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

D8.4.2
Final Refactoring Report

Final

Version: 1.0
Author(s): Claudio Gheller, CSCS
Date: 30/08/2014

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: D8.4.2	
	Deliverable Nature: Report	
	Deliverable Level: PU *	**Contractual Date of Delivery:** 31/08/2014
		Actual Date of Delivery: 31/08/2014
	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: Final Refactoring Report**	
	ID: D8.4.2	
	Version: <1.0>	**Status: *Final***
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D8.4.2.docx	
Authorship	**Written by:**	Claudio Gheller (CSCS)
	Contributors:	Fabio Affinito, CINECA; Alastair McKinstry, Michael Lysaght, ICHEC, Greg Corbett, Andrew Sunderland, Martin Plummer, STFC; Giannis Koutsou, Abdou Abdel-Rehim, Giannos Stylianou, 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, Lubomír Říha VSB; Andrew Porter, Stephen Pickles, STFC; William Sawyer, Anton Kozhevnikov CSCS, Ioannis Liabotis and Nikos Anastopoulos GRNET
	Reviewed by:	Peter Michielse, SURFsara; Thomas Eickermann, FZJ
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	20/06/2014	First skeleton	
0.2	15/07/2014	Introduction and Conclusions added	
0.3	17/07/2014	Best Practice section added	
0.4	18/07/2014	Section 3 improved	
0.5	21/07/2014	Section 2 updated	
0.6	25/07/2014	Section 2 updated	
0.7	30/07/2014	All inputs added	
1.0	02/08/2014	Deliverable completed	

Document Keywords

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

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] Deliverable D8.3: “Re-integration into Community Codes”
[8] Deliverable D8.4.1: “Plan for the further Refactoring of Selected Community Codes”
[9] A G Sunderland, C J Noble, V M Burke and P G Burke, CPC 145 (2002), 311-340.
[10] UKRmol: a low-energy electron- and positron-molecule scattering suite, http://oro.open.ac.uk/33130/.
[11] Intel Math Kernel Library, http://software.intel.com/en-us/intel-mkl.
[12] Matrix Algebra on GPU and Multicore Architectures (MAGMA), http://icl.cs.utk.edu/MAGMA/software/index.html
[13] PFARM Wiki HPCforge, http://hpcforge.org/plugins/mediawiki/wiki/pfarm/index.php/Main_Page.
[14] Intel Trace Analyzer and Collector, http://software.intel.com/en-us/intel-trace-analyzer
[15] ICHEC Fionn Supercomputer, http://www.ichec.ie/infrastructure/fionn.
[16] Eigenvalue SoLvers for Petaflop-Applications, http://elpa.rzg.mpg.de.
[17] CCPforge, http://ccpforge.cse.rl.ac.uk.
[18] Collaborative Computational Project Q - Quantum Dynamics in Atomic, Molecular and Optical Physics, www.ccp2.ac.uk.
[19] UK R-matrix Atomic and Molecular Physics HPC Code Development Project, www.fortran.bcs.org/2012/UKRAMP12_JDG.pdf.

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)
CMOR	Climate Model Output Rewriter
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
HD	Hydro Dynamics
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 Software Project
IFS	Integrated Forecasting System
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
MHD	Magneto Hydro Dynamics
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MIC	Intel Many Integrated Core architecture
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
OCL	OpenCL
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
SSD	Solid-State Drive
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)
Xeon Phi	Processors family by Intel using the MIC accelerator

Executive Summary

This document presents the main achievements of the extension of Work Package 8 'Community Code Scaling', which focuses on the re-design and refactoring of a number of codes for scientific numerical applications, in order to effectively run on coming generations of supercomputing architectures, optimally exploiting their innovative features. The extension was conceived in order to further enhance the results exploiting the developed competencies, skills and synergies. A subset of codes originally in WP8 underwent further refactoring. For each code, the results are summarised (the details are given in the corresponding papers, official documentation and related web sites, in particular the WP8 wiki pages: http://prace2ip-wp8.hpcforge.org). The main outcomes of the overall WP8 work are discussed and best practices for the development of scientific numerical applications on HPC systems are presented.

1 Introduction

During its one-year extension, work package 8 (hereafter WP8) re-design and refactoring programme was continued and finalised, exploiting the working methodology implemented in the first two years of the project. Such methodology was designed to support science by enabling numerical applications and simulation codes for coming generations of High Performance Computing (HPC) architectures. This was based on a close synergy between scientists, code developers and HPC experts, as extensively described in deliverables [4], [5], [6] and [7]. Figure 1 shows how the extension (Task 4) was integrated in WP8 as a natural improvement and completion of the previous work.

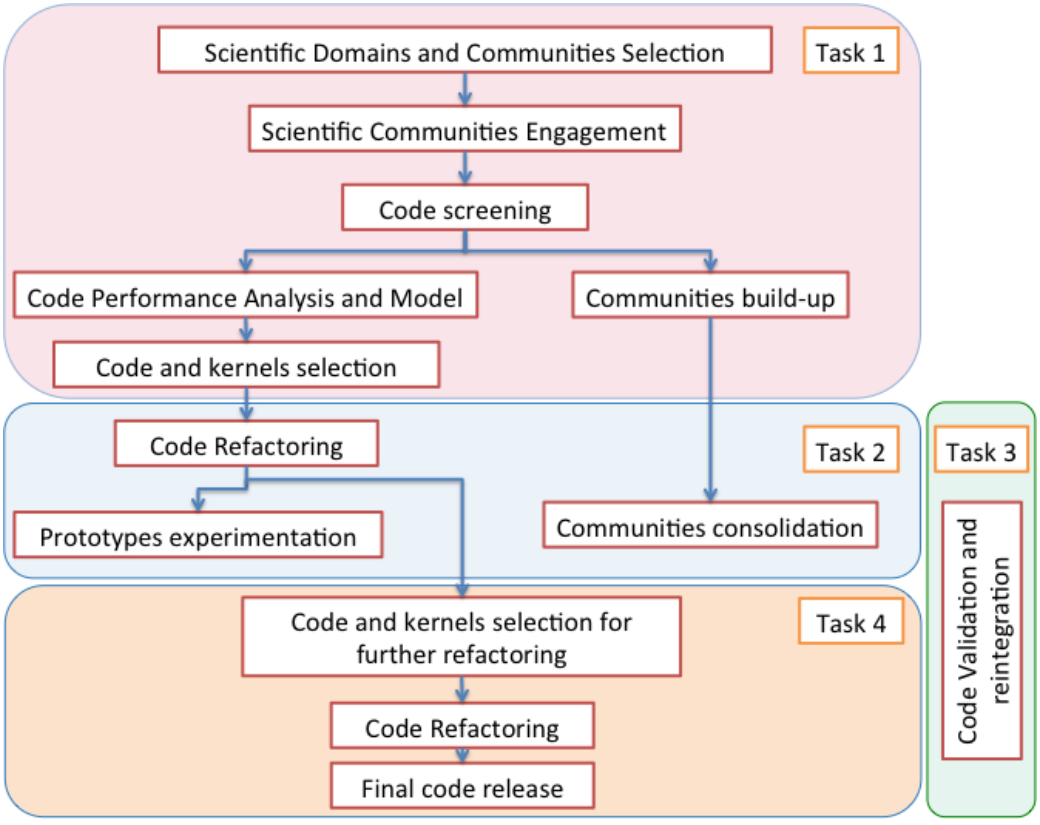

Figure 1: Sketch of the working methodology adopted for WP8 and its one-year extension (Task 4).

During Task 4 the focus was on a number of "success stories," in order to further enhance the quality, the effectiveness and the efficiency of the developed software and its readiness for the new HPC architectures that the PRACE RI will deploy as Tier-0 or Tier-1 systems. The codes have been selected according to the achieved outcomes of the first two years, the proposed objectives, and the available resources. The codes selected for further refactoring are listed in Table 1. The results were presented at the final Face to Face workshop held at CINECA (Bologna) on April 28 and 29, 2014. During the workshop the outcomes of the project were analysed and discussed with the community members and the final steps of WP8 assessed.

In this document, Section 2 will be dedicated to give, for each of the selected codes, a short description of its main features (details can be found in deliverables [2] and [3]), a summary of the work accomplished during the extension and the main achievements. In Section 3, based on the whole WP8 experience, we will provide a number of best practices finalised to facilitate and improve any further work in the field. The deliverable is completed by Section 4 with the summary and the conclusions.

Code name	Scientific Domain	Responsible partner
EAF-PAMR	Astrophysics	UC-LCA
OASIS	Climate	ICHEC
I/O Services	Climate	ICHEC
ICON	Climate	ETH
Fluidity/ICOM	Climate	STFC
Quantum ESPRESSO	Material Science	CINECA
SIESTA	Material Science	BSC
EXCITING/ELK	Material Science	ETH
PLQCD	Particle Physics	CASTORC
ELMER	Engineering	VSB-TUO
ALYA/CODE_SATURNE	Engineering	STFC
PFARM	Astrophysics	STFC
RAMSES	Astrophysics	ETH

Table 1: List of the codes selected for Task 4 (left column), corresponding scientific domain (central column) and responsible PRACE partner (right column).

2 Refactoring Work on Selected Codes and Results

In this Section we summarise the main accomplishments of the refactoring program during the work package extension for each of the codes selected for the WP8's extension (additional details on the work plan can be found in Deliverable 8.4.1 [8]).

2.1 EAF-PAMR

2.1.1 Overview and workplan

The Eborae Astrophysics Fluid-Parallel AMR (EAF-PAMR) Code originally developed by de Avillez, for dynamical studies of the interstellar medium in galaxies, solves the HD and MHD equations in a parallel fashion and uses a hybrid block-based AMR (van der Holst & Keppens 2007) algorithm, where the grids are generated on the fly according to the combined refinement criteria based on density, pressure and vorticity gradients. The code was rewritten to include OpenCL extensions (dubbed EAF-PAMR OCL module developed under PRACE-2IP WP8) and an atomic and molecular plasma emission module tracing the thermal evolution of the plasmas - the EA+MPEC module.

The EAF-OCL module, written in OpenCL, includes OpenMP directives to allow cross-platform calculations using several devices (e.g., GPGPUs) in the same machine. The code is a Piecewise Parabolic Method (PPM) based approximate Riemann solver for the hydrodynamics and magnetic components. PPM increases drastically the accuracy of the simulations regarding other popular algorithms, especially in problems where the evolution of shocks is very important for the dynamics of the problem, e.g., supernova driven shocks and the dynamics of the interstellar medium in star-forming galaxies. A further setup was the coupling of a thermal module into the EAF-OCL code. The thermal module, dubbed E+AMPEC code, deals with the ionisation structure of a plasma, tracing in a time-dependent fashion ionisation, heating and cooling processes resulting from the collective effects of two-particle processes, photoionisation due to an external field, inner-shell photoionisation followed by the ejection of photoelectrons through Auger and Coster-Kronig processes. These photoelectrons redistribute energy into excitation, heating, etc as a secondary process.

The work plan for the extension period comprised:

- Implementation of MHD algorithms in OpenCL - Implementation of the PPM and TVD MHD methods of Dai & Woodward (1998) and Ryu & Jones (1995), respectively. The latter requires the setup of a cleaning scheme for the divergence of the magnetic field.
- Implementation of thermal evolution of plasma and secondary heating modules - Incorporation of the thermal evolution of the plasma, calculated with the EA+MPEC module, into the EAF-OCL code.
- Hybridisation - Implementation and testing of a hybrid OpenCL plus OpenMP/MPI calls to take advantage of multiple devices in the host machine.
- Comparisons between GPU and Xeon Phi Coprocessor.

2.1.2 Results

The main results achieved during the extension period comprise:

- Implementation of an OpenCL version of the MHD PPM based algorithm of Dai & Woodward and of the TVD based method of Ryu & Jones. The latter method is accompanied by a cleaning algorithm for the divergence of the magnetic field. Both methods rely on dimensional splitting.

- OpenCL porting of a specifically developed atomic and molecular plasma emission module (EA+MPECp; “p” stands for PRACE) for thermal and dynamical simulations of astrophysical plasmas.
- Setup of a hybrid OpenCL plus OpenMP (in the host machine) to system to split the jobs into different compute devices in the same host. Tests carried out with the hybrid setup show an increase in speed up of 65% regarding a OpenCL code without any OpenMP directives in the host machine.
- Modifications in the variables declarations, by including them into structures, lead to reductions in computing time by 40%.
- OpenCL granularity has to be adapted according to the problem at use, e.g., dynamical calculations require a work-item granularity, while for the thermal calculations the fine-grained granularity is the most adequate.

2.1.3 Impact and Summary

The major impacts of the work developed in the WP8 comprise:

- Assessment of the advantages in using OpenCL based software to model astrophysical fluids, in particular in terms of acceleration of the simulations using GPGPUs and CoProcessors.
- Assessment on the advantages in spending a long time porting existing software into OpenCL vs. keeping the existing software and running it in the traditional way using CPUs or using directives like OpenACC

Activities done with the Communities:

- Presentation of talks at international workshops and summer schools on the usage of GPGPUs and OpenCL programming, e.g., “Parallel High Performance Computing using Accelerators”, University of Minho, Portugal, 25-27 July 2014.
- Training courses on OpenCL and applications to astrophysics fluid dynamics.
    - Computational Astrophysics Ph.D. program course on OpenMP/MPI and OpenCL programming (6 ECTS; 30 hours), University of Évora, Portugal (May 2014)
    - Publication of papers in Astronomy & Astrophysics and Computer Physics Communications related to:

“EAF-PAMR OpenCL MHD based code” (Computer Physics Communications; Carvalho, Correia, & de Avillez)
“An OpenCL module for MHD simulations of the ISM” (Astronomy & Astrophysics; Correia, & de Avillez)
“The atomic and molecular plasma emission module for HD and MHD simulations of the interstellar medium in galaxies” (Astronomy & Astrophysics; de Avillez)

2.2 Couplers: OASIS

2.2.1 Overview and workplan

OASIS is the most widely used coupler for European climate models, used in 6 of the 7 Earth System Models (ESMs) in ENES to exchange and interpolate coupling fields between individual air, ocean, sea-ice, etc. components. The previous version, OASIS3, was seen to have major limitations due to a “pseudo-parallel” implementation and was the major bottleneck for high-resolution runs. The current work was to optimise and test the new OASIS3-MCT release using the EC-Earth climate model as a test case.

2.2.2 Results

In the extension, previous work on scaling OASIS3-MCT2 which had been delayed due to staffing issues was completed. OASIS3-MCT2 was tested against EC-Earth3 on Hermit at 11,500 cores and profiling and analysis was done. Testing was done using model configurations up to T2047 (10km) on nodes with 32GB and 64GB of memory (previous work had shown problems with certain memory sizes). Linear scaling was seen up to 8440 MPI processes for the largest test case, T2047-ORCA025L46. Analysis shows that linear scaling in MPI processes should continue should larger configurations for the climate models become available (EC-Earth is aiming for O(1km) horizontal resolution). Scalability of EC-Earth3 should also benefit by concurrent work within PRACE on Nemo OpenMP improvements. At tested resolutions, OASIS3-MCT was seen to be < 1% of the total runtime.

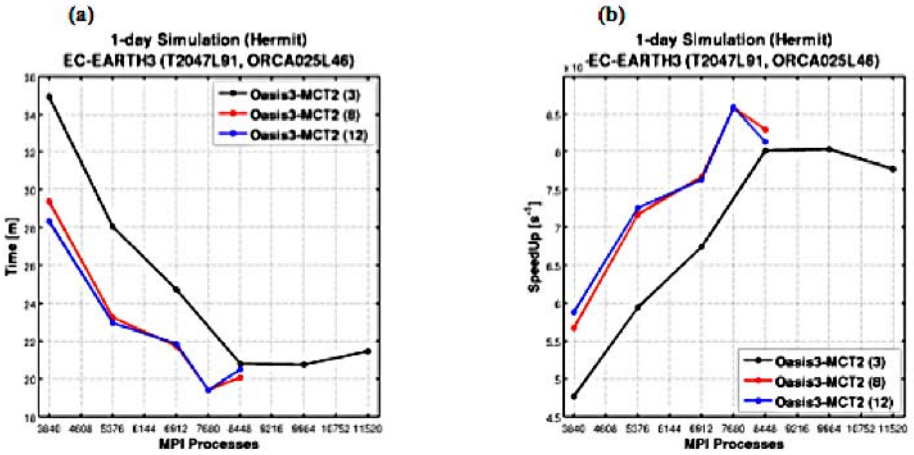

Figure 2: T2047-ORCA025L46 EC-Earth3 Scaling Results on Hermit using OASIS3-MCT2. Results are presented using a total processes per NEMO process ratio of 3, 8 and 12. (a) presents runtime [minutes] and [b] presents speedup.

2.2.3 Impact and Summary

- EC-Earth3 ported to use OASIS3-MCT2
- Bug reports and fixes communicated through the project lifetime to the main OASIS3 developers at CERFACS
- OASIS3-MCT2 code was reintroduced to the community at the dev.ec-earth.org wiki where it is now reintegrated to the mainstream branch for EC-Earth3
- Work presented at EC-Earth conference, Reading UK, in February 2014
- OASIS3 and coupling now no longer a bottleneck in EC-Earth. Climate models expected to scale to > 50,000 cores when all current changes merged; bottleneck moves to I/O and post-processing.

2.3 Input/Output: CDI, XIOS

2.3.1 Overview and workplan

The aim of this work in WP8 was to create a single I/O library for climate models that scaled optimally for I/O writing to exascale sizes (i.e. hardware bandwidth limited), and postprocessing from climate models, based on the XIOS "XML I/O Server" from IPSL/CEA and CDI from DKRZ. This allows for both NetCDF and GRIB2 output formats respectively. This was to be implemented and tested in the EC-Earth climate model. The XIOS I/O library is already used in the Nemo ocean component of EC-Earth; the work involved adding it to the IFS (Integrated Forecasting System) atmosphere component, with additional changes to

ensure it is possible to output "CMOR" (Climate Model Output Rewriter) compatible output from XIOS to avoid the need for later offline processing.

Due to staffing issues, work planned for WP8 was not completed until the project extension. Within the extension, the XIOS interface for IFS was completed and GRIB writing was added to XIOS, and the memcache layer added to XIOS in WP8 was tested and evaluated.

2.3.2 Results

The main achievements at the end of the WP extension can be summarized as follows:

- Within the IFS atmosphere code there are several I/O methods, but they are not well factorized for adding or adapting to new formats. The iostream_mix module was refactored to provide a generic I/O interface. This module was chosen to allow post-processing within IFS to be turned off; IFS has a processing module called FullPos, which contains its own I/O output module (used within the Arpege configuration of the IFS/Arpege codebase, used by CNRM and Meteo France), but adding XIOS here would require FullPos to be used. Refactoring the iostream_mix module enables FullPos to be turned off in favour of the postprocessing within XIOS which is more flexible.
- iostream_mix was refactored into an abstract I/O layer which now calls either the original I/O implementation moved to gribio_api_interface.F90 (for GRIB2 output) or the XIOS interface in xios_api_interface.F90 (for NetCDF output). Shared code was moved to iostream_common.F90.
- Following changes in the CDI project upstream, it became apparent that there would be little demand for a CDI interface to XIOS. Hence the initial plan of calling XIOS via a CDI interface from IFS was dropped, and cdi_api_interface.F90 was dropped in favour of using the xios_api_interface.F90 as described above.
- Currently XIOS provides writing only (input functionality is being added by CEA). This is not a major issue for climate models as input is minimal except at start-up; IFS within EC-Earth2 did I/O in GRIB2 format, while the preferred format for climate results is NetCDF; this was achieved by postprocessing offline. XIOS outputs in NetCDF. In the new implementation, the existing gribio interface is currently used for input, and either gribio or XIOS may be selected for output.
- XIOS was extended to write using GRIB2 format using the grib_encode() methods used in CDI and the ECMWF grib library. The current implementation uses a single I/O node for GRIB writing; parallel grib writing was postponed to later work to use a planned redesign of XIOS communications.
- In WP8, we added a "memcache" layer to XIOS to handle cases with large numbers of clients. This layer acts as a buffer to cope with the case where there are a large number of clients (tens of thousands) and a small number of memory constrained I/O servers. In this case the model may stall as the I/O servers cannot buffer results for writing to disk, so the clients cannot execute the next timestep but must pause to wait for IO. In this case, the "memcache" layer adds extra nodes between the client and I/O server nodes to allow this buffering. The current solution in XIOS is to add extra I/O nodes, but this reaches the limit of scalability at ~10-20 I/O servers, as memory is needed on each client for each server connection, and in the current design each client connects to all servers.
- A memcache layer added to XIOS was designed to involve minimal changes in the design. Memcache nodes sit between the clients and writer (I/O server) nodes, acting as servers to the clients, and clients to the servers, buffering communications in memory as needed.

- The memcache was tested on Hermit with ~1000 clients and up to 10 I/O nodes doing writing, and up to 5 memcache nodes per I/O server. In practice the results were mixed. Analysis of benchmark and profiling runs showed that for short runs, filesystem `noise' dominated, making long runs of at least 10-20 minutes desirable for benchmarking; better results were achievable by adding more I/O nodes than adding memcache nodes for less than ~10-20 I/O nodes (depending on system). This analysis implied longer runs of at least 5-10,000 client nodes would be needed to see benefits of the memcache as currently implemented.
- Nevertheless, at a design review meeting in January 2014 in Dublin it was agreed that the memcache approach had merit, but required a redesign of the XIOS internal communications patterns to exploit fully. A redesign where clients / servers write to a specified set of nodes (either I/O or memcache) rather than the writing task being split across all I/O / memcache nodes. This would have two benefits: firstly it would minimise the nxm communications problem, and secondly it is required for GRIB writing for large files. GRIB requires files to be written in horizontal layers while clients have vertical slices: this needs a transposition of the 3-D matrices to be done before writing, which ceases to be possible as the model resolution goes to O(1km). Hence it is planned that the memcache layer be used and adapted to make GRIB transposition possible.
- Previous experience with the CMIP5 climate project showed that post-procesing climate model (offline) output took as much time, if not more, than the original model computation; the additional steps also led to much error in the process. XIOS enables post-processing to be done on-line as the model runs, avoiding extra input-process-output steps. Hence it was strongly desirable that the output was written in CMOR-compatible format. To this end, changes were made to XIOS to allow global attributes to be provided in the NetCDF output, and variable names from the models adapted to desired output conventions using an include file in the XIOS XML file "iodef.xml". A unique UUID can also be generated by the model (IFS or Nemo) and passed via namelist parameter as a global attribute and included in the output. This enables output to be CMOR-compliant. One feature remains to be completed which is the appending of new time series to an existing NetCDF file; this is being added by CEA.

2.3.3 Impact and Summary

- XIOS interface implemented in IFS; GRIB interface implemented for XIOS.
- IFS changes on the development branch of EC-Earth3 at the community site dev.ec-earth.org.
- Changes presented at EC-Earth semiannual conference, Reading, UK in February 2014
- The work in the project is being further developed to be used in the cy38 configuration of IFS/Arpege used in CNRM / Meteo France. Other projects such as the TM5 atmospheric chemistry component are now being ported to use XIOS, along with plans for the next-generation model at UK Met Office, and there is a convergence towards a single I/O library across climate models in Europe.

2.4 ICON

2.4.1 Overview and work plan

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) as well as static grid refinement to enhance resolution in geographical areas of interest.

The work plan for the ICON extension consisted of:

- One-week visit to DWD German Weather Service to plan the implementation and propose a validation strategy. This visit took place Jul. 22-26, 2013.
- Participation in the OpenACC consortium, and presentation of the OpenACC needs of the ICON NHDC, in particular for derived types. The OpenACC meeting took place Sep. 24-26, 2013.
- Code changes to allow the use of the Cray CCE deep copy of derive type instances to/from the accelerator, with the help from Cray.
- A validation framework to compare results on accelerator to those from a single CPU node
- Augmentation of NHDC kernels to utilise OpenACC in the framework agreed upon with DWD.
- Augmentation of ICON trunk communication modules to utilise OpenACC.
- Performance optimisation and tuning.
- Periodic merging of trunk into development branch.
- Final NHDC performance evaluation, dissemination of results, possible publication. Final update of trunk.

2.4.2 Results

In the WP8 extension, we introduced the modifications from the test-bed into to actual ICON development trunk, along with mechanisms to validate the accelerated code with respect to a version run on CPUs. All the performed tests show that the developed version of the code gives commensurate performance and scalability improvement to that of the testbed code, as reported in document D8.3 (see Figure 3). Note that the utilisation of the Cray CCE deep copy required extensive workarounds, and caused extensive delays.

We have validated numerous components of the dynamical core in the trunk, and have illustrated that it can run with local refinement to the grid. We have agreed with the ICON community on the process for integrating OpenACC code into the development trunk.

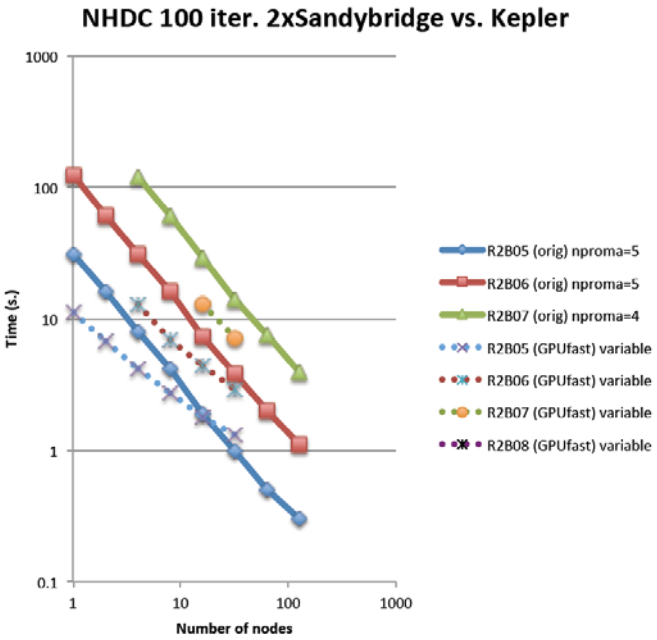

Figure 3: ICON's performance for several resolutions. A speedup of roughly 2x is measured for the configurations of interest, where the problem size first fits into system memory (left-hand side of graphs).

2.4.3 Impact and Summary

The ICON extension of WP8 proved that the techniques used in the testbed (result of WP8 first two years) could be adopted in the development trunk, albeit with some difficulty, and provide similar results.

The extension's main impact was that the OpenACC effort was accepted by the ICON development team, and its inclusion into the trunk seems assured through an iterative process agreed upon with the ICON community. Thus this project started the process toward realising a community-accepted accelerator-enabled version of ICON.

The accelerated work was presented at the following venues during the extension period:

- *Using GPUs for ICON: An MPI and OpenACC Implementation;* Heterogeneous Multi-Core 3 Workshop (http://data1.gfdl.noaa.gov/multi-core), Boulder, CO, USA, 19-20.9.2013. Presentation.
- *Using GPUs for ICON: An MPI and OpenACC Implementation;* OpenACC Face-to-Face Meeting, Oak Ridge National Laboratory, 24-26.9.2013. Presentation.
- *Multi-node OpenACC Implementation of the ICON Non-hydrostatic Dynamical Core;* Institute for Atmospheric and Climate Science, ETH Zurich, Switzerland, 27.1.2014. Presentation.
- *Towards a Multi-node OpenACC Implementation of the ICON Model;* European Geophysical Union (EGU), 28.4 - 2.5.2014, Lugano, Switzerland. Presentation.
- *Towards a Multi-node OpenACC Implementation of the ICON Model;* High Performance Computing for Science and Engineering, 19.5.2014, Lugano, Switzerland. Presentation and poster.
- *Towards a Multi-node OpenACC Implementation of the ICON Model;* PASC'14 (http://www.pasc14.org), 4-5.6.2014, Zurich, Switzerland. Presentation.
- *Proposal for an OpenACC implementation to the ICON development trunk;* ICON all-hands meeting, 10-13.6.2014, Loewenstein, Germany. Presentation.

2.5 Fluidity-ICOM

2.5.1 Overview and workplan

Fluidity-ICOM is built on top of Fluidity, an adaptive unstructured finite element code for computational fluid dynamics. It consists of a three-dimensional non-hydrostatic parallel multiscale ocean model, which implements various finite element and finite volume discretisation methods on unstructured anisotropic adaptive meshes so that a very wide range of coupled solution structures may be accurately and efficiently represented in a single numerical simulation without the need for nested grids. It is used in a number of different scientific areas including geophysical fluid dynamics, computational fluid dynamics, ocean modelling and mantle convection. Fluidity-ICOM uses state-of-the-art and standardised 3rd party software components whenever possible. For example, PETSc is used for solving sparse linear systems while Zoltan is used for many critical parallel data-management services. Both have compatible open source licenses. Python is widely used within Fluidity-ICOM at run time for user-defined functions and for diagnostic tools and problem setup. It requires in total about 17 other third party software packages and uses three languages (Fortran, C++, Python). Fluidity-ICOM is coupled to a mesh optimisation library allowing for dynamic mesh adaptivity.

As the I/O component has now become the major bottleneck we are now applying the extension effort in WP8 to tackle this issue. Our proposed solution is to integrate PETSc's DMPlex module. This module is able to generate the necessary halo/ghost regions at runtime from the underlying mesh topology, eliminating the need for external domain decomposition and improving start-up times. It furthermore provides native interfaces for common meshing formats. Based on the above work, we have further optimized the hybrid OpenMP/MPI code for the large ocean test case.

2.5.2 Results

As Fluidity-ICOM already uses PETSc for solving sparse linear systems, it is a natural progression to use DMPlex, a subclass of PETSc data management (DM) object, which allows the user to handle unstructured grids using the generic DM interface for hierarchy and multi-physics. It also facilitates the way to switch between different discretization methods in the same code for further scientific investigation. DMPlex provides an abstraction layer for mesh topology, supports unstructured meshes with multiple mesh and I/O file formats and it decouples topology from field data. The main advantage of DMPlex in representing topology is that it treats all the different components of a mesh, e.g. cells, faces, edges, vertices in exactly the same way. This allows the interface to be very small and simple, while remaining flexible and general. This also allows “dimension independent programming”, which means that the same algorithm can be used unchanged for meshes of different shapes and dimensions.

Figure 3. shows that significant I/O efficiency has been achieved by using the current mechanism implemented together with DMPlex. This significantly reduces the number of metadata operations on large numbers of nodes, which may otherwise hinder overall performance.

Figure 4. shows the total Fluidity-ICOM runtime and parallel efficiency. It is clear that pure MPI runs faster up to 4096 cores. However, due to the halo size increasing exponentially with the number of MPI tasks, the cost of MPI communication becomes more dominant from 4096 cores onwards, where the mixed mode with reduced halos begins to outperform the pure MPI version.

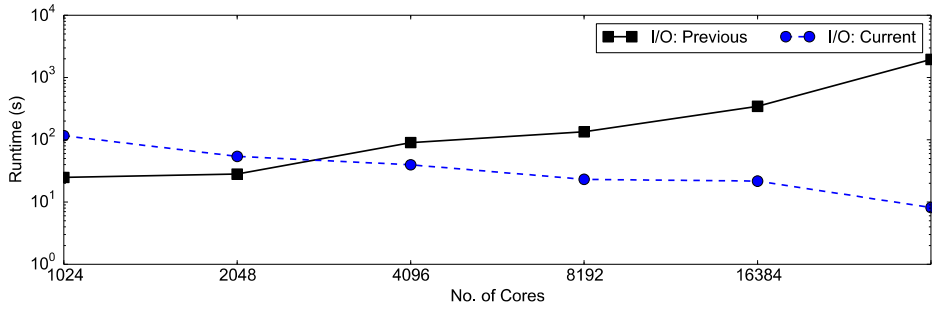

Figure 3: The Fluidity-ICOM current I/O Performance comparison.

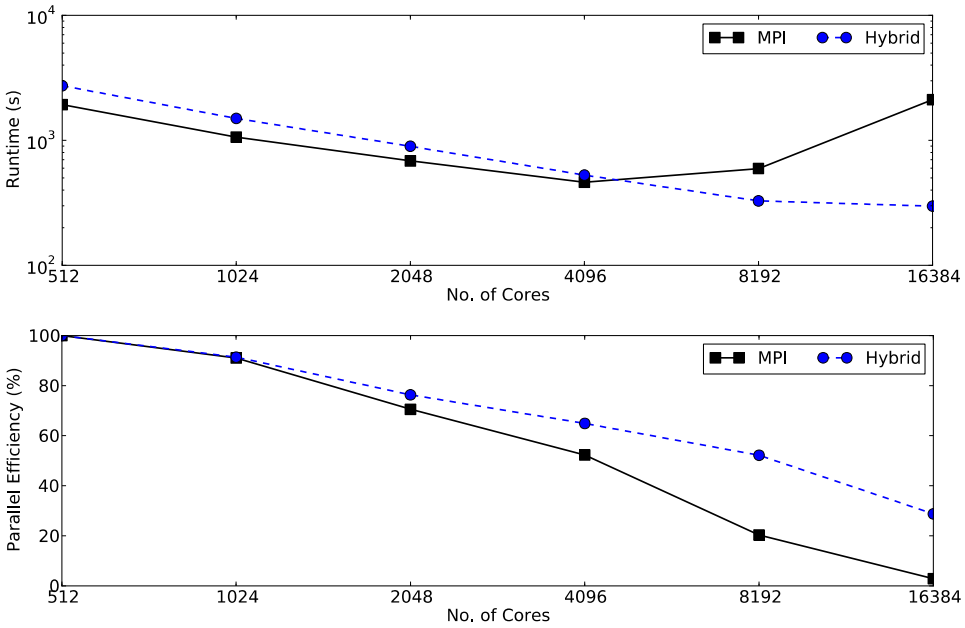

Figure 4: The overall performance of Fluidity-ICOM hybrid OpenMP/MPI compared with the pure MPI version.

2.5.3 Impact and Summary

The main results obtained in WP8:

- We have accomplished threading the Fluidity-ICOM code during this project. This has involved parallelizing CG(Continuous Garlerkin), CV(Control Volume) matrix assembly kernels with OpenMP. The performance results indicate that node optimisation can be achieved using OpenMP with efficient colouring methods. As a result, the matrix assembly kernels now scale well up to 32768 cores.
- We have done various optimizations of the threaded sparse linear preconditioner and KSP(Krylov Subspace Methods) solvers. The work has been carried out using a range of large ocean test cases.
- Several scalability bottlenecks have been identified and fixed. The mixed mode begins to outperform the pure MPI version in runs using 4096 cores and upwards, Fluidity-ICOM is now therefore able to run efficiently on Petascale platforms.
- PETSc DMPlex has been introduced into Fluidity to tackle the I/O bottleneck, which offers a potential solution for both performance and scientific data visualization and analysis solutions.

The activities undertaken with the communities to reintroduce the code are:

- All our contributions from PRACE-2IP WP8 for Fluidity-ICOM have already passed through all validation tests by using buildbot and merged into the Fluidity-ICOM main trunk.
- All these optimizations offer Fluidity-ICOM the capability to solve "grand-challenge" problems.
- The work has been presented at EASC2013 (http://www.easc2013.org.uk/easc2013-solving-software-challenges-exascale).
- The work has been presented at ParCFD 2013 (http://aca.hnu.cn/parcfd2013/).
- The wiki on HPCForge with key results and conclusions.
- Published papers:
  - Guo Xiaohu et al. "*Developing a scalable hybrid MPI/OpenMP unstructured finite element model*." In: Computers & Fluids to appear (2014).
  - Guo Xiaohu et al. "*Developing the multi-level parallelisms for Fluidity-ICOM – Paving the way to exascale for the next generation geophysical fluid modelling technology*." In: Advances in Engineering Software to appear (2014).
  - Xiaohu Guo et al. "*Exploring the thread-level parallelisms for the next generation geophysical fluid modelling framework Fluidity-ICOM*." In: Procedia Engineering 61 (2013), pp. 251–257.
  - Michael Lange, Gerard Gorman, Michele Weiland, Lawrence Mitchell, Xiaohu Guo, James Southern, "*Benchmarking mixed-mode PETSc performance on high-performance architectures.*" In: Advances in Engineering Software to appear (2014).

Other impacts on the scientific community:

Fluidity-ICOM is an open source project, all the improvements developed in this project will benefit all Fluidity-ICOM users and methods are available for analysis and use by the wider scientific community

2.6 Quantum ESPRESSO

2.6.1 Overview and workplan

Quantum ESPRESSO is an integrated suite of computer codes based on density-functional theory, plane waves, and pseudo-potentials - separable, norm-conserving and ultrasoft - and projector-augmented waves. The acronym ESPRESSO stands for opEn Source Package for Research in Electronic Structure, Simulation, and Optimisation. It is freely available under the terms of the GNU General Public License (GPL). It builds upon newly restructured electronic-structure codes that have been developed and tested by some of the original authors of novel electronic-structure algorithms and applied in the last twenty years by hundreds of materials modelling groups.

Quantum ESPRESSO is a modular software package developed and presented with two goals: 1) to enable state-of-the-art materials simulations, and 2) to foster methodological innovation in the field of electronic structure and simulations by providing highly efficient, robust, and user-friendly open source codes containing most recent developments in the field. This approach blurs the line separating development and production codes and engages and nurtures the user community by inviting their software contributions. These are included in the distribution after being verified, validated, and made fully inter-operable with other modules.

Speed of execution and parallel scalability have always been considered key criteria for the assessment of quality of a software. Although they both are major aspects in a community

code, an important characteristic is also represented by its flexibility. By "flexibility" we mean the possibility to run an application on a variety of architectures as wide as possible. This has always been one of the driving criteria in the development of Quantum ESPRESSO.

Following this principle, for Quantum ESPRESSO we adopted a strategy aimed to expose as much parallelism as possible, using a multi-level hierarchy of parallelisms and using both MPI and OpenMP approaches. More recently, with the advent of heterogeneous systems, a branch of development was devoted to the porting to the GPUs of Quantum ESPRESSO (named as QE-GPU).

The activity inWP8 permitted us to extend the flexibility of Quantum ESPRESSO, making it able to run on the new Intel MIC architecture. This work was conducted in synergy with the activity of Cineca in the framework of the Intel Parallel Computing Center. In order to take advantage of the Intel Knights Corner cards, a mechanism to offload the linear algebra kernels was implemented in Quantum ESPRESSO. All the Intel MKL functions can use the card(s) to accelerate the execution of the computation trying to hide as much as possible the latency due to the data transfer between host and device.

The first part of this work was concluded and preliminary results were shown to the community of users/developers.

2.6.2 Results

In order to enable Quantum ESPRESSO to run on Intel MIC architectures, a precious contribution was available from the experience previously obtained during the porting to the GPUs. In particular, that experience stressed that the advantages coming from the use of an external device (such as an accelerator) were strictly related to the input data set and from the right choice of run parameters. This fact was indeed confirmed by our experiences with the porting of Quantum ESPRESSO to the Intel MIC.

The driving idea behind the activity of porting were the following:

1) Preserve the maintainability of the code, by reducing as much as possible the impact on it;
2) Hide the latency of data moving between host and device.

Target 1) was fulfilled using an approach that permits us to intercept the MKL calls through a dynamic library. A MKL function called by the application is wrapped and the execution is assigned to the MIC card. With this approach, the original code is left untouched. Using this wrapping library different versions of the code can be run exploiting the presence of a MIC card on the system.

Target 2) was the main obstacle from the point of view of performance (i.e. time-to-solution). Just as with GPUs the gain in performance when using an accelerator strongly depends on the input dataset. A significant improvement of the time-to-solution is achievable only when the computational workload is large enough to make the data transfer/computation ratio small.

When the workload related to the execution of an MKL function (mainly matrix-matrix and matrix-vector operations) is large enough, an improvement of the value of the time to solution can be achieved, as shown in Figure 4.

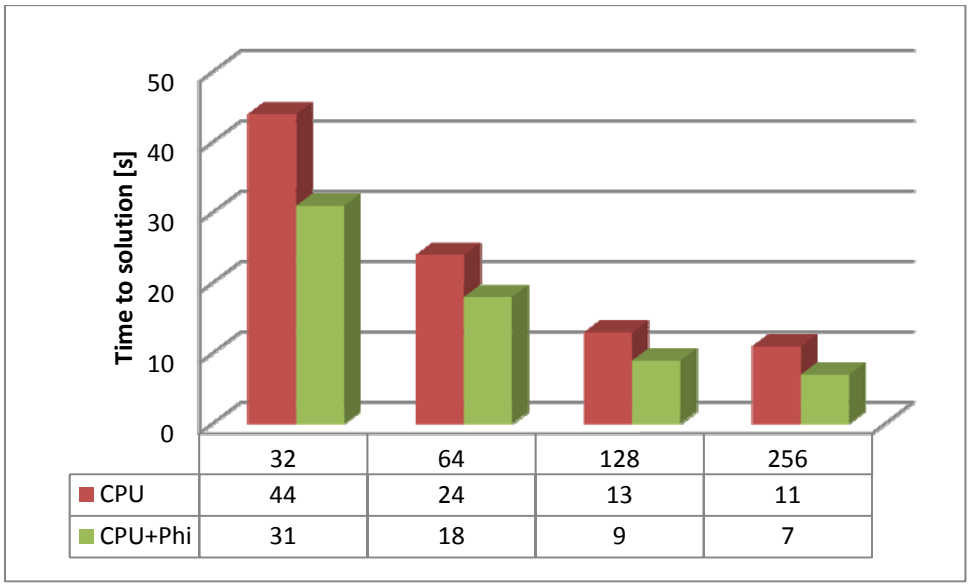

Figure 4: comparison of performance on CPU and on Xeon PHI

Inspired by the work carried out by N Varini et al at ICHEC in PRACE-1IP on parallelising the GIPAW code over electronic bands using MPI, ICHEC continues to work on introducing a new level of parallelism into the PHonon code in collaboration with the QE community.

During the PRACE-2IP WP8 extension period, PHonon has been further profiled using Allinea MAP v4.2 on ICHEC clusters to identify the most time consuming parts of the overall execution. As an example, we consider a phonon calculation at the X point for Silicon in the data set that comes with QE-PHonon. By analyzing the output of the MAP profiler, compute intensive loops over electron bands are found in many subroutines in the PHonon code. Motivated by this, we have focused on the parallelization of the following subroutines over bands analogous to the parallelization strategy that has been implemented in the GIPAW code: `cgsolve_all.f90, ch_psi_all.f90, h_psiq.f90`.

The calling order of these routines are: `phonon(), do_phonon(),phqscf(), solve_linter(),cgsolve_all.f90, ch_psi_all.f90, h_psiq.f90`. In particular, `h_psiq_k()`, called with in the `h_psiq()` is one of the most time consuming part in this branching. We illustrate the parallelization of the calculation of the HPSI matrix below. Here, loop iterations are divided evenly among the specified band groups. So, each group computes `ibnd_end-ibnd_start` iterations. At the end of the loop, the results of each group are summed up.

```
hpsi=(0.0_dp,0.0_dp)
DO ibnd = ibnd_start, ibnd_end
   DO j = 1, n
      hpsi (j, ibnd) = g2kin (j) * psi (j, ibnd)
   ENDDO
ENDDO
call mp_sum(hpsi, inter_bgrp_comm)
```

We have implemented this approach on other loops over bands in a similar fashion since there are no loop-carried data dependencies. However, the PHonon code is more complicated than GIPAW and there are more dependencies between band groups. Thus, care must be taken that ‘reductions’ are implemented at relevant points in the code. So far, we have modified many relevant loops in the call tree to allow for parallelization over bands. Although there are minor

inconsistencies in results between the newly bands-parallelized version of PHonon and the original version, there are clear indications that the method of band parallelization could work successfully for PHonon. We are currently at the stage of testing the implementation in close collaboration with the QE community and are seeking larger data sets to profile the code over larger node counts. We intend to complete a more detailed whitepaper for both PRACE and the QE community at the end of PRACE-2IP.

2.6.3 Impact and Summary

During the whole activity of the WP8, the following targets have been achieved:

- Extension of the level of parallelism
  - Band parallelization
  - Extension of the OpenMP parallelization over the different modules of the Quantum ESPRESSO distribution
- Improvement of the modularization of the application
  - Making the code more homogenous
  - Sharing low-level libraries
- Implementation of more effective linear algebra libraries
  - ELPA
- Testing and validation of new algorithms
  - EXX benchmarking

A part of the work, such as the improvement of the parallelism in the EPW and PHonon modules, was not completed. This activity is however likely to be continued after the conclusion of WP8, thanks to the efforts of the developers community.

The developers community has been continuously involved during the evolution of the WP8 work. Results obtained during the WP8 were shown during the “Developers’ meeting” held in Trieste and Lausanne in the January of each year. The work performed has always raised interest in the developers’ community and attracted external contributions.

The code has been continuously updated through the SVN server and the modifications on the main trunk have been released during the PRACE-2IP project.

Scientific community appreciation was increased and the deployment of the new versions of Quantum ESPRESSO on the supercomputers available at CINECA attracted more and more users.

2.7 SIESTA

2.7.1 Overview and workplan

SIESTA is both a method and its computer program implementation, to perform efficient electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids. SIESTA's efficiency stems from the use of strictly localized basis sets and from the implementation of linear-scaling algorithms which can be applied to suitable systems. A very important feature of the code is that its accuracy and cost can be tuned in many aspects, from quick exploratory calculations to highly accurate simulations matching the quality of other approaches, such as plane-wave and all-electron methods.

The possibility of treating large systems with some first-principles electronic-structure methods has opened up new opportunities in many disciplines. The SIESTA program is distributed freely to academics and has become quite popular, being increasingly used by researchers in geosciences, biology, and engineering (apart from those in its natural habitat of materials physics and chemistry). Currently there are several thousand users all over the

world, and the paper describing the method (J. Phys. Cond. Matt. 14, 2745 (2002)) has had more than 3500 citations.

The work previously done in PRACE-2IP, WP8, extended the range of systems that can be calculated towards much larger problem sizes due to the implementation of the PEXSI solver, which reduces the scaling of the computational cost with the system size, without loss of generality or accuracy.

A consequence of SIESTA's localized basis is, that the essential matrices are sparse, which saves a lot of memory, but makes I/O more demanding due to irregular data structures. The I/O operations implemented follow the easiest approach of gathering the data on one processor, which also dumps it into a file. But for very large systems this can become a bottleneck, regarding the time as well as the amount of memory needed.

The work for the extension is to implement a parallel version of SIESTA's writing and reading of those matrices that can be used for a restart of the calculation. Furthermore, the checkpointing and restarting capabilities are extended.

2.7.2 Results

More efficient serial IO

As a first step serial I/O has been improved. By processing bigger blocks of data, the time for I/O was reduced by a factor of 10. This improved code served as a base for the parallel implementation.

Parallelized IO:

Decision was taken to use HDF5 as parallel I/O infrastructure due to previous, positive experience at BSC. A file format for writing sparse matrices was defined. The format, as printed by the tool h5dump, is:

```
HDF5 "<<filename>>" {
GROUP "/" {
   ATTRIBUTE "blocksize" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   ATTRIBUTE "number_of_orbitals" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   ATTRIBUTE "number_of_spins" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   ATTRIBUTE "use_padding" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "col_ind" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( <<num_orb>> ) / (<<num_orb>> ) }
   }
   DATASET "row_ptr" {
```

```
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( <<num_nonzeros>> ) / (<<num_nonzeros>>) }
   }
   DATASET "val" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { (<<num_spin>>, <<num_nonzeros>>  ) /
                          (<<num_spin>>, <<num_nonzeros>>  ) }
   }
}
}
```

Implementation into SIESTA

The infrastructure for writing HDF5 files in parallel on a set of processors, which can be only a subset of the processors doing the SIESTA-calculations, was implemented. Some options are exposed to the user and can be set in the fdf input file. Those options are:

`WriteDM.HDF5`	Activate or deactivate using the HDF5 format.
`WriteDM.num-IO-nodes`	Number of processors to use for the parallel IO.
`WriteDM.collective_IO`	Activate or deactivate collective I/O routines. The performance of collective I/O is much better, but it comes at the cost of larger file sizes due to some padding needed in the process.

Improved checkpointing

There are a few new options for IO, which go with recent developments for mixing and checkpointing:

- The density matrix or the Hamiltonian, or both, can be saved.
- The matrices corresponding to the last electronic state computed, or the matrices already mixed for being the input of the next SCF iteration can be written.
- The matrices can be written after each SCF iteration, or only after finishing the whole SCF loop (one geometry step).

2.7.3 Impact and Summary

Main results:

- Adaptation of the PEXSI solver and implementation into SIESTA
- Testing capabilities and performance of SIESTA-PEXSI for insulating, semi-metallic and metallic systems, up to a size of ~20000 atoms calculated on up to 16000 processors.
- Implementation of parallel IO

Dissemination:

- Paper: SIESTA-PEXSI: massively parallel method for efficient and accurate ab initio materials simulation without matrix diagonalization; Lin Lin, Alberto Garcia, Georg Huhs, Chao Yang; 2014 *J. Phys.: Condens. Matter* 26 305503
- Presentation of SIESTA-PEXSI at the following conferences:
  - PASC 2014; Zürich
  - APS March meeting 2014; Denver, Colorado
  - SIAM Parallel Processing 2014; Portland, Oregon

    - Trends in Nanotechnology 2013; Seville, Spain
- Presentation at seminar talks at
    - Lawrence Berkeley National Laboratory
    - Graz University of Technology
    - Barcelona Supercomputing Center
- Contribution to a library of electronic structure functionalities driven by CECAM
    - Documenting PEXSI and its usage for electronic structure calculations
    - Contributing experience with parallel I/O and the sparse file format to I/O libraries

Other impacts:

PEXSI extends significantly the applicability of SIESTA due to its reduced computational cost, its potential to use large amounts of processors, and its reduced memory needs. Large systems can be solved orders of magnitude faster, and system sizes that were not even reachable can be treated now. The solver is general, hence not limited to specific cases or problem classes. However, the largest gain can be achieved for lower-dimensional systems like nanotubes, molecules, or Graphene and similar materials.

2.8 Exciting/ELK

2.8.1 Overview and workplan

Exciting

Exciting is a full-potential all-electron density-functional-theory (DFT) package based on the linearised augmented plane-wave (LAPW) method. It can be applied to all kinds of materials, irrespective of the atomic species involved and also allows for the investigation of the atomic-core region. The code particularly focuses on excited state properties, within the framework of time-dependent DFT (TDDFT) as well as within many-body perturbation theory (MBPT). The code is freely available under the GNU General Public License.

Elk

Elk is an all-electron full-potential linearised augmented-plane wave (FP-LAPW) code with many advanced features. Written originally at Karl-Franzens-Universität Graz as a milestone of the EXCITING EU Research and Training Network, the code is designed to be as simple as possible so that new developments in the field of DFT can be added quickly and reliably. The code focuses on ground state properties with some effort devoted to excited state properties. The code is freely available under the GNU General Public License.

Both Exciting and Elk codes are successors of the original EXCITING FP-LAPW code and have a lot of common algorithms and functionality. In order to preserve the “canonical” Fortran implementations which are well known to the corresponding scientific communities and at the same time to avoid a redundant job of optimising two codes the decision was made to isolate generic LAPW method algorithms in a standalone domain-specific LAPW library ‘SIRIUS’ which is independent of a particular LAPW code implementation.

During the extension period of the WP8 the following refactoring work plan was accomplished:

- Most critical parts of the Kohn-Sham SCF loop are ported to GPU. This includes the setup of the Hamiltonian and overlap matrices (using vendor’s GPU-aware PBLAS library), solution of the generalized eigenvalue problem (using a new GPU-enabled distributed eigenvalue solver) and construction of the interstitial charge density (using CUDA FFT).

- The low-level domain specific LAPW library was fully integrated back into the original Exciting code. The public release of the new version of Exciting code and SIRIUS library are scheduled for the last quarter of 2014.

2.8.2 Results

The extension period of the WP8 was solely dedicated to porting the SIRIUS library to the GPU devices and subsequent benchmarking. The GPU porting strategy relies heavily on the accelerated versions of (P)BLAS and ScaLAPACK libraries provided by vendors. Only interstitial charge density summation was hand-coded using CUDA FFT. In order to benchmark the GPU implementation a full-potential DFT ground states simulation of Li-ion battery cathode was selected as a representative test case. Li-intercalated CoO_2 supercell containing 432 formula units of CoO_2 and 205 atoms of lithium (1501 atoms in total) was created. Li sites were randomly populated to produce a ~50% intercalation. A single Γ-point calculation with ~115000 basis functions and ~7900 lowest bands to compute was setup. The results of the benchmark are shown in Fig. 3. These results demonstrate that highly accurate and transferable quantum simulations are now usable for high-throughput materials search problems, given the necessary compute capabilities.

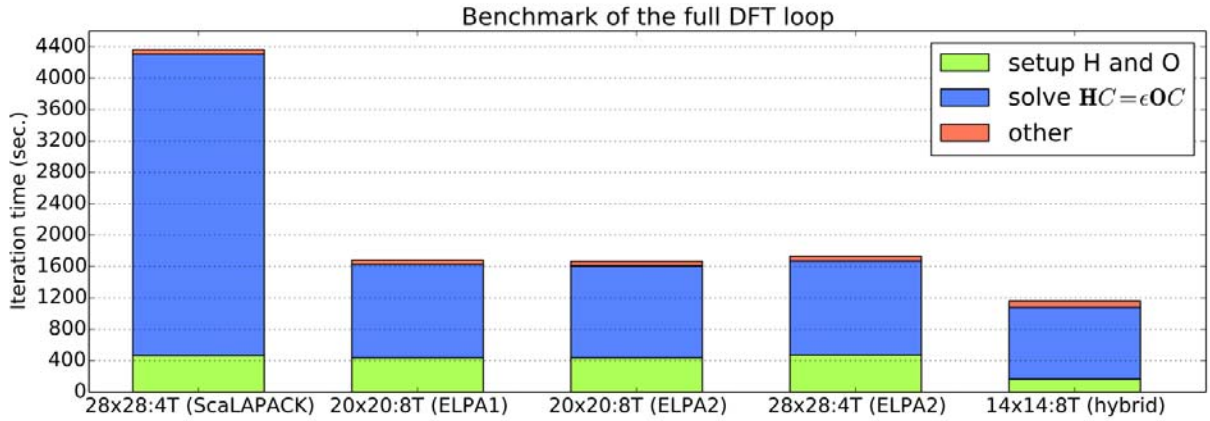

Figure 5: Kohn-Sham SCF iteration time for ~1500-atom unit cell of Li_xCoO_2 on a Cray XC30 platform. X-axis labels contain the dimensions of the BLACS grid and number of threads per MPI rank. N hybrid nodes to 2N CPU-only sockets comparison is used. Time for the eigenvalue problem setup (green), eigenvalue problem solution (blue) and the rest of the DFT cycle (red) is measured. A full SCF iteration step is executed in less than 20 minutes on 196 hybrid CPU-CPU nodes and in about 30 minutes on an equivalent number of 400 CPU sockets. The results of the benchmark show that the hybrid CPU-GPU implementation of the LAPW code is faster than the best CPU-only realisation of the code by a factor of ~1.44.

2.8.3 Impact and Summary

The main result of the WP8 is associated with the development of a standalone domain-specific LAPW library which is bitwise compatible with Exciting and Elk codes and which can be further expanded to provide more LAPW-based functionality (excited state properties is one example). The library can also be used as a sandbox to test new algorithms and concepts. The impact on scientific community is starting to evince itself as some implementation ideas of SIRIUS library are gradually incorporated back into native Fortran90 code.

- The accomplished work was presented during the Exciting code developer's week (Berlin, Germany, December 5, 2013) in the talk "SIRIUS library: present and future" by A. Kozhevnikov (CSCS). Another talk about GPU-enabled LAPW calculations will be given during the CECAM workshop (HoW exciting! Hands-on Workshop on Excitations in Solids, Berlin, Germany, July 31, 2014 to August 8, 2014).

2.9 PLQCD

2.9.1 Overview and workplan

As a part of the code re-factoring efforts in WP8, the goal of this work was to improve the scaling of community codes for Lattice QCD applications on multi-core architectures. Four main computational tasks were selected: application of the Lattice Dirac operator, iterative solver, Landau Gauge Fixing, and optimization of the Monte Carlo integrator. For the Lattice Dirac operator, a new library was built in which a hybrid parallelization with MPI and OpenMP was used in order to improve the scaling on multi-core machines. This library was given the name PLQCD. In addition to using a hybrid parallelization approach, other improvements were implemented using compiler intrinsics for SSE3 vectorization and the implementation of AVX instructions. The next computational task considered was the improvement of the iterative solver. For this work we focused on the tmLQCD software developed by the European Twisted Mass Collaboration. We implemented a deflated Conjugate Gradient solver for Twisted Mass fermions based on the EigCG algorithm. This solver has been very beneficial within the ETMC community and has reduced computational cost by a factor of 2-3 as reported in a recent large scale physics calculation. For the Landau Gauge Fixing and the Hybrid Monte Carlo integrator, we focused on the Chroma software developed by the U.S. Lattice QCD Collaboration. In the case of Landau Gauge fixing, we have implemented a code that uses a new parallel Fast Fourier Transform library called PFFT. This library allows for parallelization in 3 dimensions and thus improving the scaling of the Gauge Fixing code. Finally, in the case of tuning the Hybrid Monte Carlo integrator, we have implemented a new method based on Poisson Brackets to determine the optimal value of the Omelyian integrator parameter. This tuning leads to a better acceptance during the generation of Gauge configurations, consequently reducing the computational cost as well. These developments have been done in close collaboration with the developer of the selected community codes leading to mutual benefit between the developers and users.

During the extension of WP8, we focused on PLQCD library for the Lattice Dirac operator aiming at optimizing the implementation for the new Intel MIC architecture.

2.9.2 Results

MIC cards offer the advantage of allowing codes with MPI and OpenMP to run directly on them without the need to re-write the code using a new programming language. Intel MIC cards are becoming an important component of current and future machines. In order to prepare for the deployment of such machines we focused on re-factoring the implementation of the Lattice Dirac operator to achieve maximum benefit of the performance on the Intel MIC. This required the re-design of the order of the operations involved in the Lattice Dirac operator and the layout of lattice fields to benefit from the vectorization capabilities on the MIC. The 512 bit wide registers available on the MIC allows for the simultaneous operation on 8 double or 16 single precision numbers. In order to utilize these vectorization capabilities we found that it was necessary to group the lattice sites into 8 sites, when the real and imaginary parts of the lattice fields are stored separately or into 4 sites when lattice fields are stored as complex double precision numbers. In case of using single precision, the number of lattice sites to be grouped together will be twice as much. This was found to be the best strategy to map the computation of the Dirac operator to the MIC. The hopping part of the Lattice Dirac operator is given by

$$\psi(x) = \kappa \sum_{\mu} [U_{\mu}(x)(1-\gamma_{\mu})\phi(x+e_{\mu}) + U_{\mu}^{-1}(x-e_{\mu})(1+\gamma_{\mu})\phi(x-e_{\mu})]$$

where ϕ is the input spinor field, ψ is the output spinor field, U_μ is the gauge link (3x3 unitary matrix), γ_μ are Dirac spin matrices, x is the lattice site and e_μ is a unit vector in the μ direction. Two approaches are usually used in lattice codes. The first approach is loop over the input spinor using some auxiliary fields η(x) and χ(x) defined by

$$\eta_\mu(x) = U_\mu^{-1}(x)\big(1+\gamma_\mu\big)\phi(x)$$

$$\chi_\mu(x) = \big(1-\gamma_\mu\big)\phi(x)$$

and build the result by

$$\psi(x) = \kappa\sum_\mu[U_\mu(x)\chi_\mu\big(x+e_\mu\big) + \eta_\mu\big(x-e_\mu\big)]$$

This approach could have improved cache utilization since the input spinor is accessed in order. It requires however the memory access of the auxiliary fields more than once. In the second approach one loops over the output spinor and avoid the use of auxiliary fields. This might be more efficient for the case of shared memory. We tested both methods. Another implementation approach we tested is using complex fields versus separating the real and imaginary parts. Lattice data are complex and in virtually all codes the data is stored and treated as complex variables. Performing complex number multiplications require some shuffling when the data is stored as complex in memory. Alternatively one can use separate arrays for the real and imaginary parts. This has the advantage of avoiding shuffle operations for the vectorization part. When splitting the real and imaginary parts however, we need to pack data for more than a single site together even when using SSE2 vectorization. This is not a problem since the lattice size is large enough to allow for this. To pack sites together, (either as complex or separate real and imaginary parts), we chose the x-direction and divide the sites according their x-coordinate. For SSE2 the lattice size in the x-direction must be a multiple of 2,4,8 when using complex representation for SSE2, AVX, and MIC cases respectively. Twice this number is needed when dealing with real and imaginary parts separately. In this discussion we are assuming double precision, for single precision these sizes has to be doubled again to allow for packing more data. We have implemented the necessary codes for this vectorization and change of the data layout within PLQCD. This vectorization together with the use of openMP was found to lead to a considerable gain in performance.

In addition to the Dirac operator work in PLQCD, we also worked on the GMRES-DR solver within tmLQCD. Our goal is to combine eigenvectors computed with GMRES-DR with a deflated BiCGStab for Twisted-Mass fermions. Progress is made in this algorithmic work and we hope to have results for this solver in the near future.

An implementation of the Lattice Dirac operator within PLQCD for the Intel MIC architecture has been done. For this purpose, the data layout of lattice fields has changed in order to benefit from the vectorization capabilities of the MIC. This new implementation also takes into account the very likely situation that new MIC cards will have even a wider registers. In such case, the newly implemented approach can be easily extended with small code modifications. we show the achieved performance of the Hopping part of the Dirac operator for the case of PLQCD on a single MIC for the case of complex lattice field in double precision. For this test we use a lattice of size L=32 in the spatial direction and T=64 in the time direction. It is interesting to point out that our codes allows for such a large lattice to fit on a single MIC card. In this plot we show results for the case when complex and split (real and imaginary separated) representation of the complex data is used. We also show the effect when using an auxiliary field as discussed before. These results show that for a single MIC, the split representation is more efficient. It also shows that performance is better without the auxiliary field. This is probably due to the extra memory access which is not

needed in case of a single MIC. The results are encouraging and we plan to continue our investigation of possible performance improvements.

The best performance we get is when all 240 threads on the MIC were used. We compare this performance to the CPU performance. The host CPU is Intel Xeon E5 at 2.0GHZ with 8 cores and 16 threads. For this comparison we have implemented the split representation for SSE3 and AVX and run the code with 16 threads. Results are presented in Table 2 which shows the MIC to be about 4 times faster than the SSE3 implementation.

In addition, preliminary work has been done in combining GMRES-DR iterative solver with BiCGStab solver for tmLQCD.

Case	GFlops/s	#of threads	Speed up factor
SSE3	20.7	16	1
AVX	25.1	16	1.2
MIC	85.2	240	4.16

Table 2: comparing CPU and MIC performance for the hopping matrix

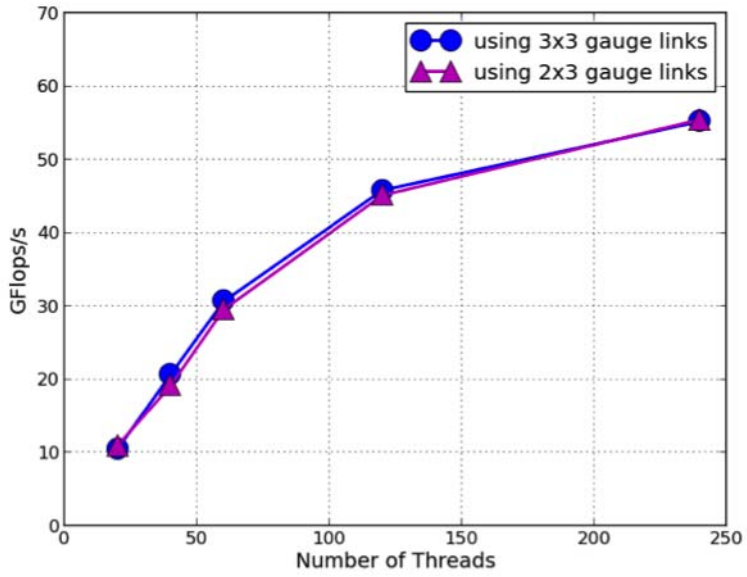

Figure 6: Performance of the hopping part of the lattice Dirac operator on a single MIC for a lattice size L=32, T=64 with complex double fields

2.9.3 Impact and Summary

In summary, the following has been accomplished in WP8 for Lattice QCD codes:

- Writing a new library (PLQCD) for implementing the Lattice Dirac operator in which both MPI and OpenMP parallelization is used. This approach targets multi-core architectures which are expected to be the main feature of future machines. This implementation has been extended to the Intel MIC cards benefiting from the many cores and the large vectorization capabilities available on the MIC.
- The EigCG iterative solver is implemented in tmLQCD software suite which leads to a significant savings in computational cost used in large scale lattice calculations.
- The Fast Fourier Transform part of the Landau Gauge Fixing code in Chroma has been improved using the PFFT library allowing parallelization in three dimensions and thus improving the scaling of the code.
- An optimization approach of the parameter used in the Omelyian integrator used in the Hybrid Monte Carlo simulation is implemented in Chroma. This leads to a higher acceptance rate than simply guessing this parameter during the simulation.

- Preliminary work for combining the GMRES-DR solver with BiCGStab solver for tmLQCD has been done which hopefully allows the efficient use of BiCGStab for Twisted Mass fermions. It is known that BiCGStab solver doesn't converge for Twisted-Mass fermions.

Code development and refactoring has been done in close collaboration with communities and developers of tmLQCD and Chroma codes. Results from code development work have been presented during the European Twisted Mass collaboration meetings. In addition, it has been presented at the annual International Lattice QCD Symposium. To highlight the importance of code development activities in light of the emergence of new architectures and to initiate a community wide effort in this direction, we have organized a special coding session during the 31st International Symposium on Lattice Field Theory that was held at Johannes Gutenberg University Mainz, Germany from Monday 29th July to Saturday 3rd August 2013. Work done in WP8 has been presented in three publications and we also plan to write the work done for the MIC in a forthcoming publication.

One of the great benefits of the refactoring effort in WP8 is the strengthening of collaboration in code development efforts with developers of some of the main community codes such as tmLQCD and Chroma. This collaboration is expected to continue beyond WP8. Another important achievement of this work is that it highlights the extreme importance of improving algorithms. The large reduction in computational cost offered by an efficient algorithm such as EigCG, emphasizes the importance of working not only on improving the hardware, but also on efficient numerical algorithms. One important aspect of this issue which is expected to pay a role in the near future is how to map these algorithms properly to the new architectures. In the past, it was assumed that these algorithms will be run on CPUs. Now, however, it is important to see how or if it will be possible to map these algorithms to new architectures such as the Intel MIC or GPUs.

2.10 ELMER

2.10.1 Overview and workplan

The main objective of the previous subtask was the scalability analysis and bottleneck identification of the Elmer (open source multiphysical simulation software developed by CSC - IT Center for Science in Helsinki, Finland) solvers and its performance improvement by means of of FLLOP (FETI Light Layer On top of PETSc - a novel software package developed at IT4Innovations, VSB-Technical University of Ostrava, Czech Republic, for solution of constrained quadratic programming problems (QP)) solvers for large systems of linear equations arising from FEM via implemented Elmer-FLLOP interface. The FETI type solvers were identified as the most efficient tool. The final contribution was 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 constrained QPs.

The main target for an extension was the scalability testing and comparison of optimized implementations of linear solvers (direct, iterative) running on CPUs only, CPUs+GPUs and CPUs+MICs on benchmarks with 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. We paid extra attention to the implementation, testing and comparison of linear solvers running on:

- CPUs only,
- accelerated CPUs by means of GPUs,
- accelerated CPUs by means of MICs.

We plan to test:

- direct solvers – e.g. LU in Magma, SuperLU, MUMPS, etc.
- iterative solvers – e.g. CG, DCG, BiCG in PETSc etc.,

for both matrix formats:

- sparse,
- dense.

2.10.2 Results

For the numerical testing a loaded elastic cube generated by PermonCube is used. In our case, the loading is f_z = 77 N/mm^3, Young's modulus E = 2e5 MPa, and Poisson's ratio μ = 0.33. The numerical experiments have been performed using Anselm (Bull cluster at IT4Innovations), HECToR (Cray XE6 at EPCC) and Sisu (Cray XC30 at CSC Helsinki).

- Coarse problem (CP) (TFETI, DCG - $GG^Tx = y$) solution using dense MagmaLU on CPU and CPU accelerated by GPU and MIC was tested through interface accessible from Elmer. We compared LU factorization and solve functions.

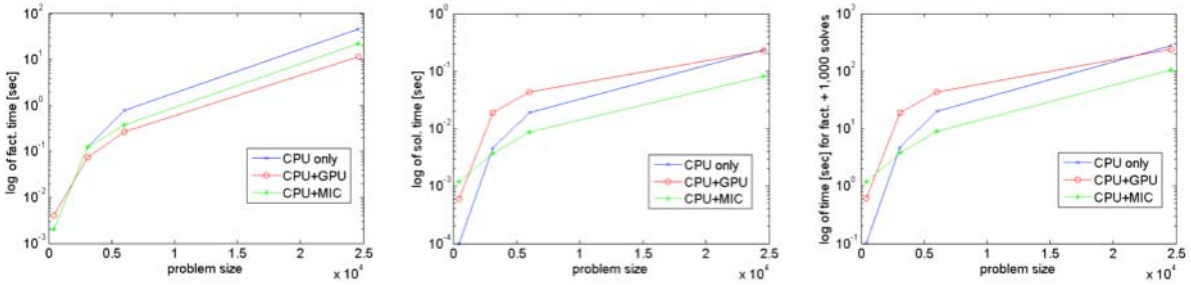

Figure 7: Log of times in sec for MagmaLU factorization, solution and factorization+1,000 solves using CPU only, CPU+GPU and CPU+MIC, Anselm

- CP (TFETI) solutions for time dependent (plasticity) problems using explicit inverse computation by MUMPS or SuperLU in subcommunicators were compared with direct parallel solution of the CP.

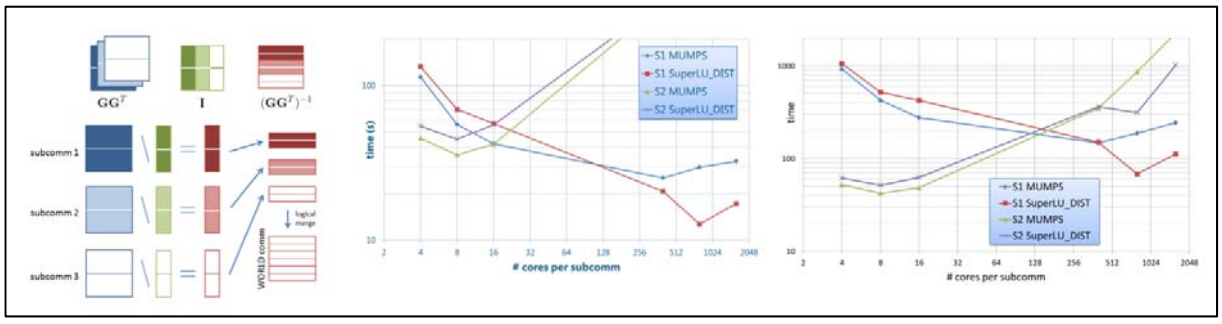

Figure 8: Scheme of explicit inverse $(GG^T)^{-1}$ computation and comparison of log of times in sec for sparse parallel direct CP solution (S1) vs. explicit inverse (S2) using MUMPS vs. SuperLU, 100 and 1,000 actions, HECToR

- Comparison of the stiffness matrix K factorization and pseudoinverse K+ application in TFETI using PETSc LU, MUMPS and SuperLU locally.

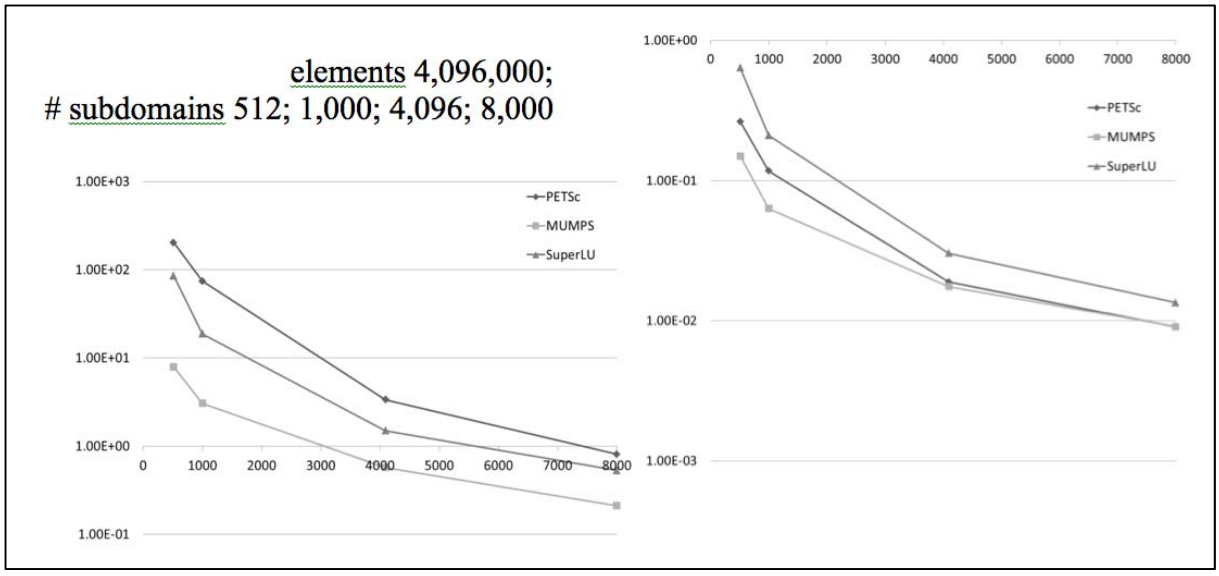

Figure 9: Parallel scalability of K factorization and K+ action, HECToR (y-axis is log of time in sec, x-axis is number of subdomains/cores)

- Comparison of KSPCG solver with the novel pipelined version KSPPIPECG implemented in PETSc 3.4 for system
    a) primal block-diagonal assembled ($A = K, b = f$),
    b) primal decomposed penalized unassembled – PFETI ($A = K + \rho B^T B, b = f$),
    c) dual decomposed projected unassembled – TFETI ($A = PBK^+B^T, b = Pd$).

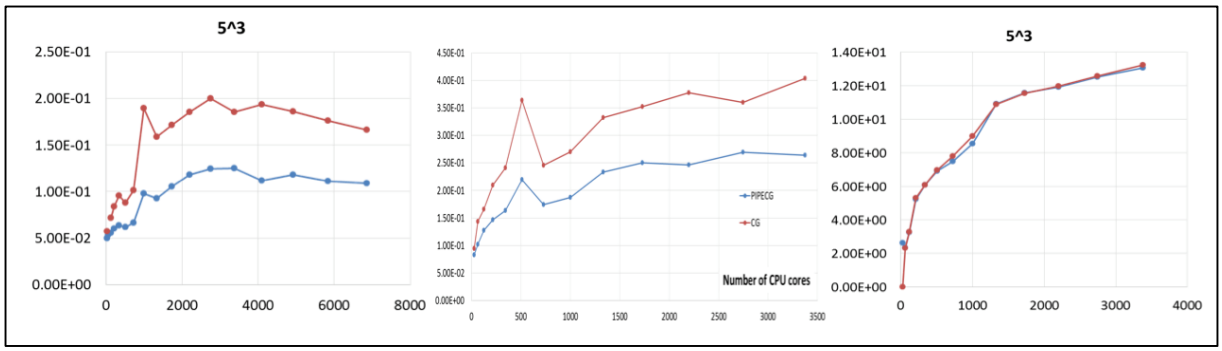

Figure 10: Performance of CG (red) vs. PIPECG (blue) for subdomain size 53 for cases a-c, Sisu (y-axis is solution time in sec for 1,000 iterations, x-axis is number of cores)

- Stiffness matrix null-space orthonormalization by means of classical Gramm-Schmidt process and its impact on the CP iterative solution, various preconditioners were tested.

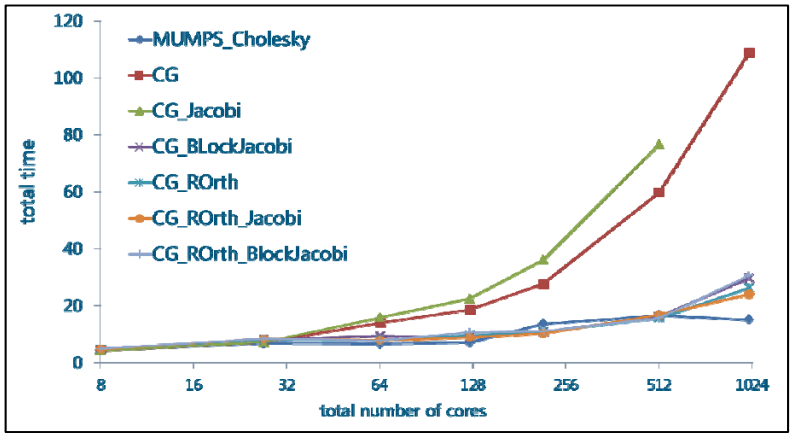

Figure 11: Comparison of the direct CP solution using MUMPS vs. iterative solution using CG without and with preconditioning.

2.10.3 Impact and Summary

- CP solution using MagmaLU on GPU and MIC accelerators improves the performance and it becomes more significant with increasing number of iterations, e.g. for the largest problem size 24,576 for 1 factorization and 1,000 solves CPU needs 279 sec, CPU+GPU 239 sec and CPU+MIC only 105 sec!
- CP (TFETI) solution for time dependent (plasticity) problems using explicit inverse computation by MUMPS or SuperLU in subcommunicators is faster for hundreds of actions than direct parallel solution of the CP.
- Stiffness matrix factorization and pseudoinverse action were the fastest with MUMPS library; SuperLU was for the factorization of the problem with the largest subdomain dimension 10 times slower and PETSc built-in LU even 25 times slower.
- PETSc implementations of KSPCG vs. KSPPIPECG were compared:
  a) purely block-diagonal assembled system, no communication in action of the system operator $A = K$, PIPECG more efficient for subdomain sizes up to 11^3
  b) PFETI – some communication in action of $A = K + \rho B^T B$, involves only SpMV, hundreds of cheap iterations, PIPECG more efficient for subdomain sizes up to 11^3
  c) TFETI – much more communication and computation in action of $A = PBK^+B^T$ involving parallel direct solve, tens of expensive iterations, PIPECG and CG exhibits highly similar behaviour for all subdomain sizes and number of MPI processes.
- Iterative solution of CP can be even comparable with the most efficient direct parallel CP solution if the null-space matrix has orthonormal columns, otherwise the convergence of the iterative solver is very poor or furthermore it diverges. The most promising is CG method with orthonormalized null-space matrix and Jacobi preconditioner. Modification of this approach for PIPECG or PIPEDCG usage is a subject of future work.
- The accomplished work was presented at:
  - EASC 2014 conference, Stockholm, Sweden, 2nd-3rd April 2014, D. Horák, V. Hapla, A. Markoupoulos, L. Říha, IT4Innovations, VSB-TUO: FLLOP: library, poster
  - PMAA 2014 conference, Lugano, Switzerland, 2nd-4th July 2014, D. Horák, V. Hapla, A. Markoupoulos, L. Říha, IT4Innovations, VSB-TUO**:** Scalability improvement of FLLOP solvers, lecture

- Modelling 2014 conference, Rožnov pod Radhoštěm, Czech Republic, 2nd-6th June 2014, V. Hapla, D. Horák, A. Markoupoulos, L. Říha, M. Čermák, L. Pospíšil, A. Vašátová, IT4Innovations, VSB-TUO: FLLOP: A Massively Parallel QP Solver Compatible with the TFETI Substructuring Scheme, lecture + poster

2.11 ALYA and Code Saturne

2.11.1 Overview and workplan

An interface for the coupling library of Code_Saturne named Parallel Location and Exchange library (PLE), was developed to be used in ALYA. This interface enables to solve multiphysics coupled problems running instances of each code.

Code_Saturne is a Computational Fluid Dynamics code based in a finite volume formulation. It is written in a mixture of C (50%), FORTRAN (37%) and python (13%) languages. The coupling library of the code, PLE, is written in C and has been used to couple different instances of Code_Saturne and/or similar codes developed by the EDF Research Department. Details of the code can be consulted in http://code-saturne.org/cms/.

ALYA is a multiphysics code developed at the Barcelona Supercomputing Center. It is written in FORTRAN language and is based on a finite element formulation. ALYA relies on a modular architecture organized in kernel, modules and services. The kernel contains the facilities required to solve any set of discretized partial differential equations, while the modules provide the physical description of a given problem. There are modules to handle several different physical problems, and in the present contribution, the module to solve thermal problems, called TEMPER, is used. It is noted that currently two modules of ALYA, the one for incompressible flow equations, called NASTIN, and the one for solid mechanics, called SOLIDZ, as well as Code_Saturne, are part of the benchmark suite of the Partnership for Advanced Computing in Europe (PRACE) http://www.prace-ri.eu/.

The coupling library to be used, PLE, is written in C language, and is able to handle communication between different running applications. PLE has two main features. First, it provides a location service. Given a set of coordinates, in one of the running applications, PLE is able to search in parallel for the cells or elements that contains (or are closest) to the points included in the set in the other application domain. Once the points are located, data structures are generated in order to be able to interpolate values in the interest locations. Second, it provides a parallel communication ability through MPI messages. PLE can send data from one running application to the other using the data structures generated in the location phase.

The interface developed to use the full capacity of PLE in ALYA is programmed in C++, taking advantage of data structure construction and manipulation provided by this language. It enables exchange of information between a Code_Saturne running simulation to a running instance of ALYA. The interface adapts the C data structures needed in PLE to the FORTRAN data structures used in ALYA. It should also be pointed out that care must be taken for the conversion from the finite volume formulation used in Code_Saturne to the finite element formulation used in ALYA. With the use of this interface, it has been possible to solve a Fluid-Thermal interaction problem using Code_Saturne for the fluid physical problem, and ALYA for the thermal physical one.

In current developments, the coupling interface, together with Code_Saturne and the SOLIDZ module of ALYA will be used to address Fluid-Structure Interaction problems (FSI), combining two of the members of the benchmarking suit for PRACE. This involves further research for robust coupling strategies, and tackling physical issues that are intrinsic to FSI

modelling. Performance and scalability tests will be carried out in order to see the impact of the coupling strategy and interface in the overall simulation.

2.11.2 Results

In the following, the results of the work done during the extension period are described.

The coupling library of Code_Saturne, PLE, was tested in code examples in order to identify the main problems to be addressed regarding the different programming languages used in the two codes to be coupled. This test phase was divided in three main parts, and was important to the later development of the final form of the interface. It is highly recommended to follow these phases for possible future code coupling applications. In the first phase, the location and communication service of PLE was used in two small codes to solve thermal problems, the first written in FORTRAN and the second written in C. In the second phase, a coupled problem was solved using two independent instances of ALYA. Using the information collected in the first test phase, the needed changes in the structure of ALYA where carried out, and a fluid-thermal interaction problem was solved. In the third phase, the final form of the interface was developed, facing the biggest challenges concerning the conversion of data structures and communicators between the two codes, and a fluid-thermal interaction was solved using Code_Saturne for the fluid domain and ALYA for the thermal domain.

The problem addressed is shown schematically in Figure 12.

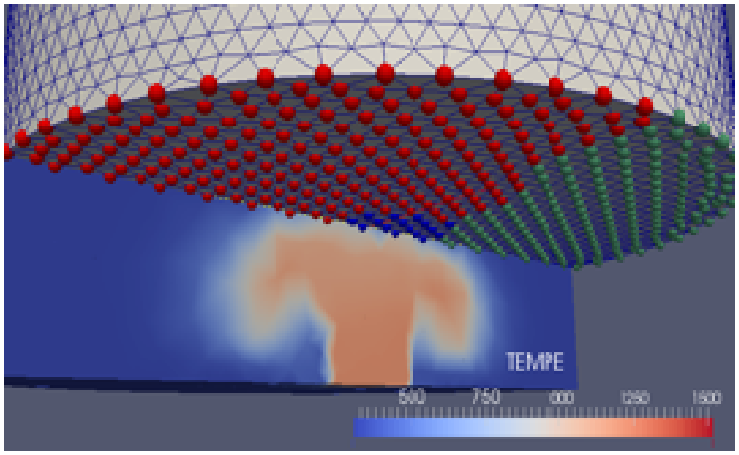

Figure 12: Schematic representation of the Fluid-Thermal interaction problem.

An impinging jet impacts a cylindrical solid heat conducting structure, the temperature of the impinging jet at the fluid-solid interface is imposed as a boundary condition to the thermal problem, meanwhile the heat flux obtained in the thermal domain is imposed as a boundary condition for the fluid problem. Both applications are run in parallel, and the full capacity of PLE is used by ALYA with the use of the developed interface, the figure also shows the points located by PLE at the fluid-solid interface.

The communications environment set up for the general coupling strategy is shown in Figure 13.

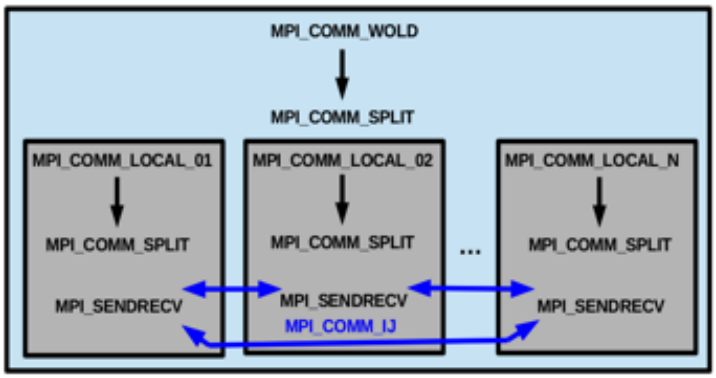

Figure 13: MPI communication environment created for general coupling problems using PLE.

The MPI_COMM_WORLD communicator is separated in applications communicators, and inter-communicators are created between the processes that have part of the coupling interface.

Typical results of the study cases for the temperature distribution in the whole domain are shown in Figure 14 and Figure 15.

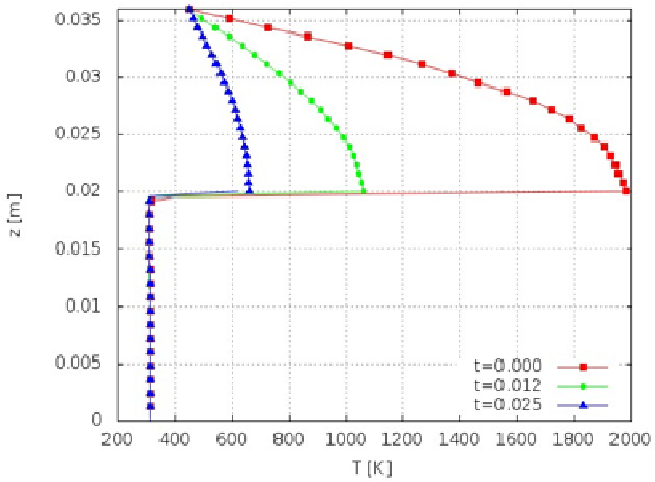

Figure 14: Temperature along the axis of the cylinder, z = 0.02m is the position of the fluid-solid interface

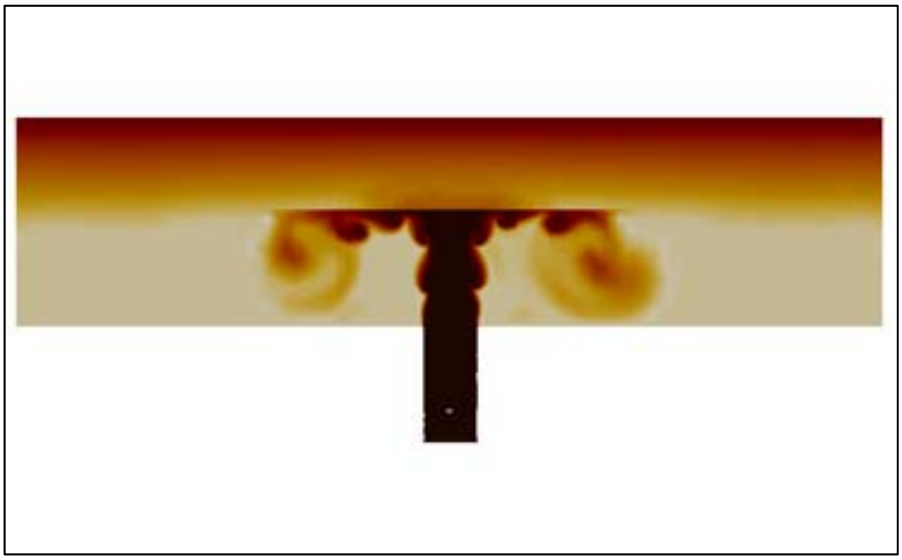

Figure 15: Temperature field at a plane parallel to the axis of the cylindrical region

2.11.3 Impact and Summary

- The PLE coupling library was used in test and study cases
- An interface for using PLE in ALYA was developed
- The interface was tested in ALYA-ALYA coupling and Code_Saturne-ALYA coupling problems for Fluid-Thermal interaction problems
- Coupling frameworks and strategies were presented in ICMEg workshop http://web.access.rwth-aachen.de/MICRESS/ICMEg1/
- Results of a Fluid-Thermal interaction problem using the interface coupling for PLE will be presented in the 10th International ERCOFTAC symposium http://www.ercoftac.org/etmm10/

2.12 PFARM

2.12.1 Overview and workplan

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 [[9]]. The package has been used to calculate electron collision data for astrophysical applications (such as: the interstellar medium, 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. The code has recently been adapted to form a compatible interface with the UKRmol suite of codes for electron (positron) molecule collisions [[10]] thus enabling large-scale parallel outer-region calculations for molecular systems as well as atomic systems. Each of the main stages of the calculation is designed to take advantage of highly optimised, numerical library routines. For efficiency, the external region calculation takes place in two distinct stages, named EXDIG and EXAS, with intermediate files linking the two. EXDIG uses a parallel domain decomposition approach where large-scale parallel eigensolver computations predominate. EXAS uses a combined functional/domain decomposition approach where good load-balancing is essential to maintain efficient parallel performance.

- PFARM developers have focused on optimizing the code for new and emerging architectures during the extension period. The PFARM code is developed in a scalable, modular style and was designed from the start to exploit a range of highly optimized numerical library routines for the computational cores of the overall calculations. These features mean that the code has the potential to exploit heavily the computational power of new many-core architectures and computational accelerator hardware.
- Initial Testing of MKL [11] and MAGMA [12] libraries on Xeon Phi architectures with offloading capabilities. Benchmarking and analysis of results and hence identification of suitable candidate PFARM code kernels for offloading to Xeon Phi accelerators (STFC).
- Development of Fortran interface routines for MAGMA MIC calls and memory management routines. Although the MAGMA for GPUs library comes complete with Fortran interfaces, the MIC version currently does not. The PFARM code suite is written entirely in modern Fortran variants and therefore it has been necessary to design interfaces to the C library routines in MAGMA in order to use them (STFC).
- Initial port of PFARM (EXDIG) and associated third-party software packages to the Intel Xeon Phi architecture (STFC 1PM). Firstly port to a single Intel Xeon Phi and then multiple Intel Xeon Phis. Taking forward work undertaken in the previous WP8 project (details above), the overall performance implications of in-situ offloading of

computational bottlenecks such as BLAS and LAPACK (e.g. eigensolvers) routines to Xeon Phi accelerator hardware is to be investigated (STFC).

- Enabling of the PFARM (EXAS) code on a Xeon Phi-based cluster. EXAS incorporates a functional decomposition of tasks within the parallel decomposition and therefore the strategy implemented here involves running entire sections (i.e functional tasks) of the code exclusively on Xeon Phi processors at the same time as other parts run on the host. This partition is achieved using "MPI symmetric" mode. MPI symmetric mode is of interest for codes already running in parallel on distributed memory systems (e.g., MPI-based codes). While the steps needed for a basic port to coprocessor-based clusters in symmetric mode is generally as trivial as the steps taken for those taken to enable the code in native mode, the complexity of load-balancing over such a heterogeneous system of compute devices can in reality be very challenging and, as evidenced by this work, often involves significant non-trivial refactoring of code (ICHEC).

2.12.2 Results

Initial work in the extension period involved testing the performance of several computationally intensive routines used by the EXDIG and EXAS codes on the Intel Xeon Phi. This was undertaken using the numerical libraries MKL v11.1 and MAGMA MIC v1.1.0 for a wide range of representative data sizes. The numerical routines were BLAS 3 (DGEMM), linear solver (DGETRF), eigensolver (DSYEVD), and singular value decomposition (DGESVD) routines [13]. All these linear algebra operations are used throughout the PFARM calculations on matrices of varying dimensions. One general pattern (which also follows intuition) that became clear from our performance analysis was that the larger the matrix, usually the more beneficial was offloading calculations to the Xeon Phi either from MAGMA MIC or MKL. The operation in PFARM that is undertaken on the largest dataset is the Hamiltonian eigensolver calculation in EXDIG and it is this operation that would normally take the most time. Sector Hamiltonian matrices setup for diagonalization here are generally of dimension 10000 to 30000 and are well into the range that demonstrate large benefits in performance from offloading to Phi [14].

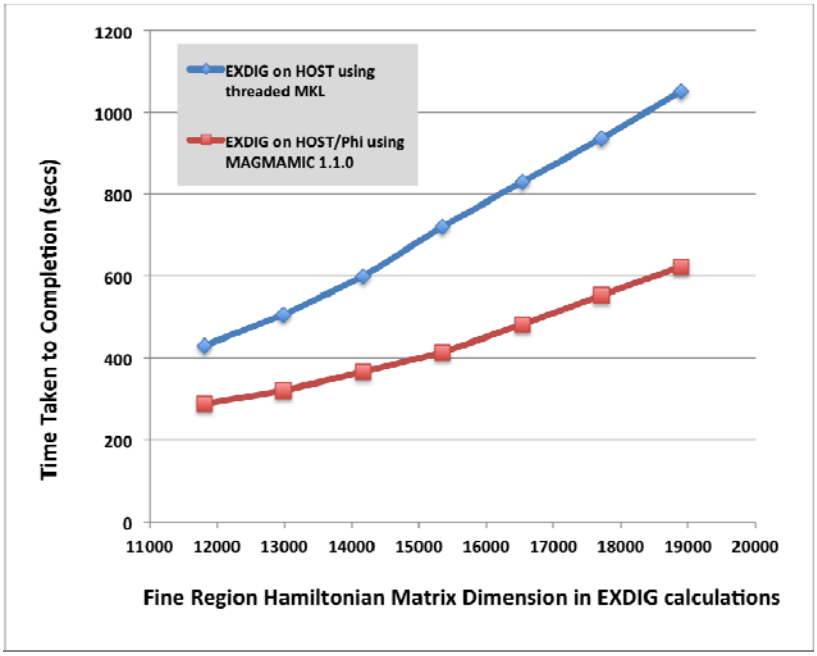

Figure 16 Performance of EXDIG code on Xeon Phi with MAGMA

Given the encouraging performance results for parallel diagonalization using MAGMA MIC, further development work in the WP8 extension period involved integrating the MAGMA MIC eigensolvers into the RMX code. As there are no Fortran wrappers currently available for MAGMA MIC, specialist interface code was developed, using the Fortran to C interoperability features available in modern Fortran. Figure 16 shows performance results for EXDIG with offloading of Hamiltonian sector matrix diagonalizations to the MAGMA MIC eigensolver routine DSYEVD for an FeII electron-atom scattering calculation with 1181 channels. Each point in Figure 16 shows the time taken for a complete calculation comprising two Hamilton sector matrix diagonalizations from the Fine region calculation, and two smaller Hamilton sector matrix diagonalizations from the Coarse region calculation, both sets with associated I/O. To generate a range of representative calculations the number of basis points in the Fine region Hamiltonian size is increased whilst the coarse region matrix dimensions remain fixed. Interface codes for the MAGMA solver and MAGMA memory management routines calls from the Fortran application have been developed using the C interoperability features available in modern Fortran. Figure 16 shows that the MAGMA Xeon Phi accelerated version is around 50% faster than the host for the smaller cases and around 70% faster than the host for the largest case. The Xeon host calculations are run using MKL v 1.1 with 32 threads and the Xeon Phi MAGMA v1.1.0 calculations use 240 threads. More details of the hardware and software environment used for testing and further benchmarks can be found at the HPCforge Wiki [13].

A schematic of the new accelerator-based implementation of the EXAS code is shown in Figure 17. The overall strategy of the new implementation of the EXAS code is that each Xeon Phi coprocessor available on the system can be exploited to process a full systolic pipeline (or possibly multiple pipelines). In the original implementation, at least 12 CPU cores would be required to process such a pipeline. Since excessive MPI communications are known to degrade performance on the Xeon Phi, much more so than on a Host CPU, we have focused on minimizing such communications on the Xeon Phi, by exploiting OpenMP parallelization across a given pipeline.

In the new implementation the code has been refactored so that, rather than have a single host process read the large (~300 MB) AMP files and then broadcast them to the processes running on the Xeon Phi, each process running on the co-processor reads from disk itself (via the PCIe bus). The original code is designed such that a given pipeline will only be built if enough MPI processes are used at runtime: the more MPI processes employed at runtime, the more pipes will be built. However, in the original implementation of the code, at least 12 MPI processes are required to build a single pipeline and so the code needed to be significantly refactored so that a full pipeline is built from only a single MPI process, but only if that process exists on the co-processor. In this way, if a co-processor is found at runtime, a full extra pipeline will be constructed, as can be seen for the example of two available co-processors shown in Figure 17. Finally, since a pipeline that is processed on the Xeon Phi no longer relies on multiple MPI processes, but rather exploits OpenMP parallelism, a new subroutine has been developed and introduced to the code, `prop_mic()`, which is only called by those MPI processes running on the co-processor.

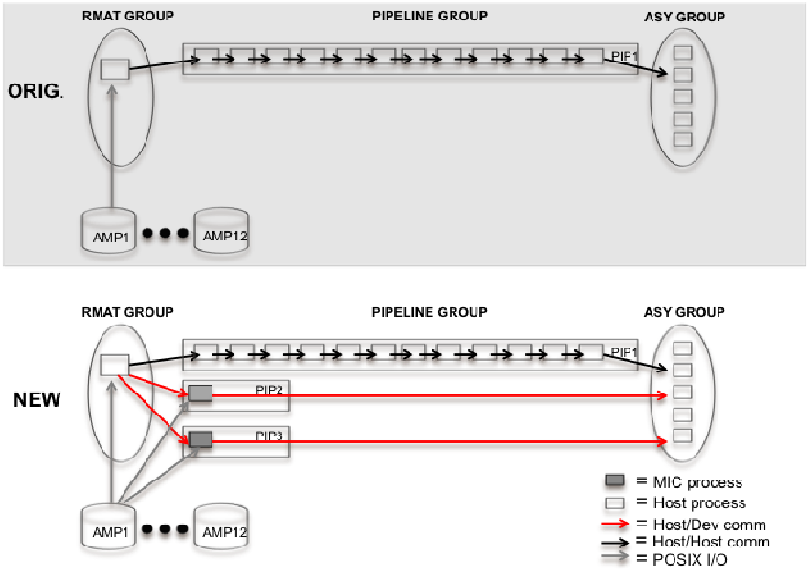

Figure 17: Schematic of original EXAS implementation for a single pipeline (top) and schematic of new EXAS implementation enabled for Xeon Phi - based clusters (bottom)

A performance analysis of the new implementation of the code compared to the original version of the code can be seen in the form of profiles from the Intel Trace Analyzer and Collector (ITAC) tool [14] in Figure 18. The same FeII test case was used as above, where collision rates for 100 electron collision energies were calculated. All development and testing was carried out on the Fionn system at ICHEC, a heterogenous machine made up of four components: Thin, Hybrid, Fat and Service. Details of the testing platform can be obtained from the HPCorge Wiki [13] or ICHEC Fionn website [15].

The left hand side of Figure 18 shows a profile of time spent in application time (as opposed to MPI time) for the original implementation of the code, where the MPI processes involved in the PIPELINE GROUP (processes 1-12) are highlighted in light blue. It can be seen that for a given process, ~565 seconds is spent in application time in the PIPELINE GROUP in the original implementation of the code (TSelf is time spent in the given function, excluding time spent in functions called from it) On the right hand side we see the profile for the same test case, but for the new implementation of code, where two co-processors are exploited on top of the host node. It can be seen that, by exploiting the two co-processors, less application time is spent in the PIPELINE GROUP (TSelf ~371 seconds), but also that the time spent in the PIPELINE GROUP for each of the co-processor MPI processes (highlighted in red) is shorter (TSelf ~216 seconds) compared to the application time spent in the PIPELINE GROUP for each of the host MPI processes, indicating the clear potential for exploiting the Intel Xeon Phi for processing the pipelines in full.

However, while the exploitation of the Xeon Phi architecture does show genuine potential in reducing time to solution of the EXAS code (as well as scalability on large-scale systems), the initially enabled code suffers from communication bottlenecks that prevent speedup of the overall code relative to the original code (the new implementation is ~1.6x slower than the original implementation due to these communication bottlenecks). Indeed, these bottlenecks have been identified and further non-trivial refactoring of the code is currently underway to reduce these communication bottlenecks

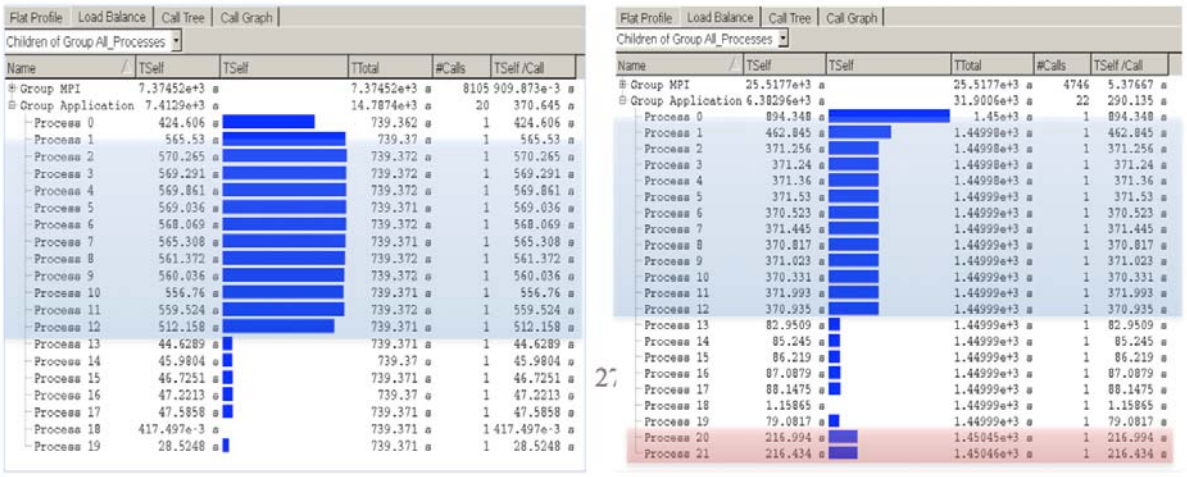

Figure 18: Performance analysis of original implementation of EXAS (left) and new implementation of EXAS on Xeon Phi (right) using the Intel Trace Analyzer and Collector (ITAC) profiler.

2.12.3 Impact and Summary

The main application enabling and performance enhancing features introduced to PFARM during the first two years of WP8 are:

- More efficient parallel ELPA [16] eigensolvers in EXDIG for large Hamiltonian matrices.
- A highly efficient scalable parallel I/O implementation for EXAS, suitable for large-scale calculations and exploiting underlying properties of the functional decomposition.
- Improved load-balancing algorithms for calculations on petascale systems.
- Initial investigation of Intel Xeon Phi performance for BLAS 3 and dense linear solver operations
- Establishment of PFARM HPCforge Wiki for detailed descriptions and analyses of enabling methodologies

The main application enabling and performance enhancing features introduced to PFARM during the WP8 extension project are:

- An NVidia GPU port of EXAS with exploitation of phiGEMM and MAGMA GEMM routines for BLAS operations.
- Development of modern Fortran interfaces for MAGMA MIC routines (The authors wish to thank Vendel Szeremi from STFC for his assistance with this stage of the work).
- Xeon Phi port of EXDIG with MAGMA MIC for accelerated eigensolvers.
- Xeon Phi port of EXAS with functional tasks (pipelines) accelerated using Xeon Phi hardware.
- An in-depth analysis of MKL and MAGMA usage and performance for a range of relevant computationally intensive linear algebra operations on the Xeon Phi ready for future integration into PFARM.

The code will be made available via the CCPForge repository [17] so that, after additional fine-tuning, the upgraded code in PFARM can be used as straightforward Xeon-Phi or GPU option in ongoing atomic and molecular physics work (from astrophysical ions to organic molecules). The developments are being made known to active members of the community via the Collaborative Computational Project CCPQ [18]: a summary of the work was presented at the CCPQ Steering Panel meeting (UCL, July 2014). More detailed presentations

will be made at forthcoming CCPQ workshops (2014-5) and at a UK-RAMP meeting to be held in autumn 2014.

After some further fine-tuning and follow-up it is expected that the documentation and benchmark reports will be upgraded to appear as peer-reviewed papers. The 'spin-off' Fortran-C/C++ interface currently developed for efficient and easy MAGMA-MIC access from standard scientific Fortran codes will have broader applications.

2.13 RAMSES

2.13.1 Overview and workplan

Ramses is an open source, Fortran 90 code for astrophysical simulations. It describes the behaviour of both baryons, represented as a fluid on the cells of an adaptive resolution mesh (Adaptive Mesh Refinement Method – AMR), and the dark matter, represented as a set of particles. The two interact via gravitational forces. Various other physical processes can be included (MHD, radiative transfer, cooling processes…). The AMR approach provides high spatial resolution only where this is actually required, thus ensuring minimal memory usage and computational effort. RAMSES' Fully Threaded Tree AMR scheme dynamically creates refinements on a cell-by-cell basis assigning to each cell the information about the neighbouring and parent cells. This allows great flexibility to match complicated flow geometries. However, it represents a challenge for GPU computing, due to complex memory access patterns.

The hydrodynamics solver was identified as the target for GPU implementation within WP8, being the most computational intensive part of the original code and, at the same time, solving a local problem, ideally fitting the accelerator's architecture. The GPU implementation is based on the OpenACC standard, which effectively supports parallel programming on hybrid CPU/GPU systems. OpenACC allows adopting an "incremental development approach", extending and optimizing progressively the fraction of the code ported on the accelerator, strongly facilitating the refactoring process and minimizing the effort for managing and maintaining the software.

The GPU refactoring required specific care in the effective and efficient management of data structures, in order to optimize memory access patterns and minimize the CPU-GPU data transfer overhead. Appropriate approaches were followed to adapt the workflow of the different algorithms to the accelerator's architecture, in order to maximise the computational throughput.

Once the GPU enabled hydrodynamic kernel was ready a final effort was done to verify if the new code architecture could be exploited also by a multithread processor adopting an OpenMP based approach (in the perspective, for instance, of a possible extension to Xeon PHI).

Translation of OpenACC to OpenMP directives was sometimes a tricky process. They both have similarities but also meaningful differences, sometimes difficult to spot at first glance. One of them is that in OpenACC parallel regions scalars (variables) and loop index variables are private by default. However, in OpenMP parallel regions loop index variables are private by default and scalars are shared. This, in turn, implicates the correctness of calculated data. Awareness of those differences was the key point to valid directives translation and to receive valid results of the parallelized program version.

2.13.2 Results

The major achievement within the WP8 extension is represented by the full OpenACC implementation of the RAMSES' hydrodynamic kernel. A number of tests and benchmarks

have been run in order to prove the correctness of the new version of the code and to analyse the performance.

The results presented in Table 3 compare the performance of the new version RAMSES, running on CPUs and on GPUs. If we focus on the hydro kernel only (T_hydro), we see that the GPU (Nvidia K20X) performs like 16 cores of a Intel Sandybridge Xeon E5-2670 CPU (comparing orig_V10_N16 and ACCyes_C1000_N1 rows). The GPU time accounts for a data copy overhead of about 25% of the total hydro time (T_copy).

Overall, the computing time of a GPU run is just 1.5 smaller than that on the CPU (orig_V10_N1 vs. ACCyes_C1000_N1 rows of the T_tot column). In fact a large part of the code is still running on the CPU in both cases. In particular, when running on the GPU, the hydro part becomes negligible and the computing time is dominated by the gravitational potential solver (see Figure 19), which is not GPU enabled. This shows how a full porting of the code on the accelerator is often crucial to achieve good speed-ups.

Cosmo 3 Levels (6-8)	T_tot	T_hydro		T_god_fine	T_copy	T_tot speedup	T_hydro speedup		T_god/ T_copy
		Sec	Percent				1 core vs 1gpu	1 cpu VS 1gpu	
orig_V10_N1	155662	56218	36.1	56218					
orig_V10_N2	75905	27625	36.4	27625					
orig_V10_N4	36147	13207	36.5	13207					
orig_V10_N8	17755	6243	35.2	6243					
orig_V10_N16	8775	2918	33.3	2918					
ACCyes_C1000_N1	104811	3009	2.9	2270	739	1.49	18.68	2.07	3.07
ACCyes_C1000_N2	49718	1425	2.9	1040	385	1.53	19.39	2.05	2.70
ACCyes_C1000_N4	23372	693	3.0	485	208	1.55	19.07		2.33
ACCyes_C1000_N8	11543	344	3.0	231	113	1.54	18.15		2.03
ACCyes_C1000_N16	5718	179	3.1	115	64	1.53	16.26		1.79

Table 3: Performance comparison of the CPU and GPU versions of RAMSES

```
|-----------------------------------------------------
| 100.0% | 8775.076792 |USER
||----------------------------------------------------
|| 17.2% | 1510.795560|godfine1_
|| 14.6% | 1277.922684|get3cubefather_
||  8.1% |  712.218697 |gauss_seidel_mg_fine_
||  6.1% |  534.160606 |interpolate_and_correct_fine_
||  6.0% |  523.049099 |make_virtual_fine_dp_
||  5.5% |  485.875381 |make_virtual_reverse_dp_
||  4.5% |  393.035509 |cmp_residual_mg_fine_
||  3.5% |  305.048373 |interpol_phi_
||  3.4% |  295.687628 |interpol_hydro_
||  3.0% |  261.753846 |unsplit_
||  2.7% |  238.224640 |cmpflxm_
||  2.1% |  182.118400 |ctoprim_
||  2.0% |  176.510382 |build_parent_comms_mg_
||  2.0% |  174.412867 |gauss_seidel_mg_coarse_
||  1.5% |  133.619786 |riemann_llf_
||  1.4% |  125.394321 |synchro_hydro_fine_
||  1.3% |  111.616723 |restrict_residual_fine_reverse_
||  1.3% |  110.802651 |uslope_
||  1.1% |   94.464831 |getnborfather_
||  1.1% |  93.612857 |interpolate_and_correct_coarse_
||  1.1% |  92.368937 |make_virtual_mg_dp_
||  1.0% |  90.738933 |get3cubefather_godfine_
||  1.0% |  85.241909 |make_fine_bc_rhs_
|=====================================
```

```
|-----------------------------------------------------
| 100.0% | 5718.250985 |USER
||----------------------------------------------------
||  22.3% | 1276.678459|get3cubefather_
||  12.8% |  732.494081 |gauss_seidel_mg_fine_
||   9.4% |  536.284852 |interpolate_and_correct_fine_
||   9.1% |  522.834264 |make_virtual_fine_dp_
||   6.9% |  394.130303 |cmp_residual_mg_fine_
||   5.3% |  304.073254 |interpol_phi_
||   3.1% |  176.423148 |gauss_seidel_mg_coarse_
||   3.1% |  176.036009 |build_parent_comms_mg_
||   2.3% |  129.439764 |make_virtual_reverse_dp_
||   2.2% |  125.986470 |synchro_hydro_fine_
||   2.0% |  111.681912 |restrict_residual_fine_reverse_
||   1.7% |   95.785096 |interpolate_and_correct_coarse_
||   1.6% |   92.444358 |make_virtual_mg_dp_
||   1.5% |   87.124500 |make_fine_bc_rhs_
||   1.5% |   83.867781 |cic_cell_
||   1.0% |   56.125077 |gradient_phi_
||   1.0% |   54.569618 |courant_fine_
|=====================================
```

Figure 19: Performance profiles of the CPU (left) and GPU (right) runs. The functions in red in the left panel are all part of the hydrodynamic solver. They disappear in the right panel, showing that the GPU efficiently accelerates the solver. On the GPU, the computing time is dominated by functions related to the gravitational solver.

The new OpenMP (hereafter OMP) implementation was tested comparing the results in four primary configurations: no MPI and no OMP, no MPI with OMP, MPI without OMP and MPI with OMP.

To check what is the main impact of work done on application scalability the primary goal was to compare single node OMP and non-OMP versions elapsed times. However it was also interesting to check how the application behaves and scales in hybrid MPI and OMP version. The results, presented in below and in Figure 20, are promising but require further work in order to find other spots in the source code to be possibly optimized. Overall we manage to lower the execution time from around 300 seconds for basic setup to around 10 seconds using 8 nodes with 2 task each (8 OMP threads per task).

OpenMP only tests:

- 1 Thread: 304 sec.
- 4 Threads: 110 sec.
- 16 Threads: 112 sec.

MPI only tests

Elapsed	Task desc.
282	1 Tasks, 1 per node
127	2 Tasks, 1 per node
78	4 Tasks, 1 per node
41	8 Tasks, 1 per node
37	8 Tasks, 2 per node
29	16 Tasks, 2 per node

MPI + OpenMP

Elapsed	Task desc.
134	2 Tasks, 1 per node, OMP_NUM_THREADS=1
48	2 Tasks, 1 per node, OMP_NUM_THREADS=4
74	4 Tasks, 1 per node, OMP_NUM_THREADS=1
26	4 Tasks, 1 per node, OMP_NUM_THREADS=4
42	8 Tasks, 1 per node, OMP_NUM_THREADS=1
20	8 Tasks, 1 per node, OMP_NUM_THREADS=4
18	16 Tasks, 2 per node, OMP_NUM_THREADS=4
13	16 Tasks, 2 per node, OMP_NUM_THREADS=8

Figure 20: Tests of the new implementation of the OpenMP code in various hybrid (MPI+OMP) configurations.

2.13.3 Impact and Summary

RAMSES refactoring work was performed in close synergy between CSCS and the numerical astrophysics group of the University of Zurich. This way, the software design involved the HPC experts and the main code developers, who also progressively verified the results of each implementation stage. This way, the kernels implemented in the work package became part of the standard development process, without requiring special acceptance procedures.

The new RAMSES implementation is available to the community through the WP8 web site (http://hpcforge.org/plugins/mediawiki/wiki/ramses/index.php/Downloads) and is being integrated in the main RAMSES GitHub repository (https://github.com/samgeen/ramses). The new features were presented to the community members at the annual Ramses User Meeting (RUM 2014, Paris, June 2014) and in a number of local and international events, like:

- the GPU Technical Conference 2014 (San Jose, USA, March 2014), with the talk "RAMSES on the GPU: An OpenACC-Based Approach" (Gheller, Rosiliho de Souza, Teyssier, Wetzstein),
- the Conference of Computational Physics (Boston, USA, August 2014), with the talk "Cosmology on the GPU: Enzo and Ramses in action" (Gheller, Teyssier, Vazza, Wang),

- the Perspectives of GPU Computing in Physics and Astrophysics Conference (Rome, September 2014) with the talk "The Ramses code for GPU accelerated numerical cosmology" (Gheller).

3 Discussion and Conclusions

WP8 working methodology was based on the concept that the process of code design and development had to be driven by scientific applications, towards the effective usage of the next generations of high performance computing systems.

This was accomplished by involving the scientific communities in the process of enabling some of their most relevant simulation codes on innovative supercomputing architectures. This process relied on a close synergy between scientists, code developers and HPC experts, the first two contributing with their deep understanding of the research area and of the algorithms, the latter providing the necessary skills and competencies for exploiting novel hardware architectures, programming models and software solutions.

Besides improved codes, WP8 produced extensive knowledge on the current methodologies and models to develop numerical applications on HPC architectures. This was built upon the daily experience of the different developers teams and discussed and consolidated through a series of Face-to-Face (F2F) workshops, where scientists, software developers and HPC experts met and discussed 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.

During the first two years of WP8 four workshops were organised: at CSCS (October 18-19, 2011), BSC (February 2-3, 2012), Paris-Saclay (Maison de la Simulation, June 11-13, 2012) and again at CSCS (March 6-8, 2013). A fifth workshop was organised during the extension, at CINECA, on April 28-29, 2014, to present and discuss the main results achieved during the third year.

In the rest of the section we will summarise the most relevant outcomes matured during this three years process, which can serve as guidelines or best practices for similar future endeavours.

General topics

1. **Science and computing.** Codes must be developed in close collaboration with the end users. This is particularly true for scientific applications, where the quality of the results can never be sacrificed to performance improvement. Codes refactoring has to be carried out with the help of the scientists, who usually know the features of all adopted algorithms in detail. Scientists have also to drive the validation process, that has to be accomplished in a rigorous way, according to the their requirements and expectations. This is the only way to produce useful software and, even more critical, to have such software accepted and adopted by the scientific users.
2. **A "brute force approach" is not sufficient.** Sometimes the increasing computing power of HPC systems leads the developers to try decrease the time-to-solution just customising the code to exploit the underlying hardware and software architecture. However, often, effective and efficient solutions can be implemented studying the details of the algorithms

and improving them according to more general solutions, often portable among different architectures, increasing the long-term impact of the accomplished work.

3. **It is not only about performance.** For sure, code re-design and refactoring must be addressed to exploit novel architectures and get better and better performance. However, code portability and maintainability are both crucial aspects to consider. Portability implies both the possibility of running with minor changes, possibly affecting only the building (configure, compile and link) chain, on the largest possible variety of HPC systems, and the possibility of attaining comparable performance on a broad spectrum of different architectures. Portability can be contrasted with extreme performance that usually requires the code customisation on the specific device. However, this issue can be addressed by adopting proper, although not straightforward, programming models (templates, domain languages etc.) or pre-processing based solutions. Maintainability is a further important requirement, which encompasses a number of different aspects, like the adoption of proper programming languages and well defined and suitable programming models, the production of detailed documentation, the definition of rules for names and symbols, the usage of versioning systems (especially for large developers teams), etc. Maintainability is often underestimated, but it becomes a serious concern as soon as the code switches to a "community code".

HPC topics

1. **MPI+OpenMP.** Threaded versions of a MPI code represent an effective solution that allows an efficient exploitation of current HPC architectures. This is usually implemented by the adoption of the OpenMP paradigm, although other solutions (essentially P-threads) can be envisaged, in the case extreme performance tuning is required (at the expenses, once more, of portability). Such an approach allows to scale up to large configurations (tens or hundreds thousands cores), delegating the CPU inter-core shared memory management to OpenMP. At the same time, it alleviates the problems related to the decreasing amount of memory available per core, by exploiting the shared memory, and obviating the need of variables copies within a CPU.
2. **Do not reinvent the wheel.** Libraries should be used wherever possible. They provide a number of different functionalities and solutions that can be adopted for standard code's components (like FFTs, basic linear algebra etc.). In this way, the developers can focus on the most specific and peculiar parts of the code. In addition, libraries are highly optimised both from an algorithmic point of view, and in order to exploit efficiently the different available HPC architectures (unless extremely "exotic"). Note that the usage of libraries is not for free. Some code refactoring is always required, in particular concerning data structures, which sometimes can result to be particularly "painful".
3. **I/O is becoming a major issue.** Increasing CPU performance and HPC systems complexity makes the data reading and writing stages a critical component of the applications. A simple approach like delegating a single processor all the I/O operations leads to huge performance penalties, while asking each processor to write a different file leads to an unmanageable number of files. The adoption of more sophisticated approaches, based on low-level (e.g. MPI I/O) or high-level (e.g. HDF5 or ADIOS) libraries is necessary in order to ensure reasonable performance, while preserving the usability of output. A positive side effect of the aforementioned high-level libraries is that the resulting file is self-explanatory (i.e. contains all the information needed to access the data therein) and can host metadata describing the data itself (physical meaning, units, etc.). In high-end cases, however, this is not enough. The data access procedures have to be tuned on the hardware, in order to fully exploit its architectural characteristics. This can be done, for example, mapping the I/O algorithms on the system topology (as in the case

of the BlueGene machines) or using innovative devices, like solid-state disks (SSD), which, however, require specific drivers (which are not always available) to be fully exploitable.

4. **From Multi to Many.** There is a growing evidence that, due to production costs, technological limits and power consumption issues, standard multi-core CPU architectures are not sufficient, alone, to "build" the next generations of supercomputers. Accelerators represent the current path towards exascale computing, providing high computational power with low energy consumption. The underlying idea is to use accelerators for computational demanding kernels, using the CPU to take care of all the other code components, managing communication and running the OS. Different kinds of accelerators are available (the must popular currently being GPUs from Nvidia and AMD, and MIC from Intel), all based on the idea of having "many" (a vague quantification, dependent on the adopted architecture) computational engines, or cores, working concurrently on contiguous data chunks. Of course, not all the algorithms can match this kind of approach (e.g. calculation of gravitational forces requires for each point of the computational domain the access to all the other points). Hence, accelerators cannot be as general-purpose as CPUs. In some cases, we have algorithms that cannot run efficiently on any accelerator. In some other cases, a complete refactoring of the algorithm is necessary in order to run it with reasonable performance on some devices.

Accelerators

1. **Accelerators for HPC.** Although different kind of accelerators from various vendors are available (GPU, MIC, FPGA, Cell), only Nvidia GPUs and Intel MIC, seem, at the moment, to be plausible solutions for HPC. They are being integrated into supercomputing solutions, support well-defined programming models (CUDA and OpenACC for the former, OpenMP for the latter), and have a well defined roadmap for the next years.
2. **Code refactoring.** All the accelerator-based solutions implement the many-cores approach, which, in general, requires some refactoring of the codes. However, thanks also to the growing vector capabilities of the CPUs, the refactoring is often beneficial also for the non-accelerated code, improving the performance on single CPU or pure MPI runs.
3. **GPU vs. MIC.** The GPU is at the moment the most mature architecture, providing good performance on a broad spectrum of applications. The number of codes able to exploit the GPU is increasing (after a slow initial start-up). It is now well understood that the maximum achievable performance is on average 2-3 times that of a typical high-end CPU, only in a few cases getting performance ratios of the order of 7-8.
   In the WP8 experience, the MIC proved to be still limited by a few architectural issues that should be solved in the coming generation of Xeon PHI, the main problem being related to memory access. However, for some applications the MIC already provides performance similar to those of the GPU.
4. **Programming models.** The main interesting feature of the MIC is represented by the programming model that relies on the OpenMP standard. For this reason the MIC is expected to be more easily programmable than the GPU. However, performance tuning requires, in general, a refactoring effort comparable to that of the GPU. Note that OpenMP allows the code to be portable across any shared memory architecture.
   The Nvidia GPUs support CUDA and the OpenACC standard. CUDA is adopted whenever maximum performance has to be achieved. However, it has a few major drawbacks. First, it is not easy to use (especially for Fortran programmers). Second, it leads to two parallel versions of the same code, one for the CPU and one for the GPU, which is difficult to maintain. Finally, but most importantly, it is not portable, being available only on Nvidia architectures. This strongly limits the portability of the code. The

OpenACC solution is more programmer-friendly and it is similar to OpenMP. It is based on directives, compatible with both Fortran and C/C++ and supports incremental development of the GPU version. Furthermore, it allows a single version of the source code. The main drawback is in terms of performance, which is, on average, 20-30% lower than that achievable with CUDA. OpenACC is an open standard supported by various compilers (PGI, CRAY, gcc version is expected by the end of 2014) and accelerators, hence portability should not represent a major issue. At the moment, however, the standard is not fully defined or mature. It is also not fully adopted by the different implementations, leading to unexpected failures at compile or run time for different compilers. The OpenACC and the OpenMP standards were expected to converge in the near future, but at the moment, this seems to be unlikely.

5. **OpenCL.** The OpenCL standard represents a further solution for accelerators programming. This solution has not actually been explored in detail during WP8 (only one code, namely EAF-PAMR adopted this solution), since only recently the standard became sufficiently reliable and performing to be competitive with CUDA, with which it shares the same degree of complexity. Furthermore, only the old 1.0 OpenCL standard is supported by Nvidia, which tends to promote its proprietary products. For these reasons OpenCL was not widely adopted, but it remains an interesting option to explore, especially for the portability of the resulting code.
6. **Standardisation of the analysis of the results.** When accelerators are used, the results, both in terms of performance and in terms of final outcomes of the run, are tricky to analyse and interpret. For performance, the main issue is represented by what we compare to what. It is not uncommon to see in the same context (e.g. a conference or a workshop) or even in the same presentation, comparisons of a GPU vs. a single core or vs. a full CPU, or comparison of the GPU code vs. the original code or vs. the refactored code. A fair and standard comparison method should be defined in order to understand clearly and promptly the meaning of the presented figures.
   Also the interpretation of the outcomes of the code can be difficult. Due to the different architectures, CPU and GPU produce in general different results. The possibility of bitwise comparison is not supported by any of the above standards, hence the criteria for code validation have to be defined. However, such criteria change with the application and they have to be defined by the experts from the scientific community.
7. **Advanced topics.** Advanced features for GPU usage are still largely unexplored. For instance, the asynchronous model, based on CUDA streams or OpenACC async regions, could in principle overlap intensive computation, on the GPU, to data transfer, between GPU and CPU or among different CPUs. Though this feature is available (and already successfully adopted by some codes), its effective usage is not straightforward, since a proper load balancing between CPU and GPU can be tricky to achieve. Strong scalability of the GPU code results in general to be worse than that of the original MPI CPU code. This is due to the decreasing data size per processor (that is, per GPU) with increasing number of processors, which reduces the computational efficiency of the accelerator. The adoption of a direct communication model among GPUs could help in minimising the MPI communication overhead and in efficiently feeding the different GPUs, especially when the computation is highly unbalanced. Currently GPUDirect model deployed by Nvidia supports direct communication, but the reliability and the performance of such solution have still to be investigated. Furthermore, the limited portability of this approach makes it unattractive.

4 Summary

The WP8 one year extension has focused on a subset of the codes developed in the first two years of WP8, further enhancing the accomplished refactoring work. The main results for each application are summarised in the following table:

Code name	Main results
EAF-PAMR	• Implementation of the OpenCL version of the MHD PPM algorithm • Implementation of the OpenCL version of the MHD TVD algorithm
OASIS	• Completion of scaling work, optimal configurations found for EC-Earth climate runs
I/O Services	• Implementation of XIOS interface for IFS • Implementation of GRIB interface for XIOS
ICON	• OpenACC version of ICON finalized, presented and accepted by the users community
Fluidity/ICOM	• PETSc DMPlex has been integrated in Fluidity. • Fluidity is now able to deal with ExodusII mesh formats. • Further optimized the hybrid MPI/OpenMP code for large ocean test case.
Quantum ESPRESSO	• Enhancement of the parallelization layers (cfr. Band parallelization, multithreading, etc.) • Further modularization of the common kernels • Implementation of ELPA libraries • Porting to the MIC platform • EXX benchmarking
SIESTA	• Checkpointing capabilities improved • Basic I/O performance improved • Parallel I/O based on HDF5 implemented
EXCITING/ELK	• Porting the setup of the Hamiltonian and overlap matrices to GPU using vendor's GPU-aware PBLAS library • Incorporation of a new GPU-enabled distributed generalized eigenvalue solver • Porting summation of the interstitial charge density to GPU using CUDA FFT. • The domain specific LAPW library is fully integrated back into the original Exciting code
PLQCD	• Re-factoring the PLQCD for the Intel MIC. This included Re-design of the lattice field data layout to benefit from vectorization on the MIC and implementation of these codes using compiler intrinsics. • Preliminary work on the combination of GMRES-DR and BiCGStab iterative solvers for tmLQCD code.

ELMER	• Coarse problem solution using
	  o MagmaLU on GPU and MIC accelerators improves the performance
	  o explicit inverse computation by MUMPS or SuperLU in subcommunicators is faster for hundreds of actions
	  o iterative solver can be comparable with the most efficient direct parallel solvers if the null-space matrix has orthonormal columns
	• Stiffness matrix factorization and pseudo-inverse action were the fastest with MUMPS library
	• KSPCG vs. KSPPIPECG were compared, for dual TFETI no improvement, for primal TFETI PIPECG more efficient for subdomain sizes up to 11^3

ALYA/CODE_SATURNE	• The coupling library of Code_Saturne has been used successfully inside ALYA • The code coupling for ALYA-ALYA and Code_saturne-ALYA has been obtained • Test cases for Fluid-Thermal interaction problems have been run • Strategies to obtain successful code coupling are proposed
PFARM	• Optimised Intel Xeon Phi port of PFARM (EXDIG) incorporating MAGMA MIC for accelerated parallel eigensolvers. • A new version of PFARM (EXAS), restructured for accelerated R-matrix propagation pipelining. Tested and tuned on the Intel Xeon Phi and also applicable to GPUs. • Detailed analyses of MKL and MAGMA MIC numerical library routines performance on Intel Xeon Phi architectures. • A new version of PFARM (EXDIG) with threaded eigensolvers appropriate for a wide-range of shared memory platforms.
RAMSES	• Porting of the Hydro kernel on the GPU adopting the OpenACC standard completed. • New OpenMP implementation of the hydro kernel, exploiting the work done for the GPU.

Besides the outstanding technical results, a major achievement of WP8 is represented by the extensive and effective collaboration between HPC experts, code developers and scientists, which resulted in establishing durable (beyond the extent of the PRACE-2IP project) synergies. The strong commitment of the involved scientific communities was the key to the success of WP8. The experience gained during the three years of the work package provides valuable guidelines for high performance scientific code development.