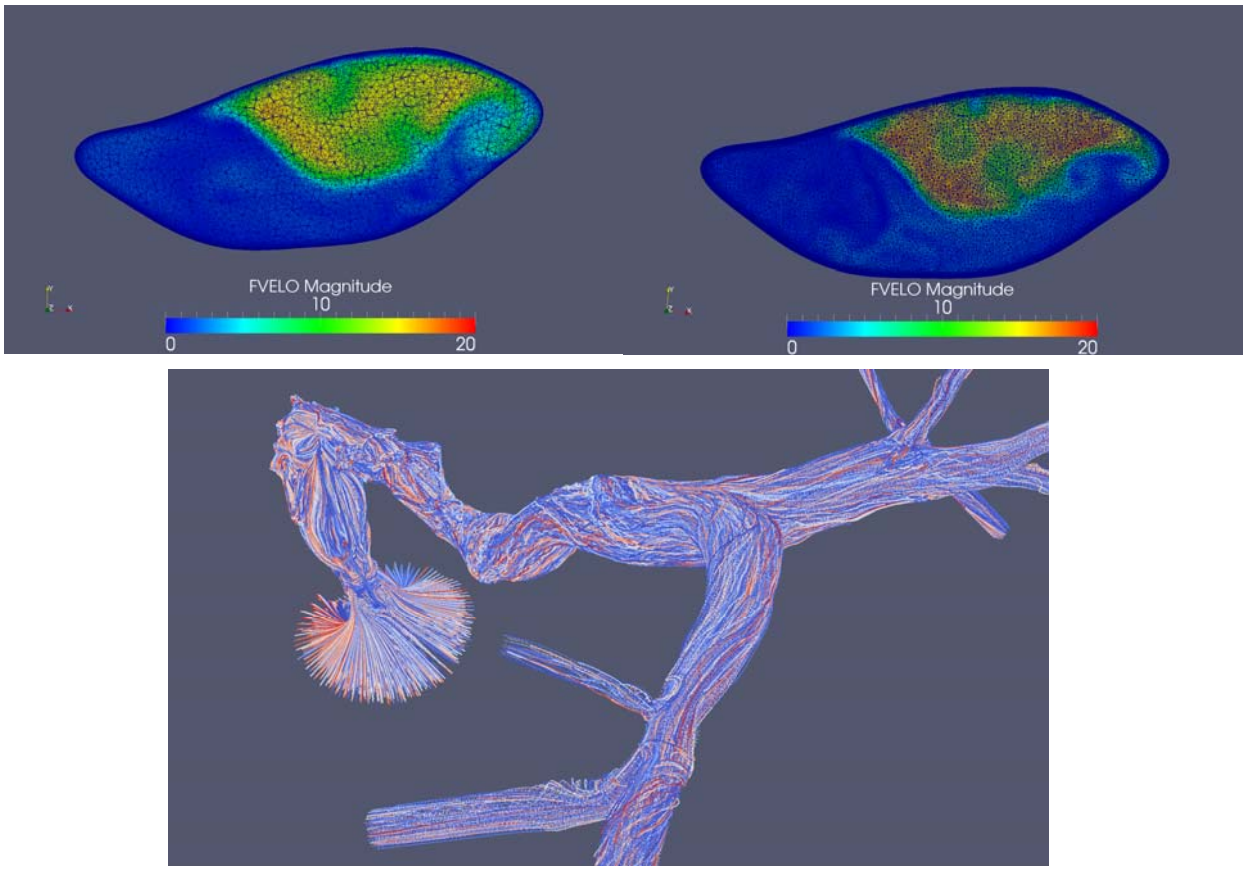

Figure 38: Airflow in human large airways. Top: some velocity contours on coarse and fine meshes. Bottom: some particle paths near the trachea.

7.4 ZFS

7.4.1 Overview

The code ZFS is developed by a group of the Institute of Aerodynamics, RWTH Aachen University. It is a modular flow solver that can use different simulation methods within the same simulation run. The code ZFS is written in C++, following the object-oriented programming paradigm.

The code ZFS can be separated into three main parts:

- Grid generation
- Solving blocks (Finite Volume and Lattice-Boltzmann)
- Output/Post-processing

The structure of ZFS is shown in Figure 39.

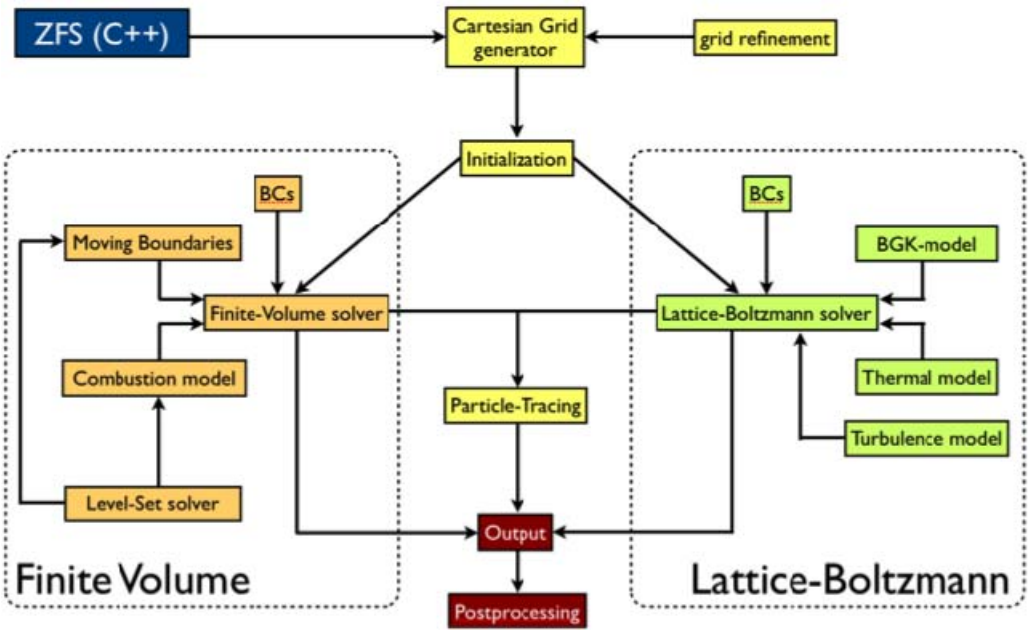

Figure 39: ZFS modules overview

Goals in WP8

The code ZFS was chosen to be part of the engineering community support of WP8 in PRACE-2IP. With the help of the code developers, the topic of parallel mesh generation has been identified as a subject with improvement opportunity.

Parallel mesh generation is used within code ZFS during grid generation and grid refinement. The generation of a Cartesian grid itself is straightforward if the distribution of the level of detail required in a computational region is known. Amongst other things, this distribution depends on the geometry, which defines the computational region by means of boundaries.

This leads to three phases of the Cartesian grid generation:

1. Create cells with high level of detail along the boundaries defined by the geometry.
2. Fill space with cells by relaxing the level of detail between the geometry boundary cells.
3. Determine which cells are inside the computational region and discard the cells outside of it.

The Cartesian grid generation can be done in parallel by simple region decomposition. As long as the full geometry can be held by every distributed memory node, there is no communication needed between nodes within phase one and three of the Cartesian grid generation process.

As problem size increases, the geometry data size may very well exceed the available memory on a node, or at least may be unreasonably large regarding the need to keep the data in memory for mesh refinement during the solution process. In this case it is necessary to distribute geometry data across the nodes. The aim of the engineering support in WP8 of PRACE-2IP is to develop and implement a method for distribution of, and access to, geometry data, which minimises the latency caused by access to distributed geometry data.

Main achievements/results/outcomes

- Development of a caching/paging scheme for the distributed geometry data access within a parallel meshing implementation.

- Two implementations of the caching/paging scheme based on a PGAS strategy for code ZFS. The first implementation uses asynchronous one-sided MPI communication, the second SHMEM (available on Cray systems).

7.4.2 Results

Design choices concerning the paging/caching system

A caching/paging scheme has been developed in order to facilitate the distributed access to geometry data within a parallel meshing application.

It is presumed that geometry data is placed in the distributed memory in a spatial locality aware way. This is ensured within ZFS by ordering the geometry data elements with respect to an Alternating Digital Tree (ADT). If the access pattern of the reading algorithm has also some spatial locality structure, this can be exploited by using a paging system. A paging system organises associated data within pages, so access to a data element locally not available involves the copy of the whole page containing the data element, thus copying also associated data elements in the neighbourhood of the requested data element. Copying several data elements at once saves communication time.

When talking about the access pattern of a reading algorithm, we mean the access pattern to geometry data elements over time. This allows the possibility of keeping a copied page over some time, so the sequential accesses to data elements in the same page can be exploited by the paging system. The temporal memory storing these pages is called a cache memory. Because cache memory is assumed to be rather small, data in cache memory has to be abandoned in order to copy new geometry data into the cache memory. If the reading algorithm also yields some temporal locality behaviour, a smart choice of the pages to be flushed away might be beneficial. The algorithm that decides which page to flush is called a page replacement algorithm. The cache memory and the page replacement algorithm are the main parts of the caching system.

For the ZFS implementation of the paging/caching system, a Clock page replacement algorithm is used. As it is assumed that communication time will be much higher than the application of a page replacement algorithm, the cache was designed to be fully associative. In order to facilitate the search for locally available pages in the cache, a cache mapping function is implemented as a binary tree.

It is required that geometry data has a home base within the memory of a specific computational node and that geometry data is not changed, only read, once it has been generated. Thus geometry data pages are only read from the home base to the cache of a computational node, geometry data is not modified and no pages are written back. There can be several consistent copies of the same data geometry page in different caches. Thus there is no need for a time consuming cache coherency protocol.

Test results

As a test case, the geometry data of the nasal cavity of a scanned proband is used. The geometry data facilitates the simulation of the nasal air flow in order to examine the influence of the nose with regard to a snoring human.

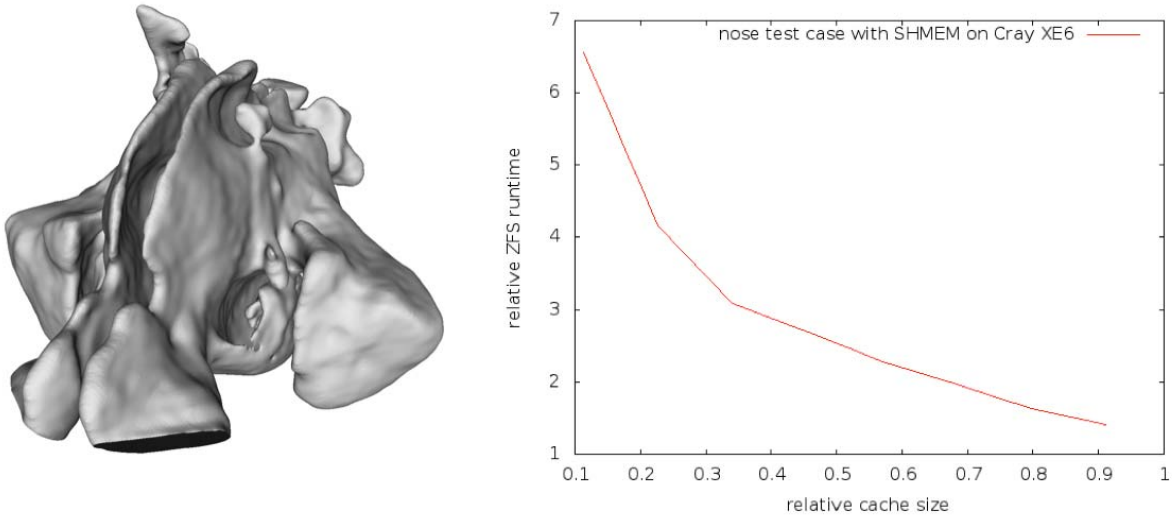

Figure 40: Nose test case. Left: Visualisation of the nasal cavity boundaries. Right: Relative runtime of ZFS plotted against relative cache size.

Test runs are conducted with the caching/paging system implementation integrated into the code ZFS.

The following characteristics for the parallel mesh generation using the paging/caching system are identified:

- Number of simplexes: 877460
- Size of one simplex: 200 Bytes
- Fixed page size of 1000 simplexes
- Test machine: Cray XE6
- Number of cores: 512
- One sided communication library: SHMEM
- Ratio no. simplexes to references: 0.71%
- Overall runtime without geometry data distribution: 80.24s

Measuring on rank 0, we obtain the following run times:

Cache slots	Overall runtime [s]	Relative runtime	Cache hit ratio [%]
100	526.46	6.56	99.968
200	333.34	4.15	99.981
300	247.97	3.09	99.988
400	216.09	2.69	99.99
500	183.62	2.29	99.992
600	157.58	1.96	99.994
700	131.13	1.63	99.996
800	113.00	1.41	99.997
900	89.85	1.12	99.999

Table 8: Relative runtime of the parallel mesh generation within ZFS against the relative cache size

This data can be fitted to an average remote access bandwidth of 17.37 MB/s with an SHMEM implementation of the caching/paging system running on a Cray XE6 system.

Table 8 shows the relative runtime of the parallel mesh generation within ZFS against the relative cache size. The relative runtime is defined with respect to the runtime of the parallel mesh generation, if the cache is big enough to hold all geometry data. The relative cache size is defined with respect to the geometry data size.

In a small cache size window, the runtime decreases considerably with increasing cache size, but this effect weakens with bigger cache sizes. One might interpret the relative cache size of 0.35 in this figure to be a reasonable choice with regard to the effectiveness of the cache.

Parameters

The caching/paging system implementation in ZFS has been parameterised with regard to the

- page size,
- cache size.

ZFS geometry data storage

ZFS organises geometry data in a single instantiation of the C++ container class "ZFSCollector". The "ZFSCollector" class declares an array of geometry elements as well as a contiguous region of memory that is used by the geometry elements to store part of their data. The geometry elements are stored in the order they are read in. A geometry element can be referenced by its index in the geometry element array.

In order to increase spatial search performance within the geometry data, an alternating digital tree (ADT), is set up, which nodes point to their respective element in the ZFSCollector geometry element array by its index. The geometry data is sorted thereafter with respect to the ADT. Figure 41 illustrates the arrangements of the geometry data elements within an ADT tree.

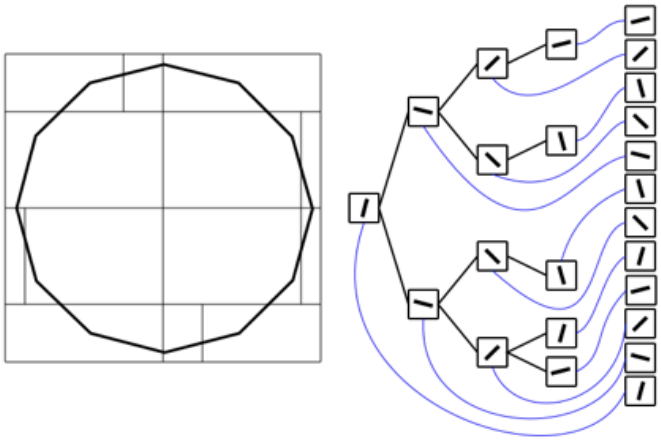

Figure 41: Sketch of the ordering of geometry elements within an ADT tree.

Remote data access

An element is referenced by a global index, which is determined by the order the elements are read in. The "ZFSCollector" class is enhanced to a "ZFSDistributedCollector" class. The access operator of the "ZFSDistributedCollector" class supports the access of remote elements, i.e. elements which are not stored on the local node.

In the current implementation, only simplex data of the geometry data is distributed. The ADT remains as a copy in the memory of every node. The size of the ADT is approximately 24% of the simplex data size.

A first implementation using passive MPI one-sided communication yielded very low communication bandwidth. The following implementation using SHMEM offers acceptable communication speed on a Cray XE6 system.

The UML-Diagram in Figure 42, sketches the relationship between the classes within ZFS:

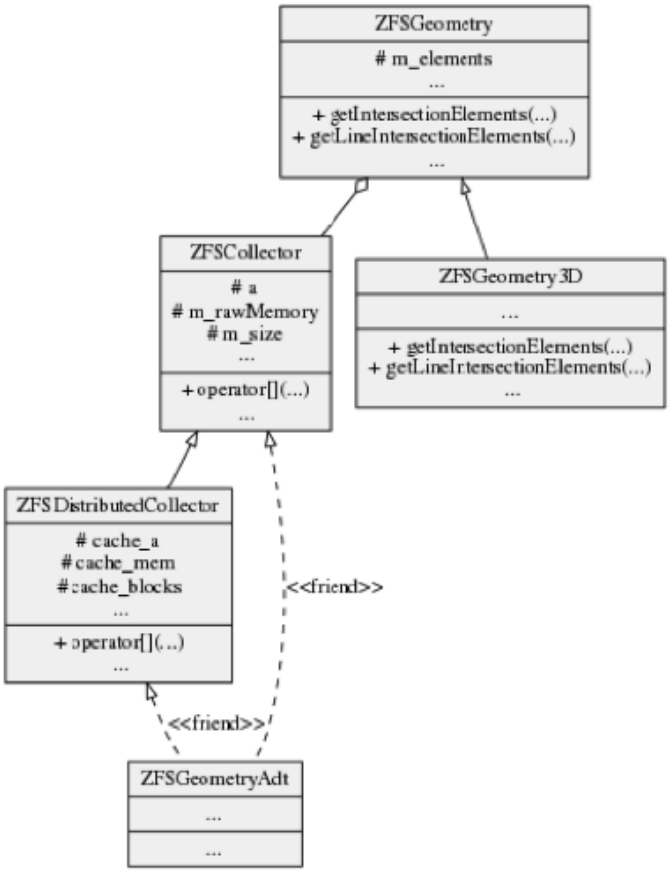

Figure 42: UML diagram sketching the embedding of the paging-/caching-system implemented as the ZFSDistributedCollector class

7.4.3 Re-introduction status

The caching/paging system developed in WP8 has been presented to the ZFS development team and it has been well received.

The implementation work in WP8 has been introduced to ZFS in form of a branch of the ZFS source code trunk path on a SVN repository, hosted at a server of the Institute of Aerodynamics of RWTH University of Aachen. As soon as there is need for geometry data distribution, the code will be validated and eventually merged with the then current trunk of the ZFS source code.

Documentation is available at a wiki page system hosted also at the Institute of Aerodynamics of RWTH University of Aachen, which one is able to access if access to the source code of ZFS has been granted.

7.5 Coupled FVM Solver

7.5.1 Overview

As described in D8.2, the coupled solver method for CFD offers some interesting features which can be exploited to reduce the data transfer between processors and thus to improve the scalability of CFD codes based on the Finite Volume Method (FVM). After testing and validating the coupled solver method we intend to integrate it into a PRACE code.

Main results:

A proof of concept demonstrated by our implementation of the coupled solver for an incompressible, stationary fluid which shows promising strong scaling behaviour.

7.5.2 Results

We implemented the “virtual walls“ and the “long-range corrections“ for an incompressible, stationary fluid (“lid-driven cavity“) from scratch in C/C++ (see D8.2) and successfully demonstrated the usefulness of the coupled solver method.

Studies of the FVM for the time-dependent Navier-Stokes equations indicate that the same methods (“virtual walls“ and “long-range corrections“) can also be applied to transient fluids (“implicit time integration“). Currently, the time-dependent coupled solver is being implemented.

7.5.3 Re-introduction status

Our coupled solver method and first results were presented to the CFD community. The responses showed that it is a valuable approach, which should be further developed. At present the integration of the coupled solver method into community codes would be premature but after the validation of the method it can be integrated into any CFD code based on the FVM.

8 Summary and conclusions

Over the course of two years, WP8 developed an intense programme of code re-design and refactoring in order to provide scientists numerical tools capable of exploiting efficiently and effectively novel HPC architectures and to promote a "culture" of high performance computing awareness among community software developers.

The work involved Astrophysics, Material Science, Climate, Engineering and Particle Physics and resulted in refactoring of eighteen codes. The results led to large improvements in the performance of the different applications. In many cases, WP8 focussed on the reimplementation of large parts of the codes, by the adoption of novel algorithmic solutions, in order to circumvent the performance bottlenecks affecting applications' kernels in terms either of computational speed or memory usage (or both). This is the case of codes such as EAF-PAMR, ELMER, CODE_SATURNE, ALYA and ZFS. This improved the algorithms irrespectively of a specific hardware architecture or programming model.

For many codes, such as PFARM, ABINIT QUANTUM Espresso, SIESTA. Exciting/ELK, PLQCD, CODE_SATURNE and ZFS, the main effort was dedicated to the improvement and optimisation of the parallel MPI implementation of the code, in order to achieve good scalability on a large number of CPUs (usually in the range 10000-100000). This could be obtained both by the refactoring of numerical kernels, in order to enforce data locality and increase the flop-to-byte ratio, and by using advanced MPI features, such as asynchronous communication, dedicated communicators and topologies and special features, like one-side communication/remote memory access (e.g. in the case of ZFS). At the same time, many of these codes adopted also high performance libraries (for example ELPA [32] or PETSC [63]) to improve the scalability of specific kernels.

The refactoring was in many cases dedicated to the implementation of a hybrid version of the code, capable of exploiting multi-core, shared memory CPU architectures, usually by means of the OpenMP programming model, and at the same time to run on multi-node distributed system, adopting the MPI paradigm. RAMSES, EAF-PAMR, NEMO, ICOM, ABINIT, Quantum ESPRESSO and PLQCD focused on this *hybridisation* effort. In most cases (all but one: EAF-PAMR) the MPI code was already available while the OpenMP version had to be implemented and optimised. The hybrid versions allow more efficient use current HPC architectures, leading to an effective use of local memory and to highly scalable codes, able to exploit efficiently computing systems with up to million of cores, and mitigating, at the same time, problems of tight memory limits.

This approach, however, is not suitable to take advantage of novel accelerated processors, where the standard multicore CPU is coupled to a GPU or to a MIC accelerator. Several codes were re-implemented to exploit such accelerators. ELK and SIESTA adopted the CUDA programming model for the porting on the GPU. In the case of RAMSES, the OpenACC [26] based approach was followed, which appeared to be suitable to a pure Fortran 90 code. In other cases, like EAF-PAMR, in order to ensure portability, the OpenCL [25] standard was used. PFARM and SIESTA used the linear algebra MAGMA library [35], while Quantum ESPRESSO used the PhiGEMM [60] library. Both libraries are based on CUDA. Finally ICON experimented with different approaches (OpenCL, CUDA Fortran [27] and OpenACC) in order to adopt the model most suitable to the different code components.

Optimisation on specific computing architectures was also performed on NEMO and PLQCD, for the IBM Blue Gene/Q system [61] and, again, on NEMO, for INTEL Sandybridge [62] processors.

Finally, a specific effort was dedicated to I/O performance optimisation. This was accomplished on specific codes, as for ABINIT and PFARM, and with the development of general-purpose libraries for climate codes.

The validation of the codes turned out to be a day-to-day process, with no need of a special procedure. The developed algorithms in most of the cases were integrated in the official distributions and made immediately available to the community software developers, for further testing and debugging, and to the users, finally, in order to let them experiment the new versions and releases.

The developed codes were introduced to the different communities through presentations at conferences and workshops, journal papers, white papers and the work package web site (http://prace2ip-wp8.hpcforge.org/). Furthermore, a specific face-to-face workshop programme was developed in order to periodically gather HPC experts, software developers and scientists to present WP8 achievements, discuss users' expectations and needs, and introduce novel hardware and software technologies. Furthermore, the prompt public availability of the new kernels facilitated their usage and dissemination.

The achievements and outcomes of WP8 can be considered in many respects extraordinary, proving the success of the adopted methodology. Despite the limited amount of time and resources available for such ambitious objectives, the deep synergy between HPC experts and software developers and the strong commitment of the involved scientific communities led to these outstanding results. Furthermore, this collaboration promoted HPC among community code developers, providing them with the tools and the knowledge to face the challenges posed by the next generations of supercomputers, allowing them to autonomously adopt and exploit the most appropriate software and hardware solutions. For some of the application, opportunities for further enhancements have been identified. Based on the commitment of the codes owners to continue their investment and work in the framework of PRACE-2IP and according to the available effort, a subset of the applications subject to the WP8 programme has been selected for further refactoring work. More specifically, in the Engineering domain, the ELMER code could be further optimized for systems of linear equations with both, symmetric and non-symmetric matrices of various structures and fill-ins, including those arising from FETI-1 and TFETI applications. In Material Science, the Quantum Espresso code could be subject to further refactoring toward the efficient exploitation of accelerated (GPU, MIC) architectures. In Climate, the ICON code would address the full integration of the OpenACC implementation in the main software trunk, while the I/O modules could be further extended and enhanced. For Particle Physics, the PLQCD code could be further enhanced on multicore architectures, while in astrophysics the EAF-PAMR porting to the GPU using the OpenCL programming model could be completed.