SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

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

PRACE-1IP

PRACE First Implementation Phase Project

Grant Agreement Number: RI-261557

D7.1.3
Report on Application Enabling for Capability Science in the MIC Architecture

Final

Version: 1.0
Author(s): Xu Guo, EPCC
Date: 13.12.2013

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="6">PRACE Project</td><td colspan="2">Project Ref. №: RI-261557</td></tr>
<tr><td colspan="2">Project Title: PRACE First Implementation Phase Project</td></tr>
<tr><td colspan="2">Project Web Site: http://www.prace-project.eu</td></tr>
<tr><td colspan="2">Deliverable ID: < D7.1.3></td></tr>
<tr><td colspan="2">Deliverable Nature: <DOC_TYPE: Report ></td></tr>
<tr><td>Deliverable Level:
PU</td><td>Contractual Date of Delivery:
31 / December / 2013
Actual Date of Delivery:
30 / December / 2013</td></tr>
<tr><td colspan="2">EC Project Officer: Leonardo Flores Añover</td></tr>
</table>

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

Document Control Sheet

<table>
<tr><td rowspan="6">Document</td><td colspan="2">Title: <Report on Application Enabling for Capability Science in the MIC Architecture></td></tr>
<tr><td colspan="2">ID: <D7.1.3></td></tr>
<tr><td>Version: <1.0></td><td>Status: Final</td></tr>
<tr><td colspan="2">Available at: http://www.prace-ri.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word 2007</td></tr>
<tr><td colspan="2">File(s): D7.1.3.docx</td></tr>
<tr><td></td><td>Written by:</td><td>Xu Guo, EPCC</td></tr>
</table>

Authorship	**Contributors:**	Cevdet Aykanat, Bilkent Raúl de la Cruz, BSC Judit Gimenez, BSC Jorge Rodriguez, BSC Mariano Vázquez, BSC Evgeny Votyakov, CASTORC Andrew Emerson, CINECA Vittorio Ruggiero, CINECA Francesco Salvadore, CINECA Eric Boyer, CINES Mikko Byckling, CSC Jussi Enkovaara, CSC Iain Bethune, EPCC Alexander Schnurpfeil, FZJ Georgios Goumas, GRNET/NTUA Ioannis Venetis, GRNET Florian Seybold, HLRS Michael Lysaght, ICHEC Momme Allalen, LRZ Gilbert Brietzke, LRZ Volker Weinberg, LRZ Damyan Grancharov, NCSA Desislava Ivanova, NCSA Elena Lilkova, NCSA Stoyan Markov, NCSA Valentin Pavlov, NCSA Peicho Petkov, NCSA Evghenii Gaburov, SURFSara Ahmet Duran, UHeM-ITU David Horak, VSB
	Reviewed by:	Manuel Fiolhais, UC-LCA; Dietmar Erwin, FZJ
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	1/December/2013	Draft	
0.2	4/December/2013	Draft	Modifications based on task internal comments. Submitted for project reviews.
1.0	13/December/2013	Final version	Revised based on review comments.

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] http://www.prace-ri.eu
[2] http://www.green500.org/
[3] http://www.hpc.cineca.it/hardware/eurora
[4] http://events.prace-ri.eu/conferenceDisplay.py?confId=181
[5] M. Casas, R.M. Badia and J.Labarta. *Automatic Analysis of Speedup of MPI Applications.* ICS'08.
[6] Introduction to the Xeon Phi Programming Model, Fabio AFFINITO, CINECA
[7] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, I. Yamazaki, SuperLU Users' Guide, 1999, update: 2013
[8] A. Duran, M.S. Celebi, M. Tuncel and B. Akaydin, Design and implementation of new hybrid algorithm and solver on CPU for large sparse linear systems, PRACE, PN: 283493, PRACE-2IP white paper, Libraries, WP 43, July 13, 2012
[9] M.S. Celebi, A. Duran, M. Tuncel and B. Akaydin, Scalable and improved SuperLU on GPU for heterogeneous systems, PRACE, PN: 283493, PRACE-2IP white paper, Libraries, WP 44, July 13, 2012
[10] http://www.mpa-garching.mpg.de/gadget/
[11] http://www.cs.uoregon.edu/research/tau/home.php
[12] J. Jeffers, J. Reinders. Intel Xeon Phi Coprocessor High-Performance Programming. Morgan Kaufmann, 2013 ISBN 978-0-12-410414-3
[13] Heinecke et al., Optimized Kernels for Large Scale Earthquake Simulations with SeisSol, an Unstructured ADER-DG Code, poster at SC'13
[14] W. F. Spotz and G. F. Carey, Formulation for the 3D Poisson Equation," Numerical Methods for Partial Differential Equations, Vol. 12, No. 2, 1996, pp. 235-243.
[15] Laurence Tianruo Yangyz, Richard P. Brent, Improved BiCGStab Method for Large and Sparse Unsymmetric Linear Systems on Parallel Distributed Memory Architectures (http://maths-people.anu.edu.au/~brent/pd/rpb206.pdf)
[16] Intel Xeon Phi Coprocessor High-Performance Programming, Jim Jeffers, James Reinders; Morgan Kaufmann (an Elsevier imprint), 2013
[17] Van der Vorst, H. A. (1992). "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems". SIAM J. Sci. and Stat. Comput. **13** (2): 631–644.
[18] https://wiki.fysik.dtu.dk/gpaw/install/MIC/eurora.html.
[19] R. Yokota, L.A. Barba, "A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems", *Int. J. High-perf. Comput*
[20] R. Yokota, J.P. Bardhan, M.G. Knepley, L.A. Barba, T. Hamada "Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns", *Comput. Phys. Commun.*,182(6):1271–1283 (2011)
[21] http://www.bu.edu/exafmm
[22] https://bitbucket.org/rioyokota/exafmm-dev
[23] J. Jeffers, J. Reinders. Intel Xeon Phi Coprocessor High-Performance Programming. Morgan Kaufmann, 2013 ISBN 978-0-12-410414-3
[24] M. Bolten and F. Fahrenberger and R. Halver and F. Heber and M. Hofmann and I. Kabadshow and O. Lenz and M. Pippig and G. Sutmann. ScaFaCoS, C subroutine library. http://scafacos.github.com
[25] http://www.scafacos.de
[26] L. Greengard, V. Rokhlin. A Fast Algorithm for Particle Simulations. J. Comput. Phys. 73 (1987) 325
[27] E. Gallicchio and R.M. Levy, J. Comput. Chem. 25(4) 479-499 (2004).
[28] E. Gallicchio, K. Paris and R.M. Levy J. Chem. Theory Comput. 5(9), 2544–2564 (2009).

[29] Jeffers, Reinders, Intel Xeon Phi Coprocessor High Performance Programming, 1st Edition, 2013, Morgan Kaufmann.

List of Acronyms and Abbreviations

BLAS	Basic Linear Algebra Subprograms
BSC	Barcelona Supercomputing Center (Spain)
CEA	Commissariat à l'Energie Atomique (represented in PRACE by GENCI, France)
CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CINES	Centre Informatique National de l'Enseignement Supérieur (represented in PRACE by GENCI, France)
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CSR	Compressed Sparse Row (for a sparse matrix)
CUDA	Compute Unified Device Architecture (NVIDIA)
DGEMM	Double precision General Matrix Multiply
DP	Double Precision, usually 64-bit floating point numbers
EC	European Community
EESI	European Exascale Software Initiative
EPCC	Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
EPSRC	The Engineering and Physical Sciences Research Council (United Kingdom)
ETHZ	Eidgenössische Technische Hochschule Zuerich, ETH Zurich (Switzerland)
FFT	Fast Fourier Transform
FMPS	Finite Method Programming System
FPGA	Field Programmable Gate Array
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)
GDDR	Graphic Double Data Rate memory
GENCI	Grand Equipement National de Calcul Intensif (France)
GFlop/s	Giga (= 10^9) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GigE	Gigabit Ethernet, also GbE
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPU	Graphic Processing Unit
GRNET	Greek Research & Technology Network
HLRS	High Performance Computing Center Stuttgart
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HT	HyperTransport channel (AMD)

IB	InfiniBand
IBA	IB Architecture
IBM	Formerly known as International Business Machines
I/O	Input/Output
JSC	Jülich Supercomputing Centre (FZJ, Germany)
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MGS	Modified Gram-Schmidt
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MKL	Math Kernel Library (Intel)
Mop/s	Mega (= 10^6) operations per second (usually integer or logic operations)
MPI	Message Passing Interface
MPP	Massively Parallel Processing (or Processor)
NUMA	Non-Uniform Memory Access or Architecture
OpenCL	Open Computing Language
Open MP	Open Multi-Processing
OS	Operating System
PGI	Portland Group, Inc.
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
RAM	Random Access Memory
SARA	Stichting Academisch Rekencentrum Amsterdam (Netherlands)
SIMD	Single Instruction Multiple Data
SP	Single Precision, usually 32-bit floating point numbers
SPH	Smoothed Particle Hydrodynamics
TB	Tera (= 240 ~ 1012) Bytes (= 8 bits), also TByte
TFlop/s	Tera (= 1012) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
UHeM - ITU	National Center for High Performance Computing, Istanbul Technical University (Turkey)
VSB	Technical University of Ostrava (Czech Republic)

Executive Summary

The PRACE-1IP project was extended by eight month to complete the evaluation of advance prototypes following a recommendation from the external reviewers. This opened the opportunity to assess if the new Intel MIC architecture can improve the performance of applications in use by European scientists. During the PRACE-1IP extension period (M34-M42), Task T7.1 "Applications Enabling for Capability Science" in Work Package 7 (WP7) focused on the application enabling support for the capability science projects on the Xeon Phi accelerators. 17 PRACE partners from 14 countries were involved in the T7.1 technical work and a total of 27 enabling projects on Xeon Phi were supported during this period. This deliverable reports on the work done and outcomes delivered by T7.1 during the PRACE-1IP extension period.

The main Xeon Phi prototype used for the enabling projects was EURORA at CINECA. Additional resources were provided on the Xeon Phi rack of the MareNostrum system at BSC. 29 proposals were received for the Xeon Phi enabling work via the PRACE Preparatory Access Type C call and an additional WP7 internal call, and 27 of the received proposals were accepted. The enabling support for all the accepted projects will complete by the end of PRACE-1IP extension period, i.e. 31 December 2013.

T7.1 provided the technical reviews for all the received proposals for Xeon Phi enabling projects, including the home site reviews and the type C reviews. Each accepted Xeon Phi enabling project was assigned to a PRACE expert who was responsible for the contact and technical support for the project. The progress of enabling projects was monitored and discussed at the monthly telcons and the WP7 Face-to-Face meetings.

In order to provide assistance for users to use EURORA, a summer school on enabling applications on Xeon Phi was organised by CINECA on 8-11 July 2013 and 34 applicants participated in this training event.

Besides this deliverable, further outcomes delivered by T7.1 in the PRACE-1IP extension period include a series of white papers which will describe more technical details of the enabling work done on the Xeon Phi accelerators. The white papers will be available for publication on the PRACE-RI web site [1] in January 2014.

1 Introduction

Task T7.1 "Applications Enabling for Capability Science" in WP7 of PRACE-1IP provides applications enabling support for capability science projects. During the PRACE-1IP extension period (M34-M42), i.e. April – December 2013, T7.1 focused on the enabling support for the capability science projects on the Xeon Phi accelerator, primarily using EURORA, the PRACE prototype at CINECA. This deliverable summarises the work done by T7.1 during the PRACE-1IP extension period and also reports on the outcome of each enabling projects on Xeon Phi.

Section 2 gives an overview introduction on the work done by T7.1 during the PRACE-1IP extension period, including a brief introduction on the Xeon Phi prototypes used (Section 2.1), the calls for the Xeon Phi enabling proposals (Section 2.2), the technical review process (Section 2.3), the PRACE expert assignment and the enabling work provided by the PRACE experts (Section 2.4), the task progress monitoring (Section 2.5), a summary of the Xeon Phi training at CINECA (Section 2.6), and the outcomes delivered by T7.1 during the PRACE-1IP extension period (Section 2.7).

Section 0 gives further details about the technical support provided for the Xeon Phi enabling projects. Each project has a brief report on the project objectives, work done and results.

Section 4 is the summary for this deliverable.

The white papers to be produced for the Xeon Phi enabling projects are listed in the Section 5 Annex.

2 Overview of T7.1 in the PRACE-1IP Extension Period

In the extension period of PRACE-1IP, T7.1 continued providing the enabling support for the capability science projects on PRACE systems and focused on the enabling work on the Xeon Phi accelerators. 17 PRACE-1IP partners from 14 countries were involved in the task and a total of 27 Xeon Phi enabling projects were supported. This section provides an overview of the work completed and outcomes delivered by T7.1 in this extension period of PRACE-1IP.

2.1 The Xeon Phi Prototypes

For the Xeon Phi enabling projects in PRACE-1IP extension period, the PRACE prototype system, EURORA at CINECA, was used for most of the projects. A Xeon Phi rack on the MareNostrum system at BSC was also allocated to some of the projects.

2.1.1 *EURORA @ CINECA*

The EURORA (EURopean many cORe Architecture) prototype aims to evaluate a new architecture for next generation Tier-0 systems and has been funded within PRACE-2IP. The original proposal was submitted to PRACE in November 2011 by CINECA, GRNET, NCSA and IPB and was approved in May 2012. The procurement was undertaken by the Italian vendor Eurotech which delivered the cluster to CINECA in January 2013. As a result of its novel, energy-efficient architecture employing a hot water cooling system, EURORA reached first place in the Green Top500 in June 2013 [2]. In the PRACE-1IP extension period, EURORA was used as the main prototype on which the Xeon Phi enabling projects were implemented and supported.

The current configuration of EURORA consists of 64 compute nodes, each with two 8-core Intel Xeon SandyBridge processors (2.10 and 3.10 GHz) and has either 16Gb (thin nodes) or 32Gb memory (Fat nodes). Since each node is equipped with either two Intel Xeon Phi (MIC) cards or two nVIDIA Tesla K20 (Kepler) graphics accelerators, the system in current production provides a total of 1024 SandyBridge cores with 64 Xeon Phi cards and 64 nVIDIA GPUs, and a total of 1.1 Tb of memory. For the interconnects between nodes EURORA uses a custom technology based on an FPGA 3D Torus and an Infiniband network.

The user environment employs a Linux (RedHat CentOS release 6.3) operating system and one login node (consisting of two 6-core Intel Westmere processors). There is 86 Tb of scratch disk space as well as access to the CINECA Data Storage Infrastructure. The software stack includes Intel, GNU, and PGI compilers and MPI implementations such as OpenMPI and IntelMPI, while user jobs are managed by the PBS Pro batch system. For the development of accelerated codes, the CUDA and OpenCL environments are available and Intel's MPSS for the Xeon Phi has been installed. Documentation and introductory tutorials are available on CINECA's HPC pages and user support is provided by the centre's HPC help desk team. Further information about the EURORA system can be found on CINECA's HPC portal [3].

2.1.2 *Xeon Phi @ BSC*

Since July 2013, MareNostrum system at BSC provides a rack with Intel Xeon Phi (MIC) accelerators for general users. In the PRACE-1IP extension period, this Xeon Phi rack at BSC was allocated to some of the T7.1 Xeon Phi enabling projects.

The Xeon Phi rack at BSC has 42 IBM dx360 M4 compute nodes, and each compute node has 2 Intel Xeon CPU E5-2670 @ 2.60GHz processors (8 cores/processor), 64 GB of RAM memory and 2 Intel Xeon Phi 5110P cards, i.e. this rack provides 672 SandyBridge cores with 84 Xeon Phi cards, and a total of 2.6 TB of memory. The Intel Xeon Phi 5110P accelerator is connected to the node's baseboard via PCI Express 2.0 and it is capable of running up to 240 processes.

2.2 Calls for Xeon Phi Enabling Proposals

There were two calls opened for the Xeon Phi enabling project proposals in the PRACE-1IP extension period, including the PRACE Preparatory Access Type C Call for EURORA access (cut-off: 3 June 2013) and the PRACE-1IP Extension WP7 Internal Call. A total of 29 proposals were received and 27 proposals were accepted. The accepted projects were provided with the Type C enabling support by PRACE experts (See Section 2.4 for more details). The following table shows the status of the received proposals for the two calls.

Calls	Open Date	Closing Date	Received Proposals	Accepted Proposals	Projects/ Support Due
PRACE Prep Access Type C Call for EURORA	21 May 2013	3 Jun 2013	21	20	31 Dec 2013
PRACE-1IP Extension WP7 Internal Call	17 Jul 2013	15 Aug 2013	8	7	31 Dec 2013

Table 1 Calls for enabling projects on Xeon Phi prototype in PRACE-1IP extension period

2.3 The Review Process

The review process for the incoming proposals for EURORA in the PRACE-1IP extension period followed the same review process for the Type C proposals in PRACE-2IP and PRACE-3IP. T7.1 provided the technical reviews for all the received proposals. The technical reviews included two aspects:

- Home site review

The purpose of the home site review was to evaluate whether the technical work proposed in the received proposals were feasible and suitable to implement on EURORA from the system aspects. The home site reviews were completed by the home site of EURORA, CINECA, within 2 weeks after the calls were closed.

- Type C review

The purpose of the Type C review was to evaluate whether the technical work proposed in the received proposals was feasible and suitable for Xeon Phi from the application enabling aspects. Each proposal was reviewed by two Type C reviewers independently and the final

Type C review result was based on the comments from both Type C reviewers. All the Type C reviews were completed within 2 weeks after the calls were closed.

The proposals together with the technical review results were then evaluated by the peer review group, before the final decisions were suggested and approved.

2.4 PRACE Expert Assignment and Applications Enabling

Each accepted Xeon Phi enabling project was assigned to one or more PRACE experts. The main responsibilities of the PRACE experts included:

- Preparing a work plan together with the PI / project users
- Taking part in the optimisation work for the accepted enabling project
- Reporting the enabling project status every month (Telcons / emails)
- Participating in the writing of the final report together with the PI / project users, contributing to the T7.1 deliverable and delivering a whitepaper of the technical results where possible

During the PRACE-1IP extension period, most of the T7.1 PRACE experts were deeply involved in the applications enabling work on Xeon Phi. Based on the timeline for the PRACE-1IP extension period, all the application enabling support provided by the PRACE expert will be completed by the end of December 2013.

2.5 Progress Monitoring

The progress tracking was well monitored for all the Xeon Phi enabling projects. Monthly telcons were organised and a mailing list were maintained / used for the status / issue reporting and discussion by the PRACE experts. T7.1 also held Face-to-Face sessions during the PRACE-1IP extension period, including the PRACE-1IP extension WP7 session at the PRACE All Hands Meeting in Varna (September 2013) and the parallel session for T7.1 in PRACE-1IP extension period at WP7 Face-to-Face Meeting in Dublin (December 2013). The overall task progress was reported at the WP7 monthly telcons.

2.6 Xeon Phi Training

To provide training for users wishing to develop codes for the Xeon PHI (MIC) architecture PRACE organized a summer school in Bologna (8 -11 July 2013) with the title “Enabling Applications on Intel MIC based Parallel Architectures”. The four-day event was held at CINECA and covered key topics such as the Intel MIC Architecture, multi- and many-core programming techniques, the Intel programming environment and scientific libraries, SIMD/vectorisation and code optimisation. As well presentations given by Intel engineers, experiences on porting applications to MIC architectures were also presented. In addition to the presentations, the agenda also included practical, hands-on sessions where participants were given the chance to experiment with MIC programming on the EURORA prototype at CINECA. The school was free of charge to students and academics residing in PRACE member states and grants were available for attendees not funded by their institutions. The hands-on training limited the number of participants to 34.

Full information on the course is available from the PRACE events website [4].

2.7 Outcomes

During the PRACE-1IP extension period, a total of 27 Xeon Phi enabling projects were supported by T7.1. This deliverable lists the brief reports from all the projects in Section 0. In addition, most of the enabling projects will deliver a white paper to include further technical details and share more experiences on the Xeon Phi accelerators. The white papers will be reviewed and made available for publication on the PRACE-RI web site [1] in January 2014. The white papers are listed in the Section 5 Annex.

3 WP7 PRACE-1IP Extension Projects Reports

In this section, each individual Xeon Phi enabling project (Type C / internal) provides a brief report on the project's basic information, project objective, the work completed and the result summary.

3.1 Convective Turbulence

Project leader: Joerg Schumacher
PRACE expert: Alexander Schnurpfeil, Stefanie Janetzko, Florian Janetzko (FZJ/JSC)
PRACE facility: EURORA
Other facilities (if any): JUROPA / JUROPA3 at FZJ/JSC
Research field: Engineering and Energy
Application code: RayBen
Project Type: PA Type C
PA number (for Type C project only): 2010PA1734

Project objectives:

This project aims to analyse and enable the hybrid code *RayBen* for the *Intel® Xeon Phi* coprocessor. The code is actively used in the research group of Prof. Schumacher at the University of Ilmenau/Germany. RayBen implements a finite difference scheme for the simulation of turbulent Rayleigh-Benard convection in a closed cylindrical cell. Besides analysis of the code and enabling considerations this project clarifies whether RayBen benefits from the Xeon Phi architecture for future high-resolution studies of large aspect ratio convection systems.

Work done and results:

The main work done in this project will be published online on http://www.prace-ri.eu as a white paper, so that here only a short summary is given of the work done so far.

In order to familiarize with the structure of the code a call graph was created in a first step. The most involved routines were identified and considered as potential candidates for the usage of the offload mechanisms the Intel® Xeon Phi architecture provides.

Further analysis focussed on the OpenMP implementation whereby race conditions were identified and fixed.

The work plan also included extended benchmark runs of RayBen on EURORA as well as on JUROPA and JUROPA3. For this RayBen was integrated in JuBE the Jülich Benchmark Environment. The scaling analysis showed that the performance is predominantly determined by the MPI part. The OpenMP part shows some impact but plays only a minor role. Nevertheless it could be shown that native compilations of the code for the MIC architecture shows comparable performance to the code executing on the CPU. However, the limited memory resources on the coprocessor restrict runs to medium sized grids.

Conclusion:

RayBen proved to be a memory intensive application that highly benefits from the MPI parallelization. This outcome plays an important role when it comes to the Intel® Xeon Phi architecture. Here offloading mechanisms will considerably lower the performance of the code as too much data needs to be transferred too often between the coprocessor and its host. Instead the approach of native binaries, i.e. binaries that run exclusively on the coprocessor becomes an option. Then RayBen shows satisfying performance. However, this behaviour is less caused by the thread but mainly by the MPI parallelization. Therefore, it can be

concluded, that Intel® Xeon Phi is not the first choice for this kind of code, in particular with regard to the limited memory resources.

3.2 Evaluation and porting of WARIS, a framework for atmospheric simulation, to the MIC heterogeneous architecture

Project leader: Arnau Folch, BSC
PRACE expert: Raúl de la Cruz, BSC
PRACE facility: Xeon Phi rack on MareNostrum at BSC
Other facilities (if any):
Research field: Earth Sciences and Environment
Application code: WARIS
Project Type: PA Type C
PA number (for Type C project only): 2010PA1740

Project objectives:

WARIS is an in-house multi-purpose framework focused on solving scientific problems using mainly Finite Difference Methods (FDM) as numerical scheme. Nevertheless, the numerical methods supported in the framework are not only tied to explicit time integration schemes, but also to semi-implicit and implicit schemes in order to guarantee stability for linear and non-linear terms. Its framework was designed from scratch to solve in a parallel and efficient way Earth Science and Computational Fluid Dynamic problems among a wide variety of architectures. Structured meshes (regular or non-regular) are employed to represent the problem domains, which are better suited to be optimized in accelerator-based architectures.

To succeed in such challenge, WARIS framework was initially designed to be modular in order to ease development cycles, portability, reusability and future extensions of the framework.

The porting of WARIS to an emerging heterogeneous architecture, such as Intel Xeon Phi (MIC), would make a step forward in our development. This new many-core architecture may enable us to run very large environmental simulations (volcanic ash dispersion or fluid dynamics cases) in a rapid and efficient way. Therefore, our main intention is to run environmental numerical simulations of actual ash dispersion cases in a massively parallel architecture such as the BSC's System (based on Intel MIC). Thus, enabling the prediction of volcanic ash dispersion in the foreseeable future at European level and at really large cases such as world-wide simulations. We strongly believe that the MIC architecture can fit quite well for these purposes.

Work done and results:

The optimization work completed: The latest trunk version of WARIS repository has been successfully compiled natively for MIC processors (*k1om* cores). This initial porting includes some minor changes in the configuration and compilation files for this specific architecture. Furthermore, many external libraries have been ported and compiled in order to be used together with WARIS on MIC processors. Those libraries are: HDF5, NetCDF, Xdmf and Xml2 enabling Parallel I/O through the MPI-Intel implementation during their compilation. A fully operational and serial version of WARIS for MIC platforms was finally obtained. This current version was not fully SIMDized neither highly tuned for the target platform, and therefore its performance was far from the desirable objective.

The optimization work on going: An initial parallel version of a pure MPI implementation of WARIS has been built. The first MPI results in a MIC card (244 MPI processes) showed a scalability ranging from 60 to 90x depending on the input case (related with the domain

problem size). Additionally, some interesting results were also achieved on improving the Parallel I/O operations. In order to reduce the I/O bottleneck in parallel executions when small data chunks are simultaneously written by MPI processes, an active buffering strategy with two-phase collective I/O calls has been implemented.

Conclusion:

Currently, the ongoing work is focused on creating an hybrid version of WARIS framework using MPI (for extra-domains) & OpenMP (for intra-domains). The target is to use several MIC cards running only one MPI process on each of them, which will command up to 244 OpenMP threads. Future work will lead to tuning the blocking techniques already included in the WARIS framework for MIC architecture, and adding Intel pragma hints for auto-vectorization and pre-fetching on some kernel's hotspots.

3.3 High End Computational Modelling of wave Energy Converters

Project leader: Professor Frederic Dias, University College Dublin
PRACE expert: Mr. Christian Lalanne, Dr. Michael Lysaght, ICHEC
PRACE facility: EURORA
Other facilities (if any): ICHEC Xeon Phi development platform
Research field: physics, engineering, renewable energy
Application code: UCD-SPH
Project Type: PA Type C
PA number (for Type C project only): 2010PA1713

Project objectives:

The purpose of this project is to first introduce OpenMP parallelization to the UCD-SPH code, to port the UCD-SPH code to the Intel MIC architecture and then to subsequently optimize the code on that architecture.

The UCD-SPH code utilises the Smoothed Particle Hydrodynamics (SPH) method for modelling wave interaction with an Oscillating Wave Surge Converter (OWSC) device. The SPH scheme used in the UCD-SPH code is based on the SPH-ALE formulation. The standard SPH method is a purely Lagrangian technique and the "particles" are moving nodes that are advected with the local velocity and carry field variables such as pressure and density. However, the SPH-ALE formulation is based on the solution of a moving Riemann problem in the Arbitrary Lagrangian-Eulerian context and hence the so-called particles are moving control volumes and not particles. As the fields are only defined at a set of discrete points, smoothing (interpolation) kernels are used to define a continuous field and to ensure differentiability.

The code was developed by Dr. Ashkan Rafiee and has been validated extensively in numerous applications. The version that is going to be ported to the Intel MIC in this project is based upon the three-dimensional SPH approach and has recently being parallelised using the MPI model.

Work done to date and results:

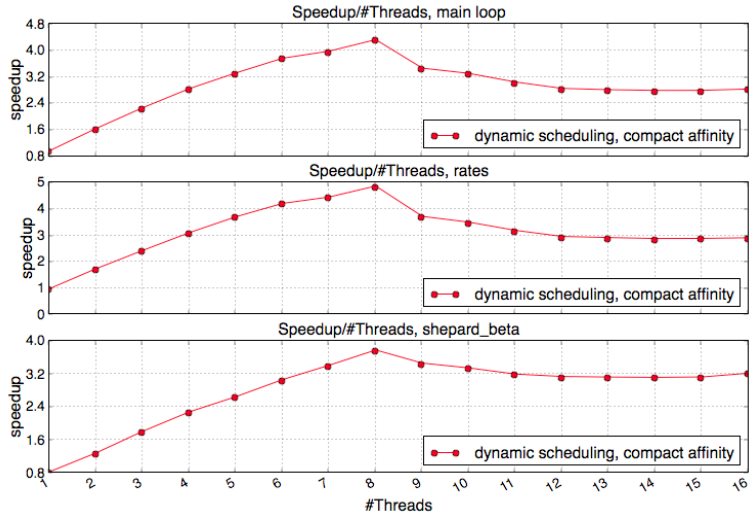
Our work to date has been carried in several phases, which are described in Table 2.

Phase 1	Study of the algorithm and its current implementation.
Phase 2	Measurement of key parts of the current implementation, to find the parts of the code suitable for OpenMP parallelization.
Phase 3	Implementation of unit tests to ensure the correctness of the code after refactoring.
Phase 4	Implementation of a particle parallelization approach, changing the algorithm for updating physical quantities to another one better fit for parallelization.
Phase 5	Measurements of the performance of the parallelization on different systems including EURORA.
Phase 6	Migration of the code to Intel Xeon Phi architecture using Native Mode.
Phase 7	Measurements of the performance of the Native version of the code on EURORA.
Phase 8	Migration of the code to Intel Xeon Phi architecture using Offload Mode.
Phase 9	Measurement of the performance of the Offload version of the code on EURORA.

Table 2 Work done in phases.

A performance analysis of the original implementation of the UCD-SPH code reveiled, that the `shepard beta()` and `rates()` subroutines were the most computationally expensive and we therefore parallelised these subroutines using OpenMP. While we have ported the code to the Xeon Phi in both *native* and *offload* modes, in this summary of our work we only report results for the code running on the Xeon Phi in *native* mode. Performance results for the UCD-SPH code running in offload mode will be reported in full in the PRACE whitepaper associated with this work.

Figure 1 shows the demonstrated speedup of each of these subroutines across OpenMP threads relative to the original serial implementation of the code (speedup = 1). In Figure 1, we also show the speedup of the 'main loop' of the code relative to the 'main loop' timed in the serial implementation of the code. The top plot shows results for a single time step (*main loop*). The middle plot shows results for the `rates()` subroutine and the bottom plot shows results for the `shepard beta()` subroutine.



Component	Speedup
`Rates()`	4.8x
`shepard_beta()`	3.8x
main_loop	4.3x

Table 3 Speedup results of the particle parallelization approach executed in the host (Intel (R) Xeon(R) CPU E5-2658@2.10GHz).

Figure 1 Speedup of the particle parallelization approach on the Host IIntel (R) Xeon (R) CPU E5-2658 @2.10GHz). The figure shows the speedup relative to the original unoptimised version of the code.

We can clearly see in Figure 1 that a reasonable speedup was achieved in comparison with the original version of the code (details in Table 3).

As is to be expected, we observed that with an increase in interacting neighbours (workload), the OpenMP parallelization scales better over a larger thread count. (The number of interacting particles is controlled by the ***hfac*** parameter, which affects the smoothing length of the algorithm) as can be seen in Figure 2. Figure 2 demonstrates that improved scalability is achieved by increasing **hfac**, which translates to more particles participating in the interactions (increased work load).

Figure 3 shows the speedup achieved on a single Xeon Phi in native mode relative to the speedup achieved on the host with the maximum value of hfac that the current version of the program can handle (absolute speedup against the original version of the code).

It can been seen in the lower subplot of Figure 3 that, currently, the performance achieved by the code running on the Xeon Phi exceeds the performance of the code running on the host for the *shepard_beta* subroutine only.

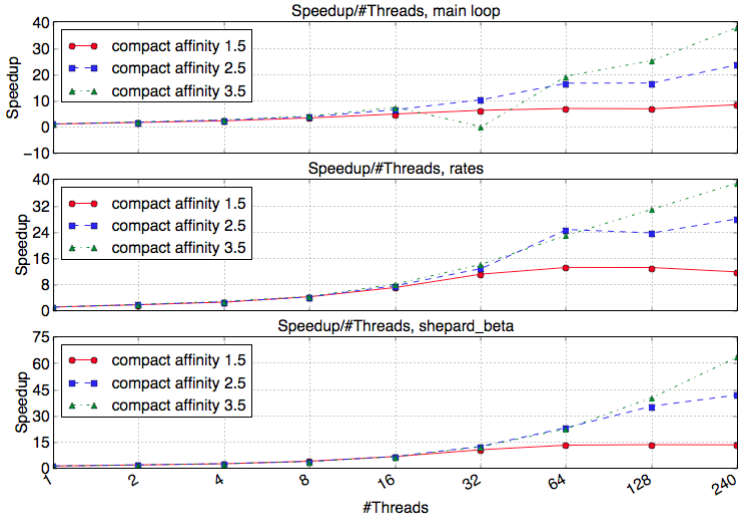


Figure 2 Relative speedup with different values of hfac **(1.5, 2.5 and 3.5), running natively in the Intel Xeon Phi Co-processor (Intel Xeon Phi 5120D,8 GByte RAM). A larger value of** hfac **results in a lager number of neighbouring particles, which results in a larger workload per thread.**

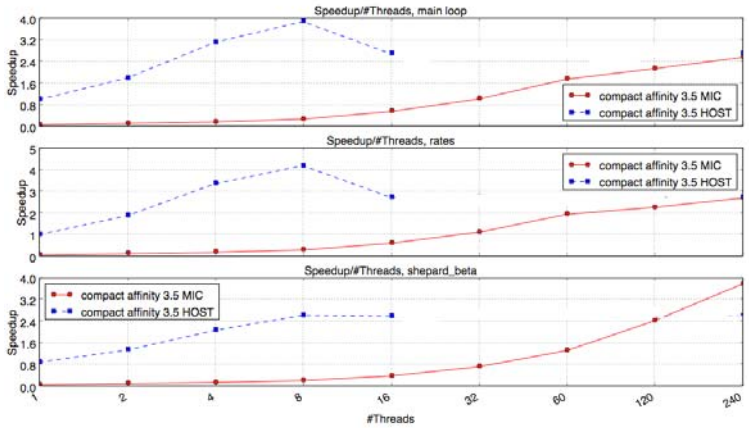


Figure 3 Comparison between speedups achieved executing the code on the host and natively on the Xeon Phi. Both speedups curves are calculated relative to the original version of the code (for hfac=3.5).

Conclusion:

We have enabled the UCD-SPH code on the Intel MIC architecture for the first time. We have done so by introducing new OpenMP parallelisation to the code, where, as a result of carrying out an in-depth performance analysis, we have targeted the most compute intensive sections

of the code. While we have enabled the UCD-SPH code on the Xeon Phi in both native and offload mode, in this summary of our work we have only shown results in native mode. The performance results for the code running in offload mode will be described in full in the forthcoming PRACE whitepaper associated with this work.

Upon introducing OpenMP parallelisation, the results we show here demonstrate that, while reasonable scalability of the code across OpenMP threads has been achieved on the host CPU side, achieving scalability across all available threads on the Xeon Phi has proven challenging. We have found that the main bottleneck of the parallelization is thread synchronization due to the unbalanced workload among threads. However, results also indicate that with further refactoring improved scaling across threads can be achieved. A new implementation to explicitly balance workload and to subsequently change the OpenMP scheduling type is currently in progress. It should be emphasised that the small size of the problem focused on here makes the full exploitation of the Xeon Phi more challenging. It can be seen from the results shown above that increasing the workload significantly improves the scalability of the code over threads.

Initial auto-vectorisation reports from the intel compiler (v14.0.0) suggest that vectorisation of loops may prove challenging. However, with further investigation we may focus effort on refactoring loops so that the compiler can take advantage of the wide SIMD units on the Xeon Phi. On top of this, we would also like to investigate the benefits of running the code in MPI *symmetric* mode, whereby multiple MPI processes could be run on the Xeon Phi, thereby significantly reducing the overhead of synchronization associated with scaling across all ~240 threads of the co-processor.

3.4 Evaluating Performance and Scalability of HPC Workloads on the Xeon Phi

Project leader: Vicenç Beltran, BSC
PRACE expert: Judit Gimenez, BSC
PRACE facility: BSC's System
Other facilities (if any): Cholesky, ALYA, iPIC3D
Research field: Mathematics and Computer Science
Application code:
Project Type: PA Type C
PA number (for Type C project only): 2010PA1730

Project objectives:

The goal of this study is to bring new insights about the performance/productivity tradeoff offered by the Xeon Phi and to investigate the role that this family of accelerators may have on future Exascale systems. Both numeric kernels and real applications were targeted.

Work done and results:

Kernel analysis: The Xeon Phi processor requires large amounts of parallelism to make the most out of all the cores available. A well-known technique to provide this extra level of parallelism is task nesting, which dramatically increases the available number of tasks, but requires efficient management of fine-grained tasks to get the best performance. This technique has been applied to the Cholesky decomposition to study its performance and scalability.

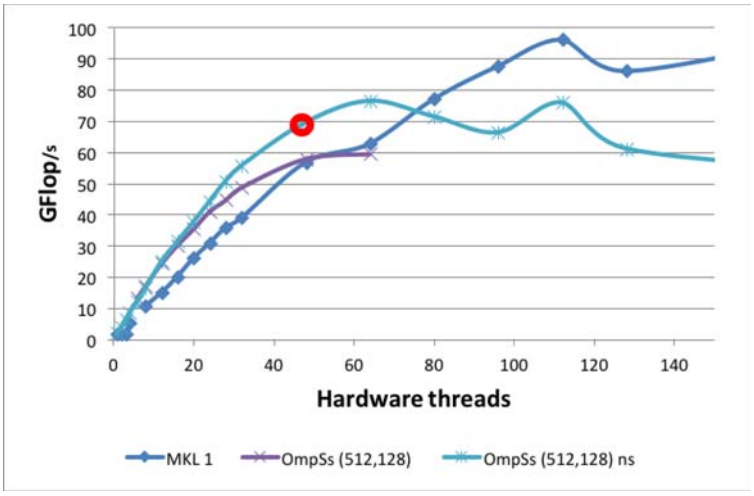


Figure 4 Cholesky Scalability on the Xeon Phi

Figure 4, compares the scalability of three different configurations of the Cholesky decomposition. The X axis shows the number of hardware threads used and the Y axis the GFlop/s obtained. The problem size is set to 8192x8192 elements. The first configuration evaluated (dark blue) is the sequential version of Cholesky compiled and linked with the parallel version of the Intel MKL library, the second one (purple) is the OmpSs version of Cholesky linked with the sequential version of the MKL library, finally the last configuration (light blue) is the OmpSs version augmented with task nesting to generate more task level parallelism. As we can see, the OmpSs version with nesting is the best performing up to 64 hardware threads. With more than 64 hardware threads the MKL obtains the best performance. Finally, the OmpSs version without nesting only scales reasonably up to 32 cores, starting at that point the task parallelism available is insufficient to make the most of more hardware threads.

Application analysis: The goal was to evaluate the performance and scalability of real applications runninng on the Xeon Phi accelerator with the minimun porting effort (no vectorization or target specific optimizations). The scalability analysed the strong scaling scenario. The applications' efficiency were studied using the BSC model described in [5] that measure different performance factors like load balance or transfer efficiency as a number between 0 (really bad) and 1 (perfect). On each plot, the X axis represents the different runs, while the series are different factors of the model.

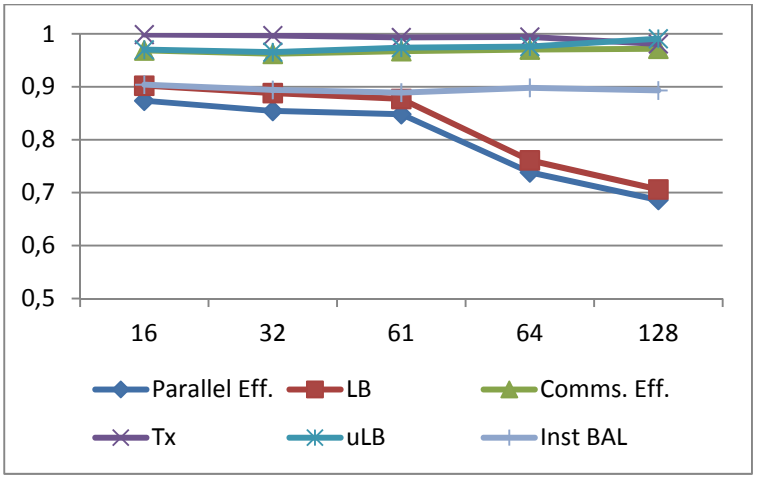


Figure 5 ALYA efficiency analysis.

The aim of **ALYA** study was to analyse the application scalability on the Xeon Phi. The execution time regularly decreases up to 61 tasks, and then it has an inflexion, when more tasks than physically available cores are used. The plot in Figure 5 indicates that the parallelization efficiency gets reduced from 0.9 (from 16 to 61 tasks) to 0.75 (64 tasks) and 0.7 (128 tasks), directly caused by the load balance (LB) degradation. The application traces indicate that when sharing the physical CPUs the performance (IPC[1]) is reduced. The process mapping causes IPC imbalance that is reflected as load balance degradation.

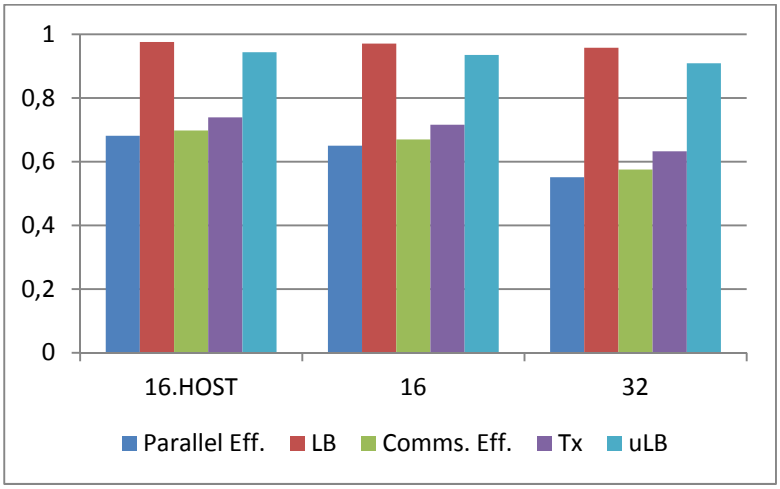


Figure 6 iPIC3D efficiency analysis.

The target of the **iPIC3D** analysis was to compare the Intel Xeon with the Intel Xeon Phi. Due to some unresolved errors running large input cases, the execution on the Xeon Phi cannot run with more than 32 MPI tasks, so only the 3 configuration reported in Figure 6 were analysed. In terms of time, we observed that the regular Xeon execution (16.HOST) was 7 times faster than the largest execution on the Xeon Phi (32). In terms of the parallel efficiency, using the same number of tasks, the efficiency is nearly the same on both architectures (around 0.6) with a similar distribution among the performance measurement factors (a number between 0 (really bad) and 1 (perfect).). Increasing the number of tasks in the Xeon Phi reduces the parallel efficiency (in this case, to 0.5), basically due to a degradation of the communications efficiency, and more precisely, in this application, this is cause by the transfer efficiency (Tx)

Conclusion:

The OmpSs version of Cholesky evaluated on the Xeon Phi shows a good potential to make the most of the large number of cores available on this accelerator. Some phases of the algorithm are scaling almost perfectly, but some other phases, specially the final one, do not scale well with the number of processors. In terms of the MPI applications analysed, we have observed that even the Xeon Phi architecture offers a good solution for dense parallelism, using these kinds of applications directly to this accelerator does not take full profit of this architecture. It is interesting to note that when we use more MPI tasks than physical CPUs, the tasks mapping produces an imbalance in the IPC affecting the total application load balance, as we observed in ALYA. Comparing the Xeon Phi with a regular Xeon, the parallelization efficiency and its decomposition were similar when using the same number of tasks.

3.5 Scaling AVBP on multi MIC

Project leader: Dr. Isabelle D AST, CINES
PRACE expert: Eric Boyer - CINES
PRACE facility: EURORA
Other facilities (if any):
Research field: Engineering and Energy
Application code: CFD code AVBP
Project Type: PA Type C
PA number (for Type C project only): 2010PA1741

Project objectives:

Porting, optimizing and scaling AVBP on multi MIC

Work done and results:

The CERFACS code AVBP is a 16-year-old code written in Fortran and C with m4 macros and relying on simple MPI for parallelisation. The code has always demonstrated excellent scalability on standard clusters but required some refreshing to tackle future architectures. The main refactoring focused on removing bottlenecks for the code porting and execution on MIC architectures.

Optimisation regarding thread parallelism: AVBP is a highly efficient parallel program based on MPI. However the increased number of tasks required to run on MIC increases the program size due to the domain decomposition techniques used for CFD that rely on overlapping elements (in our case only frontier nodes are duplicated but they can still account for up to 20% of the problem to solve).

Since the AVBP code has already demonstrated a high strong scaling qualities up to 131072 MPI tasks on Blue/Gene/Q with an 85% efficiency using 4 tasks per physical core (64 tasks per node) we chose to implement thread parallelism using a coarse grain approach. Indeed, the coarse grain approach uses similar techniques to those used for MPI parallelism where data is evenly distributed to all tasks and each worker performs as much as possible the same work avoiding to the outmost synchronisation procedures. In AVBP implementing this translate to distributing the data between the threads before the computing loop. Each thread executes the whole computing loop with ad-hoc synchronisations when using MPI synchronisations or global operations (see Figure 7).

Progress in this topic has been slow since it requires the rewriting of 80% of the code to account for the threads but it is expected to yield excellent results. A preliminary version is available with an estimated 60% of the compute loop threaded (the estimation was performed using the Intel tool Vtune) and results are shown in the next sections.

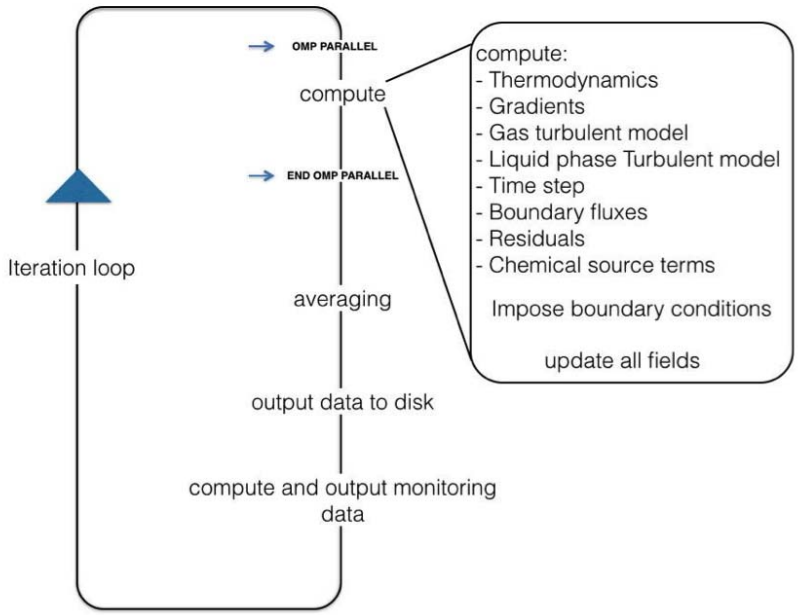


Figure 7 AVBP: each thread executes the whole computing loop with ad-hoc synchronisations when using MPI synchronisations or global operations.

Optimisation regarding vectorisation: The AVBP code was specifically written for scalar processor neglecting all the vectorisation techniques required for NEC and other vector processors. Due to this omission it has a very bad record regarding vectorisation. Performance tests were done using compiler automatic vectorisation on Xeon processors (AVX). In this case performance actually drops by 10% compared to the non-vectorised execution of the code with -O3 optimization. Using user added SIMD pragmas to force vectorisation in desired parts of the code yielded only 2% improvement, below statically measurable gains when performing a simulation multiple times.

The main reason behind this bad vectorisation comes from the data structures of the code. Most arrays are organised with non 8 bytes multiples sizes and with small heading indexes thus rendering vectorisation very difficult. Data padding has been tested on specific kernels and allowed for localised performance boosts but required up to 30% 'dummy' data increases per tasks making this approach unpractical.

Manual transposition of the array indexes has demonstrated an excellent performance return however. By simplify changing the indexing a performance boost of 200% was observed on Sandy Bridge and 400% on Xeon Phi when dealing with specific computational kernels of the code. This performance boosts motivate CERFACS to implement this on a large scale on the whole code. Currently, an automatic code to automatically transpose the data is been explored by CERFACS.

Performance and scalability: Figure 8 compares the execution of AVBP on a single Xeon Phi Card to a Xeon processor. The overall time per iteration measured for the threaded section confirms that roughly 60% of the computation work has been threaded. The increase of additional threads speeds up the work but linear scaling is not obtained. Keeping in mind that the AVBP was not threaded when work began and this work is on-going we expect significant performance boosts when the threading and vectorisation strategies are fully completed.

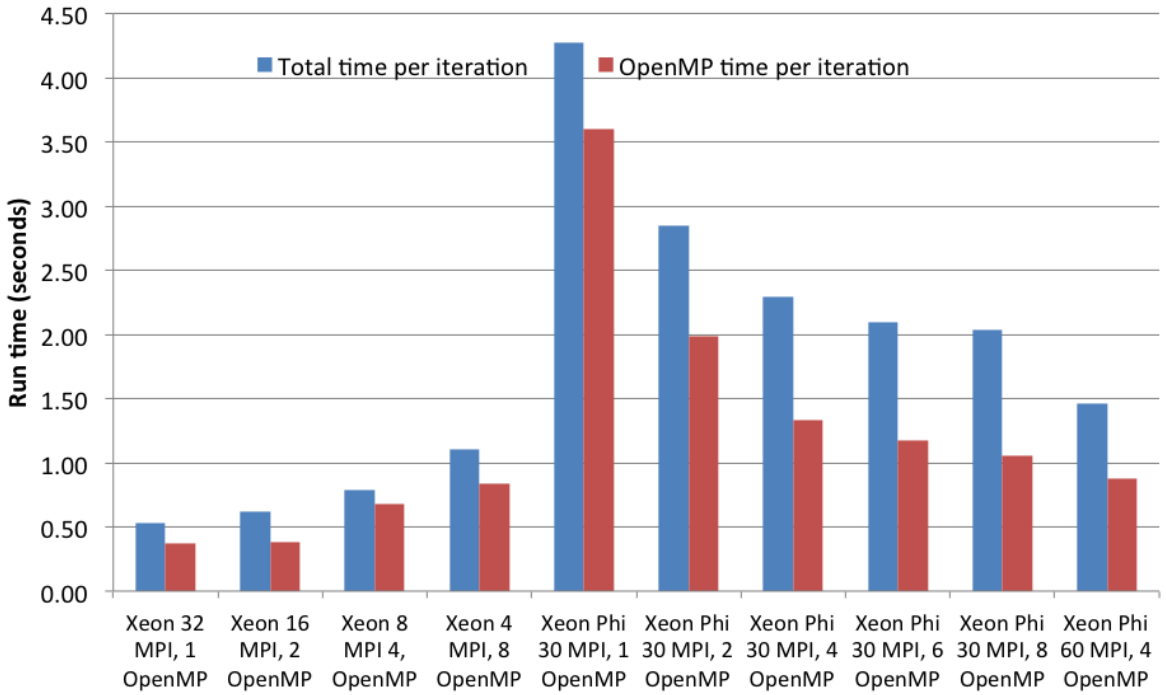


Figure 8 Xeon vs. Xeon Phi scalability and OpenMP performance, using a 2 Xeon processors in a single node, and a single Xeon Phi.

Figure 8 also shows a clear improvement of performance on Xeon Phi when increasing the number OpenMP tasks. However, the speedup plateau is rapidly going from 2 threads to 8 (the number of MPI tasks is adapted to remain below the 240 total tasks threshold).

Paraver profiling has demonstrated this is due to data dependency synchronisations when performing the gather/scatter operations. The gather/scatter operations are used to compute cell data such as gradients using mesh node input data and the results are node output data. The data is gathered from the nodes to the cells, then the gradient operations are performed and when they are done, the data is scattered back to the nodes. Since we choose to implement a coarse grain threading based on the nodes, the gather operation and the scatter operation require BARRIERS since the full data array must be treated before continuing the work. Also since the gradient data cannot be computed without the full gathered vector, each thread is required to wait for the work of the others before continuing. When the number of threads exceeds two each thread is executed sequentially one after the other following the data distribution order. This explains the very bad scalability of this part of the code and requires a full refactoring by modifying the data structure of these operations.

Additionally the remaining kernels will be extended to support threaded execution to achieve as much as possible 100% threading of the compute loop. Currently we expect only the I/O section and the MPI calls to remain thread protected with master sections and executed by a single unique thread.

Conclusion:

The work performed on vectorisation and openmp parallelisation show the code needs a full refactoring on data structure to be efficient on MIC processors. We intent to going on this thread parallelisation and we would like to parallelise the whole code from the current 60%. Adaptation of the code to take advantage of vectorisation will also be taken into account. With this refactoring finished, multi-MIC optimisation will be the next target.

3.6 MagHydroBurn

Project leader: Evghenii Gaburov, SURFSara
PRACE expert: Evghenii Gaburov, SURFSara
PRACE facility: EURORA
Other facilities (if any):
Research field: Astrophysics
Application code: MagHydroBurn
Project Type: PA Type C
PA number (for Type C project only): 2010PA1744

Project objectives:

To port and optimize the application for efficient execution on XeonPhi

Work done and results:

The enabling work involved porting and optimizing an astrophysical (magneto-) hydrodynamics Fortran code to XeonPhi, and by extension Intel CPUs. For hardware we used a single socket SandyBridge E5-2650 CPU (8 core/16 HT threads), to which we refer as SNB, and a single XeonPhi 5100 board, which will be referred to as KNC. We used the Intel Compiler suite 14.0.1 for compilation and the Intel VTune amplifier for performance analysis. The following compiler flags has been used with both SNB and KNC: "-O3 -parallel -openmp -g"; in addition, we used "-xavx" for SNB and "-mmic" for KNC. Thus, we only investigated the performance of the code in the NATIVE execution mode. This allows to exclusively focus on the application performance and to exclude any additional performance impact noise due to OFFLOAD mode. Only after we have achieved acceptable performance in the NATIVE mode, we could possibly consider expansion of the code for OFFLOAD execution.

The technical details of this project will be presented in an accompanying white paper, but it is suffice to say that initial profiling of application revealed that major bottlenecks were due to myriads of memory copies disguised as __intel_ssse3_rep_memcpy function calls. Also, the code demonstrates substantial load-imbalance manifested via __kmp_waitsleep calls. The four other major hotspots were due to two compute heavy functions that compiler failed to auto-vectorize and two hotspots were limited by memory bandwidth and load-imbalance. Needless to say that, by extrapolating to 60 KNC cores running 4 threads each, these inefficiencies and lack of code scalability will be devastating to the performance. Indeed, the application runtime, as is, in the NATIVE mode on KNC is 6x slower! Of course, the problem size is small to get any descent performance improvement from KNC, but it is highly unlikely that by increasing problem size the application efficiency will be increased by an order of magnitudes. Furthermore, the original code has a catastrophic bug, which prevents to handle the problem sizes bigger than 192x480.

In the end, we identified the causes of inefficiencies, and were able to fix them one by one. Unfortunately, the whole application (circa 20k lines) was written in this inefficient manner, and it is beyond the scope and time allocated for this project to fix this all. In other words, it is virtually impossible to undo the damage done to the code over many year of development in a just few months of hard work. However, in collaboration with the user we were able to undo the damage to the parts that are currently important to the user.

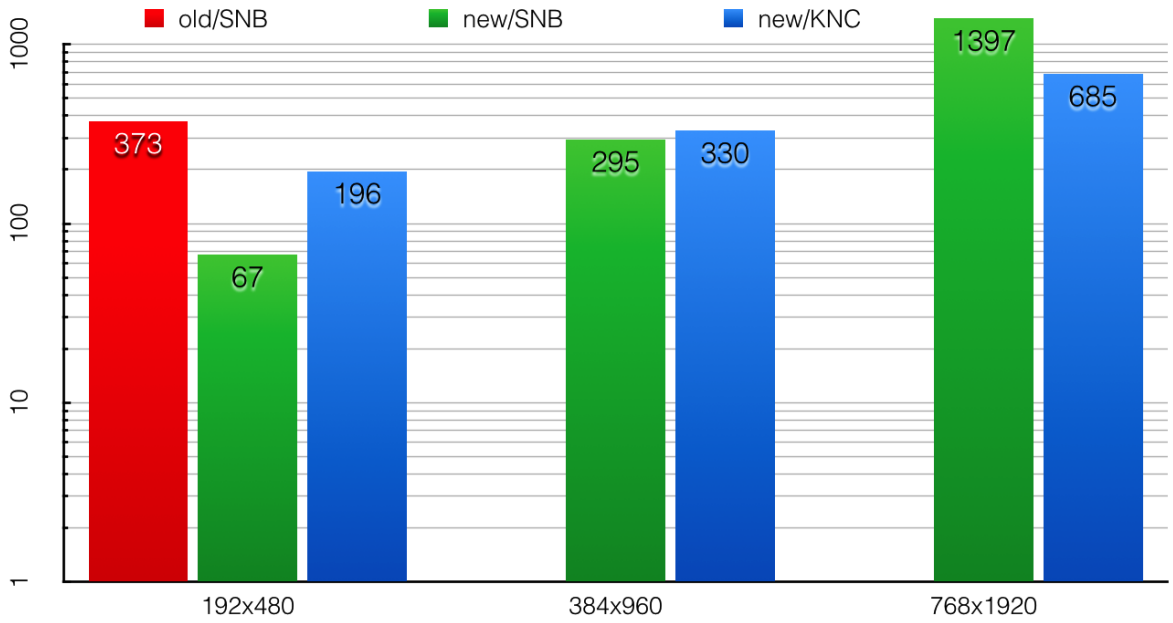


Figure 9 Runtime in seconds of the original (old) and the tuned (new) code on a single socket SandyBridge E5-2650 CPU (SNB) and XeonPhi 5100 (KNC) coprocessor as a function of the problem size.

We show the results of this project in Figure 9. The red bar on the left shows the runtime of the original code in seconds on SNB with the largest problem size that the code was able to handle. As we mentioned earlier, due to a bug, the code throws "segfault" on larger problem sizes. The green and blue bars show runtime of the tuned code on SNB and KNC respectively. We do not include the wall-clock time for the old code executing on KNC because it is off-the the chart — 38 minutes. Overall, the performance improvement for the code can summarized as follows: at least ~5.5x for SNB and ~12x for KNC. More importantly is that for large enough problem size (760x1920) the KNC is 2x faster than SNB. Further work is required to improve code execution efficiency on XeonPhi. While we identified what needs to be done, the actual work is beyond the scope of this project.

Conclusion:

The overall success of this project can be judged from the results in Figure 9. In our opinion, this project demonstrates that it a complex "home-brewed" code can achieve performance on XeonPhi above what a SandyBridge CPU can deliver. However, this is a non-trivial process and requires substantial development and code refactoring efforts. In our experience with porting codes to CUDA-enabled GPU, the amount of effort required to get such a complex application to run efficiently on XeonPhi are equivalent, if sometimes not larger, than efforts required to get application running efficiently on GPU. The biggest issue today is that current Intel Compiler tools are not mature enough to deliver this performance out-of-the-box, and therefore out-of-the-box solutions are required to achieve good performance.

3.7 SPECFEM3D code porting, optimization and scalability analysis on INTEL Xeon Phi cluster

Project leader: Julien Derouillat, CEA
PRACE expert: Vittorio Ruggiero, CINECA
PRACE facility: EURORA
Other facilities (if any):
Research field: Earth Sciences and Environment
Application code: SPECFEM3D_GLOBE
Project Type: PA Type C

PA number (for Type C project only): 2010PA1749

Project objectives:

The goal of this project is the porting of a seismic modelling and imaging tool called SPECFEM3D_GLOBE for the characterization of earthquakes and mapping of Earth's interior on all scales to an INTEL MIC cluster. An additional objective is optimizing the code and then measuring the performance and scalability on a cluster by using only the MIC.

Work done and results:

To extract the best performances from MIC both efficient shared memory parallelism and good vectorization are needed. The main limitation of SPECFEM3D_GLOBE is the fact that uses pure MPI model for its parallel implementation. The first step is the execution profile to find the main computational kernels in the code. The result is that the aggregated contributions of the five subroutines:

- compute_forces_crust_mantle_Dev.F90
- multiply_arrays_source.f90
- compute_forces_inner_core_Dev.F90
- update_displacement_Newmark.f90
- compute_forces_outer_core_Dev.F90

are about 95 % of the elapsed time for the execution of one temporal step of the simulation.

The five subroutines are been changed introducing OpenMP directives. So the hybrid OpenMP-MPI version of the code was written to benefit from efficient shared-memory parallelism.

Another step was to analyze the code vectorization with the aid of the vec-report compiler option. To aid the vectorization many loops are changed and to switch between vectorized and no vectorized version a suitable pre-processor flag (FORCE_VECTORIZATION) was implemented. To test the code a case regional_MiddleEast was created.

For this test case the result is that the mean elapsed time per time step of the simulation for the vectorized code is about 25% faster than the non-vectorized code.

To test the scalability of the Hybrid OpenMP-MPI we ran the code in native mode using 4 MPI processes and 8 MPI processes respectively and changing the number of threads. The mean elapsed time per time step (do istage = 1, NSTAGE_TIME_SCHEME in the subroutine **iterate_time.F90**) and for the OpenMP Parallel region in the main subroutine (94% of time step) are measured.

In the Table 4 and Table 5 are displayed the results (in seconds) for the two tests, using 4 MPI processes and 8 MPI processes respectively.

Number of threads	**Time step**	**compute_forces_crust_mantle_Dev num_elements=41575**	**compute_forces_crust_mantle_Dev num_elements=2825**
1	7.19	6.3	0.42
40	0.3	0.16	$1.15*10^{-2}$
80	0.24	$9.06*10^{-2}$	$6.6*10^{-3}$
120	0.22	$6.88*10^{-2}$	$5.15*10^{-3}$
160	0.22	$6.85*10^{-2}$	$5.10*10^{-3}$
200	0.22	$5.58*10^{-2}$	$4.26*10^{-3}$
240	0.22	$5.23*10^{-2}$	$4.10*10^{-3}$

Table 4 Results using 4 MPI processes

Number of threads	Time step	compute_forces_crust_mantle_Dev num_elements=20285	compute_forces_crust_mantle_Dev num_elements=2920
1	3.57	3	0.43
40	0.17	$7.83*10^{-2}$	$1.19*10^{-2}$
80	0.14	$4.26*10^{-2}$	$4.80*10^{-3}$
120	0.13	$3.21*10^{-2}$	$3.60*10^{-3}$
160	0.13	$2.97*10^{-2}$	$3.18*10^{-3}$
200	0.13	$2.75*10^{-2}$	$3.11*10^{-3}$
240	0.13	$2.58*10^{-2}$	$2.90*10^{-3}$

Table 5 Results using 8 MPI processes

The subroutine compute_forces_crust_mantle_Dev has a good scaling up to 240 threads, but when the number of threads increases the time step shows a bad scalability because of the serial parts in the code. The performances are not improved using the three basic affinity setting.

Conclusion:

We presented the effort porting the SPECFEM3D_GLOBE code to run on Intel Xeon Phi cluster. The performances for the hybrid MPI-OpenMP version executed in native mode are encouraging. The next steps involve the parallelization of other parts of the code where possible and the analysis of the performance of the computational kernel and of the other routines with OpenMP parallel regions used with the offload model.

3.8 Heisenberg spin glass on Intel Phi (HIP)

Project leader: Massimo Bernaschi, National Research Council of Italy
PRACE expert: Francesco Salvadore, CINECA
PRACE facility: EURORA
Other facilities (if any):
Research field: Fundamental Physics
Application code: Heisenberg Spin Glass
Project Type: PA Type C
PA number (for Type C project only): 2010PA1727

Project objectives:

In statistical mechanics, the term "spin system" is used to indicate a broad class of models adopted for the description of a number of physical phenomena.

The baseline application code for the project is a highly tuned version of a code designed for the simulation of the Heisenberg Spin Glass system (HSG). The code is developed in C and is fully portable. The most time consuming loops have been written in such a way that the vectorization may be carried out directly by a suitable compiler; in such a way, neither compiler directives nor intrinsic vectorization primitives are required. The code has been ported to multiple platforms including cluster of GPUs. The code resorts to MPI for inter-node/GPU parallelization whereas on each computing node we use either OpenMP and vectorization or CUDA, depending on the type of node (CPU or GPU). Extensive numerical experiments on CPUs and on clusters of GPUs showed interesting results and were presented in conferences and published in major journals like Journal of Parallel and Distributed Computing or Computer Physics Communications.

With this project, we aimed at accessing the recently available Intel MIC technology and at comparing the performance of MIC with those of latest Nvidia GPUs (based on the Kepler architecture). Although the code implements algorithms for a specific spin system (HSG), the memory access pattern and the kind of operations are quite common in other application so the results can be of interest also for people working in related fields (e.g., solution of PDE by means of relaxation methods).

The initial objective consists in tuning the single-MIC code for using at its best the technology. After that, the second objective aims at developing the multi-node version to see if it is possible to achieve the overlapping of communication and computation as we successfully did running on clusters of GPUs.

Work done and results:

During the first stage of the work, we focused on porting the code to Intel MIC using the so-called *native* mode. The optimization effort was rather high because of the architecture novelty and, in particular, due to the difficulty to predict the effect of the different strategies on the actual performances. In the end, it turned out that four main optimizations played a crucial role:

(a) enabling vectorization using a clean coding style (already available in the baseline code version);
(b) using OpenMP *collapse* to increase the number of iterations to be parallelized;
(c) padding arrays to avoid dramatic L2 TLB cache thrashing;
(d) avoiding prefetching for write-only accesses.

By turning on all the optimizations, the performance achieved using a single MIC is very close to that obtained with a K20s Kepler GPU.

The second stage was devoted to extend the native porting to multi-MIC configurations. Different asynchronous MPI patterns were implemented and tested.

The third stage was devoted to port the code using the so called *offload* mode. The serial and MPI versions have been prepared. As for the MPI version, an asynchronous pattern exploiting the *offload async* clause was developed.

During the fourth stage, we compared the performances of HSG code using the three different kind of computing units of EURORA cluster, i.e. Sandy Bridge E5-2687W CPUs, Kepler K20s GPUs, and Intel Xeon Phi 5110P.

Our findings can be summarized as follows:

- The performance of the single system may change significantly depending on the size of the problem under study.
- Vectorization is absolutely required on both traditional Intel CPU and Intel Phi coprocessor. This entails that, on a single system, two levels of concurrency must be maintained (vectorization and threads parallelism). In CUDA there are both blocks of threads and grids of blocks but there is a unique Single Program Multiple Threads programming model.
- The Intel Phi is sensitive to a number of potential performance limiting factors. The most apparent we found is the need to pad arrays to avoid dramatic performance drops due to L2 TLB trashing effects.
- A careful tuning of the C source code (i.e., without resorting to vector intrinsic primitives) running on a single Intel Phi allows to achieve performances very close to that of a latest generation (Kepler) GPU. Although the source codes for a traditional Intel CPU and an Intel Phi may be very similar, the effect of (apparently) minor changes may be significant so that the expected main advantage of Intel Phi (code

portability) may be limited. On the other hand, GPU requires a porting to the CUDA architecture whose cost depends on the complexity of the application and the availability of libraries for common operations.

- The main differences in performances between GPU and Intel Phi were obtained in multi-system configurations. Here the stream mechanism offered by CUDA allows to hide the communication overhead, provided that the computation to be executed concurrently with the communication is long enough. As a result, the efficiency is always outstanding. By running on an Intel Phi in offload mode it is possible to emulate, somehow, the CUDA stream mechanism that supports independent execution flows. In offload mode the efficiency of a multi Intel Phi configuration is higher, but it remains low compared to a multi GPU configuration with the same number of computing units for a problem of fixed size. For problems where the size for computing unit is constant (so that the total size of the problem increases with the number of computing units), the efficiency of Intel Phi is close to ideal but the performance of the single system (due to the offload execution mode), at least for our reference problem, is significantly lower with a degradation of about 40%.

Conclusion:

In a nutshell we can state that, for the problem we studied:

- Single system: 1 GPU ≈ 1 MIC ≈ 5 High-end CPU
- Cluster configuration - Strong scaling: 1 GPU ≈ 1.5-3.3 MICs depending on the number
- Cluster configuration - Weak scaling: 1 GPU ≈ 2 MICs

In conclusion, clusters of accelerators appear to be a very promising platform for large scale simulations of spin systems (and similar problems) but code tuning and compatibility across different architectures remain open issues. In the near future we expect to extend this activity to more general domain decompositions and problems in higher dimensions.

3.9 AlyaSolidzMIC

Project leader: Mariano Vázquez, BSC
PRACE expert: Félix Rubio, BSC
PRACE facility: BSC's System
Other facilities (if any): No
Research field: Computational Mechanics
Application code: Alya
Project Type: PA Type C
PA number (for Type C project only): 2010PA1739

Project objectives:

Porting and testing of Alya to MICs architectures.

Work done and results:

Alya is a parallel multiphysics computational mechanics code capable of scaling up to thousands of cores. Together with Code Saturne, Alya is one of the CFD codes of the PRACE benchmark suite. It has been ported to several supercomputers of the PRACE ecosystem and its scalability has been assessed. Alya simulates fluid mechanics (compressible and incompressible), solid mechanics, combustion, electromagnetism, etc.

Alya is paralellized using MPI and OpenMP tasks, which can be combined to achieve a hybrid scheme. Therefore, we have planned to assess parallel performance under the three

schemes: MPI, OpenMP and MPI+OpenMP, in the off-load mode (all compiled and running in the MIC). So far we have successfully ported Alya to MICs, running several test cases of progressively more complexity. The largest cases are around 1.5M elements.

In Alya, we have programmed both explicit and implicit schemes. At each time step, explicit schemes have an assembly phase and implicit schemes have both an assembly and a solver phase. Both phases have different programming features. The assembly is basically a set of gather/scatter operations and the solver is matrix-vector and vector-vector operations in sparse format. Therefore, Alya allows testing both schemes.

The selected Alya module is "Solidz", which solves solid mechanics problems. The numerical scheme is FEM-based, in a total-Lagrangian formulation including dynamical terms and large deformations. The case chosen is an arterial aneurism wall deforming under stresses imposed on the inner wall coming from the blood dynamics. The table below shows the scalability figures for the MPI strategy when solving the aneurism case using an implicit scheme. The implicit strategy is a conjugate gradient with a diagonal preconditioner.

Tasks	Time	Efficiency
10	471 min	1.00
30	151 min	1.03
60	76 min	1.03
120	52 min	0.75
240	40 min	0.49

Table 6 Scalability figures for the Alya MPI strategy when solving the aneurism case using an implicit scheme

Alya's MPI strategy is based in a mesh partition done with METIS, assigning each partition to an MPI task. METIS partitions the mesh minimizing inter-domain communication and maximizing load balance. The geometry is 3D, composed of 351465 prisms. The first column is the number of MPI tasks running within the MIC. The second one is the accumulated time of a fixed set of time steps and the third one the relative efficiency (1.00 is a perfect efficiency). The slightly superlinear figures for 30 and 60 cores should be attributed to round-off and time measuring. The fall after 60 is expected because node saturation due to sharing resources, thus diminishing the sequential performance. In any case, these preliminary results are very encouraging: scalability is optimal up to 60.

At this moment we are reprogramming part of the code to improve OpenMP behaviour. After several tests we have realized that scatter involves critical sections that are performance bottlenecks, limiting speedup for higher than 8 or 12 OpenMP tasks per MPI task (depending on the problem). To avoid high dependency on these critical sections, we will renumber the local MPI task meshes.

Conclusion:

The preliminary results are very encouraging: linear scaling is observed up to 60 MPI tasks. Afterwards, speed up still grows but sub-linearly. We are working on improving the OpenMP loops, which will allow us to get a better scalability in the hybrid model.

3.10 OSIMA: Optimising the Smeagol code for the Intel MIC architecture

Project leader: Dr Ivan Rungger, Trinity College Dublin, Ireland
PRACE expert: Dr. Alin M Elena, Dr. Michael Lysaght, ICHEC
PRACE facility: EURORA
Other facilities (if any): Irish Centre for High-End Computing
Research field: physics**,** spintronics
Application code: SMEAGOL
Project Type: PA Type C
PA number (for Type C project only): 2010P1722

Project objectives:

The aim of the project is to port and explore possible optimisations for the *ab initio* electron transport code, SMEAGOL, to the new Intel MIC architecture. SMEAGOL allows for the calculation of quantum electron transport properties of nano-devices at an applied bias voltage between two leads from first principles. See Figure 10, for a typical device.

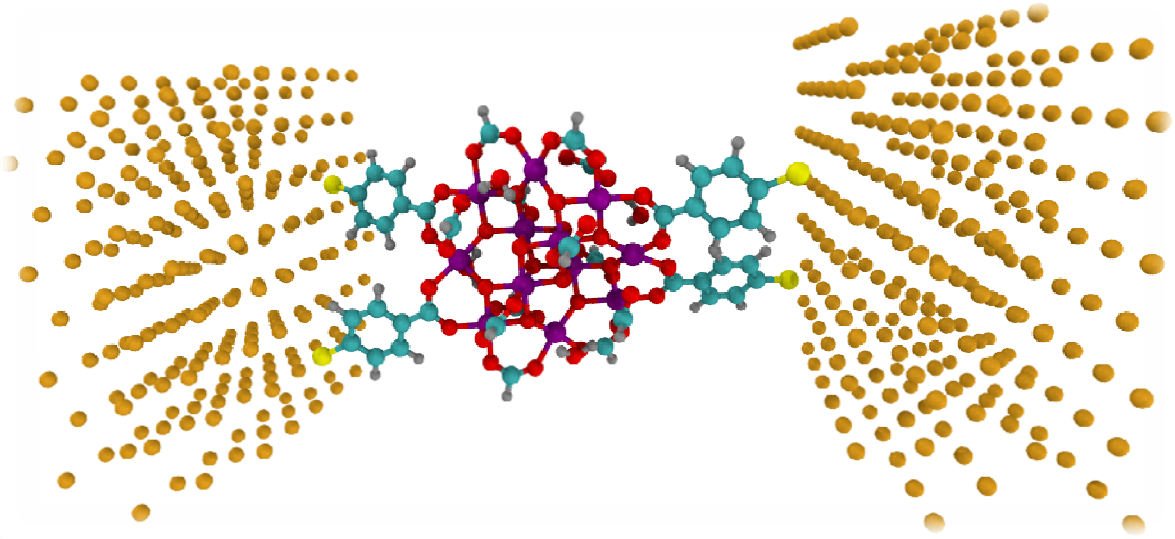


Figure 10 A manganese-12-acetate (Mn12) molecule in between two gold electrodes attached via thiol linkers. The full nano-device consists of 620 atoms. SMEAGOL is used to compute the quantum transport via this molecule when a voltage difference is applied to the electrodes.

Work done to date and results:

The work carried out included four phases. We provide a brief overview of each phase here.

Phase One: *Getting SMEAGOL to run on Xeon Phi*. We managed to compile SMEAGOL for native execution on the Xeon Phi. During this initial phase the build system needed to be extended in order to accommodate for the new architecture. Minor changes to the source code also needed to be made to work around a compiler bug. In the process several other bugs were discovered, reported and confirmed by Intel against their Fortran compiler (DPD200247222, DPD200246949, DPD200247006, DPD200247866). Unfortunately, initial tests were not very successful, as runs in native mode did not pass correctness tests of the code. After lengthy debugging sessions we concluded the most probable source of the issue was a bug in the compiler suite (Intel 13.1.x). Despite our best efforts this could not be reduced to a simple example that could be reported to the compiler developers and, as a result, a workaround was implemented to allow correct results in a limited set of cases. An automatic test-checking framework was put in place by extending 'beetest'. With the release of the new Intel compiler v14.0.0 at the end of September 2013 all the tests passed for the code running in native mode on the Intel Xeon Phi.

***Phase Two**: Initial performance on Xeon Phi.* We have successfully run SMEAGOL for a set of cases in four modes: native, MPI-native, automatic offload mode (MKL) and MPI symmetric. A considerable amount of effort was dedicated to automatizing the submission of jobs on the Xeon Phi system. The scripts will be discussed in greater detail in the associated PRACE white paper. The outcome of this stage was somewhat mixed; we have found that running on the Xeon Phi without any source changes (leveraging only the Xeon Phi version of MKL) results in slower execution times than running on the host for the same problem. (In the MKL offload mode no offload to mic was observed due to the relatively small size of the problem.) The native mode is the slowest mode out of the four but this is somehow expected as SMEAGOL has only small sections of OpenMP threaded code (it is mainly an MPI parallelised code). The MPI symmetric mode is an interesting mode which can benefit our code but further investigations are needed to implement an algorithm to effectively distribute the calculations to take into account the heterogeneous nature of the cluster. We have also found that running pure MPI in native mode does not scale with the number of processes on the MIC. For problems of moderate size, (several of them can fit in the 8GiB of RAM available on the Xeon Phi) where high throughput is desirable we concluded that packing 4-5 jobs on a Xeon Phi can achieve better performance than on the host. In this 'task farm' mode, we have estimated a factor of 2 speedup relative to only the host. Of course the speedup factor is system dependent and we cannot conclude that the same result will be obtained in with other conditions.

***Phase Three:** Investigating bottlenecks.* SMEAGOL relies on several BLAS 3 and LAPACK routines, ZGEMM, for matrix-matrix multiplications, ZGETRF/ZGETRI for performing a square matrix inversion, ZGEEVR matrix diagonalisation and ZGESVD for singular value decomposition. Typical sizes for the matrices involved range between 2000-4000. We fully benchmarked all these routines, determining the maximum size of matrices that can be used in a Xeon Phi and the performance. We have selected two routines which are important for SMEAGOL. ZGEMM, see Figure 11, presents a very good performance (830 GFlops at peak) and this is already achieved for square matrix sizes of order 1000-2000, which is fully in our range of interest. However, these results are complemented by the results for ZGETRF/ZGETRI, see Figure 12; while the peak performance for ZGETRF (the LU decomposition part of the inversion) is approximately 720 GFlops it is only achieved for large sizes of matrices that are out of our range of interest. The last part of the inversion, ZGETRI, performs even more poorly at a peak of 400 GFlops (matrix order ~15000). We have therefore concluded that using MKL routines within SMEAGOL will not help us to efficiently exploit the Xeon Phi.

Phase Four: We have identified in the inversion algorithm a stage which is time consuming and consists of the inversion of several independent small matrices (2000-4000). Part of this stage of the code is a good candidate for partial offloading to the MIC. Work is in progress to test this phase and merge it into SMEAGOL, with results forthcoming in the associated whitepaper.

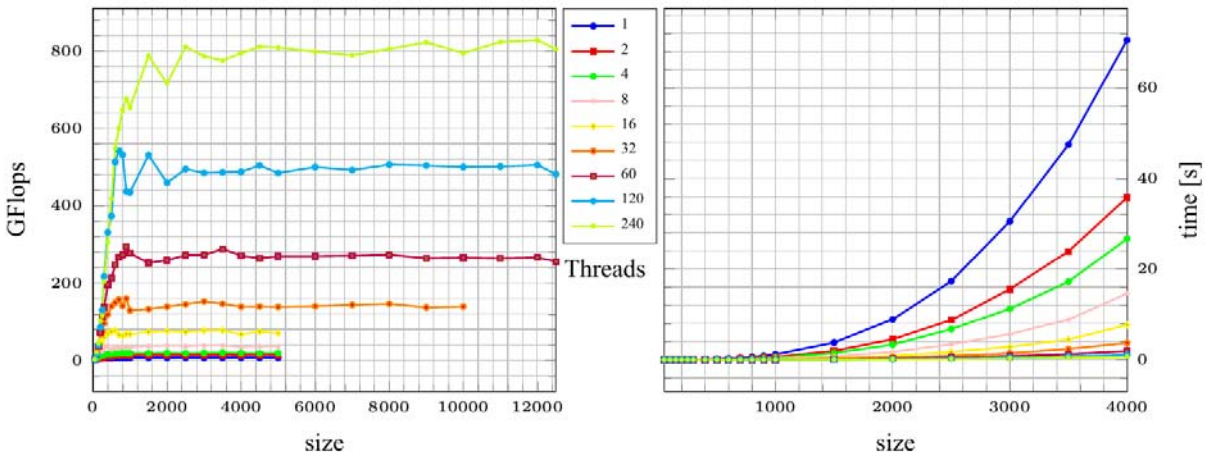


Figure 11 BLAS 3, dense Matrix-Matrix multiply, ZGEMM performance profile for square matrices. Gflops are reported on the left side panel for matrices up to 12500 (maximum that can be fitted in memory) showing a very good peak performance of 830GFlops. The performance in the matrix size ranges of interest is twice as good as of the best CPU we have tested (Intel Xeon v2). However the scaling, shown in the right hand side panel, for average time needed to perform the operation is far from perfect. Affinity used for these results was balanced, similar results are obtained for compact affinity and very bad results, half peak performance, for scatter.

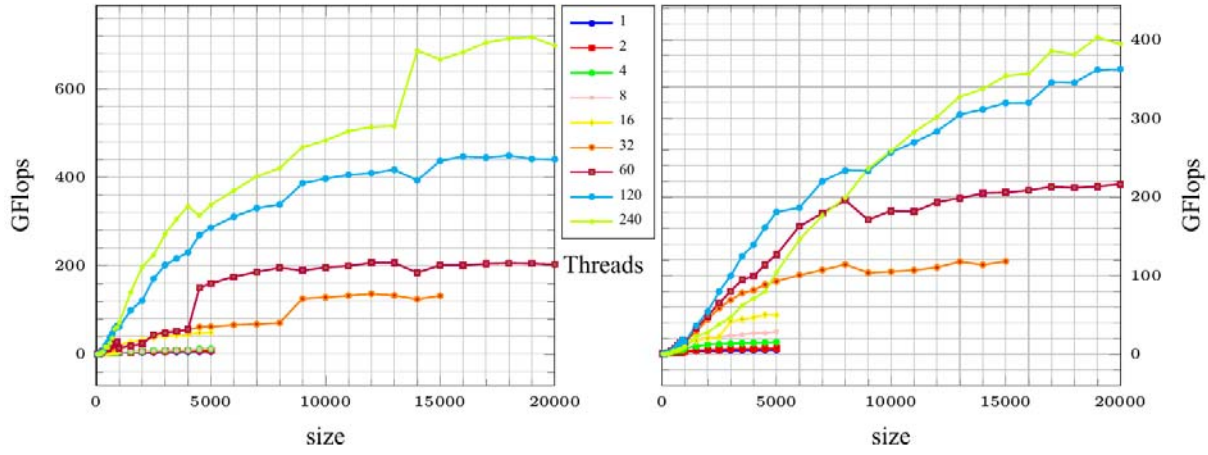


Figure 12 Inverse of a square matrix. It is a two-step process ZGETRF (left panel) followed by a ZGETRI (right panel) call. While the peak performance is good for ZGETRF is good 720GFlops it does not show very good scalability and also increases very slow to reach it. The results for ZGETRI are worst, while maintaining the same slow trend to reach a peak performance of around 400 GFlops. Affinity was balanced and the same performance behaviour as in the case of ZGEMM was observed.

Conclusion:

We have successfully enabled the SMEAGOL electron transport code on the new Intel MIC architecture. A performance analysis of the code running on the Intel Xeon Phi suggest that a combination of small problem size and poor performance of some computational kernels that heavily depend on MKL is the reason for the poor performance so far achieved. These findings have encouraged us to investigate alternative ways for SMEAGOL to exploit the Xeon Phi and we have found that using it in a 'task farming' model can deliver better outcomes than using the host CPU only (when many relative small jobs are needed to be executed). We are currently in the process of restructuring the inversion algorithm using asynchronous computation to investigate further ways of efficiently exploiting the Xeon Phi. A full set of results will be available in the forthcoming PRACE whitepaper associated with this project.

3.11 SuperLU_MCDT (Many Core Distributed) Solver on MIC Architecture

Project leader: Ahmet Duran (UHeM-ITU, Mathematics)
PRACE expert: Ahmet Duran (UHeM-ITU, Mathematics), Serdar Celebi (UHeM-ITU), Mehmet Tuncel (UHeM-ITU) and Bora Akaydin (UHeM-ITU)
PRACE facility: EURORA, Eurotech SandyBridge+Intel MIC hybrid cluster of CINECA and UHeM-ITU
Other facilities (if any):
Research field: Numerical linear algebra
Application code: SuperLU_DIST 3.3 and SuperLU_MCDT 1.0
Project Type: PA Type C
PA number (for Type C project only): 2010PA1756

Project objectives:

Intel MIC coprocessor is a many-core processor chip developed by Intel. It has some similar properties with the Nvidia GPUs like requiring a host processor or efficiency in the embarrassingly parallel tasks such as the vector additions but its architecture is different than Nvidia GPUs.

Work done and results:

Any open source program that runs on the CPU should be modified in order to use the benefits of Intel MIC architecture. These modifications can be summarized in three terms [6]:

- **Symmetric**: The CPU and the coprocessor run different part of the code and they communicate using MPI.
- **Native**: The user states to the compiler using compilation flags that the code is compiled and optimized for Intel MIC coprocessor.
- **Offload**: The programmer states in the code using the 'pragma' directive that some parts of the code should run on coprocessor.

SuperLU_DIST (see [7]) is a suitable base algorithm for developing a solver. However, it has several weaknesses that diminish its practical performance for certain situations (see [8][9] and references therein). We (ITU-UHeM) are developing a linear solver SuperLU_MCDT (Many Core Distributed) utilizing the MPI+X hybridprogramming model to reduce the communication overhead associated with MPI so that better scalability can be achieved, in addition to other new capabilities. We customize and test SuperLU_DIST on "EURORA", Eurotech SandyBridge+Intel MIC hybrid cluster of CINECA.

The aim of our project is to modify the SuperLU_DIST code to run it in many core systems. SuperLU_DIST is a parallel solver for sparse linear systems in distributed systems using MPI. It uses BLAS routines and the performance of these routines affects the overall performance of the library. Running BLAS routines in SuperLU_DIST on coprocessor can improve the performance significantly because the matrix-matrix operations (Level 3 BLAS routines) can be parallelized efficiently.

The "Symmetric" approach requires more work than other two approaches, especially serious structural modifications in the code. Therefore this approach is not preferred.

Some preliminary tests are done before determining the average wall clock time on pure CPU. First, using "native" approach the library is recompiled and adapted to run on coprocessor. To do this, prerequisite libraries (BLAS, Metis and ParMetis) are recompiled first to run on Intel MIC. Coprocessor enabled version of SuperLU_DIST library is run several times and the average wall clock time is compared with the pure CPU timings.

Second, using the "offload" approach, the computationally intensive parts of the library is determined and they are parallelized on the coprocessor using "pragma offload" directive. BLAS library calls are strong candidates for this type of parallelization because they contain intensive computations and also they can be parallelized independently.

There are also some computationally intensive loops in the source code of SuperLU_DIST. The iterations in these loops are not data dependent to each other. Therefore they are parallelized using OpenMP. Intel MIC coprocessor can allow up to 60 OpenMP threads for one chip.

The initial source code and "pragma offload" added versions (offloaded BLAS and OpenMP) of source code are recompiled and linked.

For the last step, the Intel compiler suite comes with its own MIC optimized version of MKL library. SuperLU_DIST is also linked with this version of MKL and tested again. MKL is strictly optimized version of BLAS and it gives better performance in Intel test beds for almost every problem. Therefore, using the MKL library for the MIC architecture can outperform manually "offloaded" BLAS routines. Also MKL library is optimized for MIC architecture implicitly; "offloading" MKL may either do not give any further performance improvement or it can cause decline in the performance.

Conclusion:

We are developing SuperLU_MCDT and extensive tests are needed for large and various types of matrices by using MIC architecture systematically. We believe that SuperLU_MCDT 1.0 using MIC accelerator will show more scalability for large matrices.

3.12 FluxIC porting to the MIC architecture

Project leader: Prof. Jean-Michel GHIDAGLIA - CMLA, ENS de Cachan
PRACE expert: Evgeny Votyakov - The Cyprus Institute (CASTORC)
PRACE facility: EURORA / BSC's System
Other facilities (if any):
Research field: Computational fluid dynamics, Euler solver
Application code: Fortran90
Project Type: PA Type C
PA number (for Type C project only): 2010PA1714

Project objectives:

Improvement of the parallel efficiency of the global solver scheme and communication when synchronising domain patches as well as the bandwidth-intense local solver computations.

Work done and results:

Vectorization of the initialization Assemblages.f90 routine has been performed for the porting to the MIC architecture. Then, the single-fluid test case (sod shock tube) has been checked with the old and new version of the code. The sod shock tube case has an analytical solution, so the numerical solution has been compared with the analytical one in order to get sure that the new code works completely correct. The scalability tests with the new Assemblages.f90 routine are currently run.

Conclusion:

The work is still going on; all the technical problems are resolved step-by-step. As far, it is unclear how much efficiency might be obtained with the new version of the optimised code; however, all the needed scalability tests are expected to be performed in the time of the

project. This project 2010PA1714 is a special case, since the users started really late. Therefore they have been given the extension until March 2014 for their system access.

3.13 Code Optimization and Scalability Testing of Astrophysics Software Gadget on Hybrid Massively Parallel System

Project leader: Prof. Dr. Plamenka Borovska, NCSA
PRACE expert: Plamenka Borovska, Desislava Ivanova, NCSA
PRACE facility: EURORA
Other facilities (if any):
Research field: Astrophysics
Application code: Gadget2
Project Type: PA Type C
PA number (for Type C project only): 2010PA1487

Project objectives:

Gadget2 [10] is a freely available code, widely used for cosmological N-body/SPH simulations to solve a wide range of astrophysical tasks - colliding and merging galaxies, forming of large-scale structure in the space, studying the dynamics of the gaseous intergalactic medium, forming of the stars and its regulation, etc. GADGET is written by Volker Springel to make possible the execution of cosmological N-body/SPH simulations on massively parallel computers with distributed memory.

The objective of this project is code optimization, porting GADGET for the new hybrid computing platforms, and scalability testing of the software GADGET on the Eurora system in order to assess the efficiency of the algorithm. Our work is part of the PRACE-1IP project.

Work done and results:

Based on the project objectives, the following major activities are identified:

Compiling Libraries on Intel Xeon Phi: GADGET uses an explicit communication model and is parallelized by the MPI communication interface. GADGET consists of an integration model, including a tree-code module, a communication scheme for gravitational and SPH forces, a domain decomposition strategy, a novel SPH formulation based on entropy as independent variable, and in the addition of the TreePM functionality. Parallel I/O - MPI I/O and HDF5 have been implemented. The compilation of GADGET code on Intel MIC required the following libraries: *gsl-1.9, zlib-1.2.8, szip-2.1, hdf5-1.8.12 and fftw-2.1.5.*

Compilation of the libraries and GADGET code on Intel MIC was straightforward. The next step is code optimization for Intel MIC. For the project objective a hybrid parallel (MPI+OpenMP) version has been implemented and verified.

Code optimization for Intel MIC: The dynamic analysis of GADGET code is made in the environment of Tuning and Analysis Utilities (TAU). TAU is portable profiling and tracing toolkit for performance analysis of parallel programs [11].

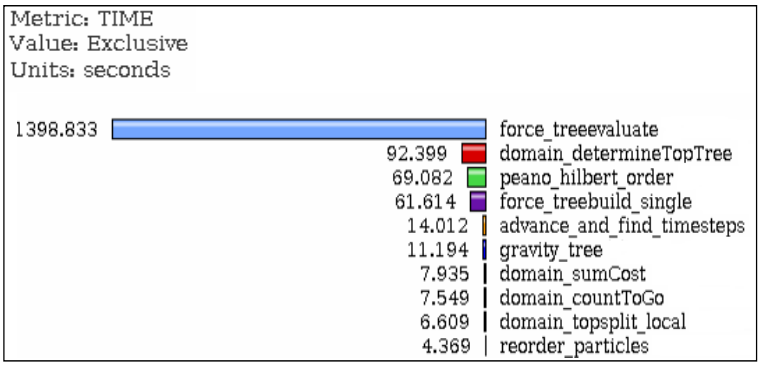


Figure 13 GAGET code analysis with TAU

The code analysis shows the most consuming functions that need to be optimized with OpenMP pragmas for efficient implementation on Intel MIC (Figure 13: GADGET code analysis with TAU).

Implementation and Scalability Tests on Intel Xeon Phi: The execution is performed on one and two MIC accelerators in interactive job session. The software is executed in native mode: all the computations are performed on the MIC accelerators. For scalability testing, the "galaxy" data set is used. The executions are performed on Intel MIC using different MPI-OpenMP configurations with maximum 240 threads on the MIC [12]. The results are shown in Table 7.

Number	MIC	Total thread number per MIC	MPI tasks per MIC	Threads per MPI per MIC	Test (Data Set)	Total execution time [s]
1	1	60	1	60	galaxy	1,75E+003
2	1	120	1	120	galaxy	1,49E+003
3	1	180	1	180	galaxy	1,45E+003
4	1	240	1	240	galaxy	1,49E+003
5	1	60	2	30	galaxy	1,42E+003
6	1	120	2	60	galaxy	1,19E+003
7	1	180	2	90	galaxy	1,16E+003
8	1	240	2	120	galaxy	1,20E+003
9	1	60	3	20	galaxy	1,57E+003
10	1	120	3	40	galaxy	1,30E+003
11	1	180	3	60	galaxy	1,20E+003
12	1	240	3	80	galaxy	1,30E+003
13	1	60	4	15	galaxy	1,69E+003
14	1	120	4	30	galaxy	1,39E+003
15	1	180	4	45	galaxy	1,36E+003
16	1	240	4	60	galaxy	1,41E+003
17	1	56	8	7	galaxy	1,70E+003
18	1	120	8	15	galaxy	1,41E+003
19	1	176	8	22	galaxy	1,38E+003
20	1	240	8	30	galaxy	1,36E+003
21	1	48	16	3	galaxy	1,64E+003
22	1	112	16	7	galaxy	1,36E+003
23	1	176	16	11	galaxy	1,31E+003
24	1	240	16	15	galaxy	1,24E+003
25	2	180	2	90	galaxy	1,06E+003

Table 7 Execution times in [s] for a galaxy data set of the software package GADGET with vary number of MPI tasks and threads per Intel MIC.

Conclusion:

The GADGET code is ported and is available in the Intel MIC architecture (Eurora system). It is best suited to work in homogenous native Xeon Phi MPI mode in a combination of MPI tasks and threads.

3.14 Optimization and Scalability Testing of Massively Parallel Multiple Sequence Alignment Method Based on Artificial Bee Colony Code

Project leader: Prof. Dr. Plamenka Borovska, NCSA
PRACE expert: Veska Gancheva, Nikolay Landzhev, NCSA
PRACE facility: EURORA
Other facilities (if any):
Research field: Bioinformatics
Application code: MSA_BG
Project Type: PA Type C
PA number (for Type C project only): 2010PA1721

Project objectives:

MSA_BG is developed by the Technical University of Sofia. It is based on a parallel version of the Artificial Bee Colony algorithm for solving the Multiple Biological Sequences Alignment problem.

MSA_BG algorithm is inherently massively parallel and is based on the SPMD parallel paradigm. Each process simulates the behavior of a beehive and each hive involves the activities of multiple bee swarms. In designing the parallel algorithm we have applied the methodology taking into consideration the correlation of the parameters of the algorithmic and architectural spaces.

The objective of this project is code optimization, porting MSA_BG for the new hybrid computing platforms, and scalability testing of the software MSA_BG on the Eurora system in order to assess the efficiency of the algorithm. Our work is part of the PRACE-1IP project.

Work done and results:

Based on the project objectives, the following major activities are identified:

Program code development: The software is written entirely in C++ without any external dependencies to third party libraries. It has a hybrid parallel implementation: uses both OpenMP and MPI. The threads simulate the behavior of multiple bees in the swarm. The metaphor: Scout bees in the hive traverse the searching space and construct feasible solutions. Once the scout bees obtain possible (feasible) solution, they return to the hive and begin to dance in the hive. Onlookers watch the dance of employed bees, choose some of the good solutions and perform local search trying to improve the current solution. The quality of solutions obtained is determined by the degree of sequences similarity. The optimality criterion is a maximum score of similarity.

Operations executed on the MIC accelerators:

- Initial alignment of the biological sequences. This is a highly parallel task as there is no dependency between the sequences at this stage.
- Creating the sequence favorite. The favorite sequence is created based on independent calculations on every genome. Again a perfect task for parallel execution on the accelerators.
- Calculating the grades of every genome

- Improving the working set of the sequences. An iterative process that can be executed in parallel.

Scalability testing and result analysis: For scalability testing, the influenza virus protein data set is used. The executions are performed on Intel MIC using different MPI-OpenMP configurations with maximum 240 threads on the MIC. The results are shown in Table 8.

MPI Nodes	Threads	Iterations	Execution Time (Sec)	Data
2	15	10000000	6145.65	H1N1_NA_Human-225.fa (255 sequences)
2	30	10000000	3093.87	H1N1_NA_Human-225.fa (255 sequences)
2	60	10000000	1550.55	H1N1_NA_Human-225.fa (255 sequences)
2	120	10000000	1121.47	H1N1_NA_Human-225.fa (255 sequences)
2	240	10000000	1090.5	H1N1_NA_Human-225.fa (255 sequences)

Table 8 Experimental results of NPI+OpenMP implementation of MSA_BG software on Intel Xeon Phi

Conclusion:

The MSA_BG code is ported and is available in the Intel MIC architecture (EURORA system). It is best suited to work in homogenous native Xeon Phi MPI mode in a combination of MPI tasks and threads.

3.15 Optimization and Scaling of Multiple Sequence Alignment Software ClustalW on Hybrid Massively Parallel System

Project leader: Prof. Dr. Plamenka Borovska, NCSA
PRACE expert: Veska Gancheva, Simeon Tsvetanov, NCSA
PRACE facility: EURORA
Other facilities (if any):
Research field: Bioinformatics
Application code: ClustalW
Project Type: PA Type C
PA number (for Type C project only): 2010PA1728

Project objectives:

The objective of this project is code optimization, porting ClustalW for the new hybrid computing platforms, and scalability testing of the software ClustalW on the Eurora system (Intel Xeon Phi) in order to assess the efficiency of the algorithm. Our work is part of the PRACE-1IP Extension project.

Work done and results:

This activity with the project PRACE-1IP is aimed to investigate and improve the performance of multiple sequence alignment software ClustalW on the Eurora, for the case study of the influenza virus sequences. For this purpose a hybrid parallel (MPI+OpenMP) version has been implemented and verified.

Some experiments have been carried out utilizing parallel program MPI implementation on the basis of ClustalW algorithm. Influenza virus A/H1N1sequences have been used as experimental data.

Detailed information about the duration of the three computational phases of the ClustalW-mpi and ClustalW-mpi-openmp for the case study of the influenza virus A/H1N1 are presented in Table 9. The input file consists of 98 influenza virus protein sequences, each with length of 481 symbols.

Numbers of cores	Execution time of ClustalW-mpi, sec				Execution time of ClustalW-mpi-openmp, sec			
	PA	GT	MA	Total	PA	GT	MA	Total
1	1809,6	0,33	3204,55	5014,48	1809,6	0,33	3204,55	5014,48
20	96,3	0,25	48,2	144,75	58,41	0,24	31,0	89,65
40	49,3	0,24	47,6	97,14	28,45	0,22	30,28	58,95
60	31,64	0,25	46,76	78,65	19,01	0,22	30,27	49,5
120	28,6	0,24	42,8	71,64	16,64	0,22	30,8	47,66
240	28,2	0,24	43,6	72,04	15,6	0,23	31,6	47,43

Table 9 Phases duration time of ClustalW-mpi and ClustalW-mpi-openmp on Eurora using various numbers of cores

The speedup is evaluated as a ratio of the execution time on 1 core to the execution time on 20, 40, 60, 120, and 240 cores respectively. The experimental results for the speedup using various numbers of cores with respect to 1 core for the cases of ClustalW-mpi and ClustalW-mpi-openmp are shown on Figure 14 and Figure 15 respectively.

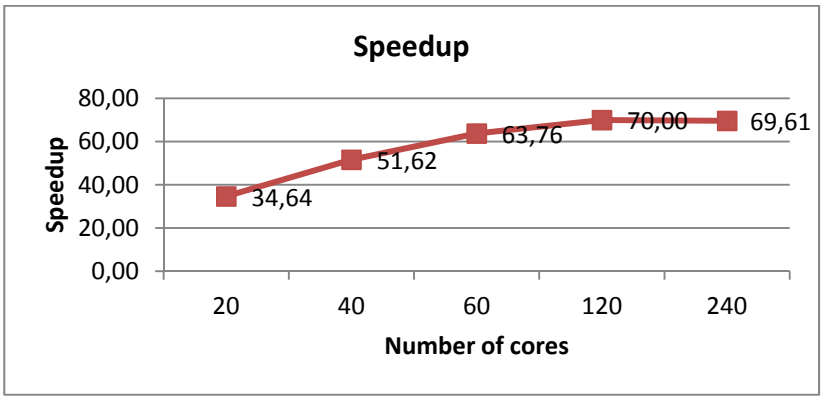


Figure 14 Speedup during the execution of clustalw-mpi on Eurora using various numbers of cores

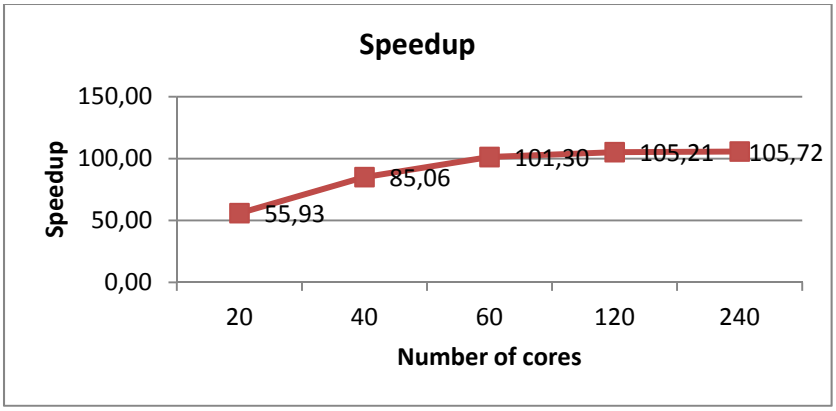


Figure 15 Speedup during the execution of clustalw-mpi-openmp on Eurora using various numbers of cores

Numbers of cores	PA	MA	Total
20	1,65	1,55	1,61
40	1,73	1,57	1,65
60	1,66	1,54	1,59
120	1,72	1,39	1,50
240	1,81	1,38	1,52

Table 10 Clustalw-mpi-openmp execution time vs. Clustalw-mpi on Eurora using various numbers of cores

The experimental results and analyses of ClustalW software show that the execution time during the first stage (pairwise alignment) decreases with increasing the number of cores, while the execution time of the third stage (multiple alignment) remains approximately constant. The experimental results show that the MPI+OpenMP program implementation achieves higher reduction in execution time as the first stage and the third stage. The total execution time is decreased by approximately 1.5 times.

Conclusions:

The ClustalW MPI+OpenMP implementation for multiple sequence alignment is ported and is available in the Intel MIC architecture (Eurora system). It is best suited to work in homogenous native Xeon Phi MPI mode in a combination of MPI tasks and threads.

3.16 Future-proof FEASTFlow: Next Generation Accelerators (FFF-NGA)

Project leader: Prof. Stefan Turek, TU Dortmund
PRACE expert: Georgios Goumas, GRNET
PRACE facility: EURORA
Other facilities (if any):
Research field: Mathematics and Computer Science
Application code: FEASTFLOW
Project Type: PA Type C
PA number (for Type C project only): 2010PA1717

Project objectives:

We aim at the development of efficient, reliable and future-proof numerical schemes and software for the parallel solution of partial differential equations (PDEs) arising in industrial and scientific applications. Here, we are especially interested in technical flows including Fluid-Structure interaction, chemical reaction and multiphase flow behaviour which can be found in a wide variety of (multi-) physics problems. We use a paradigm we call 'hardware-oriented numerics' for the implementation of our simulators: Numerical as well as hardware efficiency are addressed simultaneously in the course of the augmentation process. On the one hand, adjusting data-structures and solvers to a specific domain-patch dominates the asymptotic behaviour of a solver. However, utilising modern multi- and many-core architectures as well as hardware accelerators such as GPUs has become state-of-the-art recently. With this preparatory access application, we aim at evaluating the Intel XEON Phi accelerator as a backend-expansion for our codes.

Work done and results (1st half of the project):

Our work during the first half of the project reported here focused on various system configurations (CPU single threaded, CPU multithreaded, Intel XEON Phi) and programming environments (Pthreads, OpenMP, OpenCL). Our initial experiments with the EURORA system comprised portability checks of our parallel application frameworks on the state-of-

the-art XEON processors (without using the XEON Phi accelerator boards) and the GPUs. Performance on the different hardware architectures and scalability was as expected.

In the following paragraphs we report results on several code/system/programming model configurations.

Single Node OpenCL Application Runs (including Intel XEON Phi): When incorporating the XEON Phi board on a single node with our OpenCL backends, we were unable to obtain a reasonable application performance. Even compared to single threaded CPU code, our OpenCL backends performed worse. Several efforts to investigate the issue on the application level have not shed adequate light to the problem. Hence, we incorporated a microbenchmark kernel (see below) for further investigation.

Single Node Single- and Multicore (PThreads, OpenCL) Runs on the host CPU / OpenCL Microbenchmarks on the Intel Xeon Phi: In order to examine the above-mentioned problem, we concentrated on micro-benchmarking a single BLAS1 like operation available within our framework computing r = a*x + y with vectors r, a, x, y. For convenience, we compare fully optimised versions of the code on the CPU with handcrafted SSE intrinsic code, additionally parallelized on the shared memory level using OpenMP versus the OpenCL version on the XEON Phi. All vectors are aligned and in order to rule out granularity effects, we measure with vector length of 64^4 (double precision) and take the average of ten benchmarks each comprising 20 calls to the operation kernel. The results of the measurements are shown in the figure below.

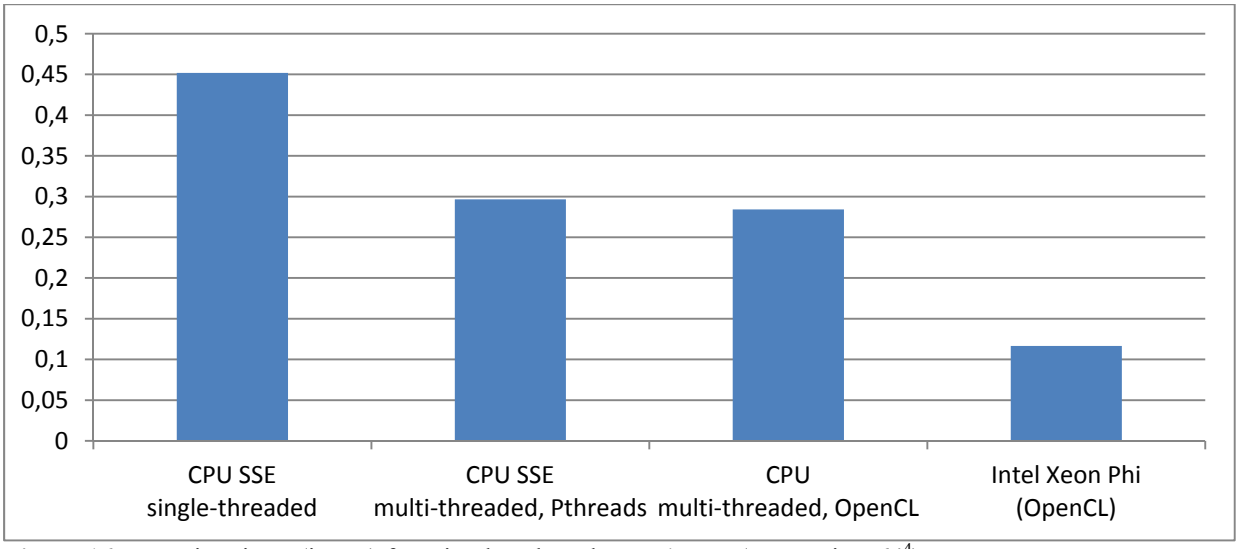


Figure 16 Execution times (in sec) for microbenchmark **r = a*x + y** (vector size: 64^4)

Results indicate that the irregularity obtained on the application level is not present in this micro-kernel benchmark. Roughly 5 to 6 times the performance of the multi-threaded, SSE optimised CPU version can be reached with the OpenCL backend on the accelerator, which is a reasonable speedup given the bandwidth difference between the architectures and a bandwidth-bound code. However, this brought no light into the issue with the CFD application and the subject has to be further examined in the second half of the project.

Conclusion and ongoing work:

Since the OpenCL backend of the application exhibits some subtle execution issues, we will focus our work on the implementation and optimization of representative application kernels on native programming models (e.g. OpenMP). The goals of this approach are (1) experimenting with the different native programming models and their optimisation regarding

performance on the micro-kernel level (see below) and (2) see how they fit into our frameworks. The different approaches we want to evaluate and profile in detail are:

- Xeon Phi native code generated by the Intel 14 cross-compiler (using -mmic) with generic vectorisation left to the compiler (GENERIC)
- Xeon Phi native code generated by the Intel 14 cross-compiler (using -mmic) with generic vectorisation left to the compiler plus OpenMP multithreading (OpenMP-GENERIC)
- Xeon Phi native code generated by the Intel 14 cross-compiler (using -mmic) with handcrafted VPU code using mm512 intrinsics (VPU)
- Xeon Phi native code generated by the Intel 14 cross-compiler (using -mmic) with handcrafted VPU code using mm512 intrinsics plus OpenMP multithreading (OpenMP-VPU)
- Xeon Phi + Host XEON using offloads (OFFLOAD)
- Xeon Phi with MPI + OpenMP (MPI-OpenMP)
- Xeon Phi with MPI + OpenMP + intrinsic VPU code (MPI-OpenMP-VPU)

We plan to implement and evaluate the basic building blocks of a full iterative solver, including vector-vector and sparse matrix vector operations.

3.17 Modelling Complex Oxides using CP2K on Intel Xeon Phi

Project leader: Mr Iain Bethune, EPCC
PRACE expert: Mr Iain Bethune, Dr. Fiona Reid, EPCC
Collaborator: Dr Ben Slater, University College London
PRACE facility: EURORA (CINECA)
Other facilities (if any): Hydra (EPCC), DomMIC (CSCS)
Research field: Chemistry and Materials
Application code: CP2K
Project Type: PA Type C
PA number (for Type C project only): 2010PA1709

Project objectives:

CP2K is a freely available and widely use tool for atomistic simulation in the fields of Computational Chemistry, Materials Science, Condensed Matter Physics and Biochemistry, among others. Today's researchers require robust and portable applications that allow them to tackle complex and challenging problems by taking advantage of the latest advances in computer hardware. CP2K has demonstrated scalability to 10,000s of CPU cores using a mixed-mode MPI/OpenMP parallelisation strategy, and has been deployed on a range of Tier-0 and Tier-1 PRACE systems. The code can make use of GPU accelerators using Nvidia's CUDA programming model, and recent work in PRACE-3IP Task 7.2 ported the code to Intel's MIC (Many Integrated Core) architecture, using the existing parallelisation, although initial performance results were disappointing.

The aim of this work was to optimise CP2K with the aim of improving the performance of a specific test simulation on the Xeon Phi. The systems of interest are the Langasites (see Figure 17), a family of solid oxides composed of Lanthanum, Gallium and Germanium, which have applications as fuel cells and we wish to be able to efficiently computationally screen different orderings of La, Ga and Ge to determine the minimum energy and maximum conductivity structures.

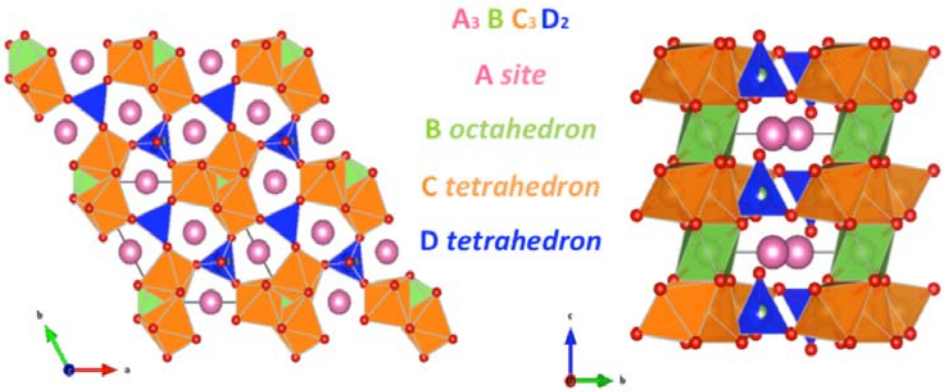


Figure 17 Generic crystal structure of a Langasite (Ben Slater, UCL)

Work done and results:

Initial investigations of the Langasite structural optimisation calculation provided by Dr. Ben Slater showed that this calculation was not well suited to the Xeon Phi. The relatively large 600 Ry cut-off used for the planewave expansion of the electronic density and the large Gaussian basis sets provided resulted in memory usage too large to fit within the 8GB memory available on the Xeon Phi when running on even a subset of the cores available. As a result, we have reduced the accuracy of the simulation in order to fit within the memory constraints available. Benchmarking of this reduced system showed that on the Intel Xeon host nodes on EURORA, the calculation took 131s using 16 MPI processes (73% parallel efficiency compared with running on a single core). On the Xeon Phi in native mode the best performance was achieved using 16 MPI processes with 8 OpenMP threads each, taking 671s for the entire calculation – 5x slower than the host! Detailed profiling showed that much of this disparity was due to several routines which were not OpenMP parallelised: `build_core_ppl`, `build_core_ppnl`, `calculate_ecore_overlap` and `calculate_dispersion_pairpot`.

We have developed an OpenMP parallelisation scheme for these routines, all of which make use of an iterator over a neighbour list data structure, describing the interactions between pairs of atoms/basis functions. The iterator has been extended to allow thread-safe access to the `neighbor_list_iterate()` subroutine and the data dependencies between iterations have been analysed to ensure threads can safely work on shared data, with a minimum of synchronisation. This has been implemented in the `build_core_ppl` subroutine, where we have achieved a 7.5x speedup when using 8 OpenMP threads, and up to 12.3x speedup with 30 threads, resulting in a 10% speedup for the entire code. We are currently implementing the parallel iterator for all of the aforementioned bottleneck routines.

In addition, we have implemented an improved overlapping of threaded computation and MPI communication in a the `rs_distribute_matrix` subroutine, as well as working closely with Intel to improve their support for the FFTW3 interface to MKL, for improved performance of FFTs on the Xeon Phi. Implemented in MKL version 11.1 (released in September 2013), this gives a 20% speedup in the parallel 3D FFT computation in CP2K compared to using the FFTW 3 library directly. We have also implemented work-arounds for a number of bugs in the Intel compiler, which are currently targeted to be fixed in the next major compiler release in 2014.

All of our improvements have been incorporated into the main CP2K SVN trunk, and are available to all users from http://www.cp2k.org.

Conclusion:

While work is still ongoing, and we have been able to demonstrate significant performance improvements in CP2K on the Xeon Phi for simulations of Langasites, we are not yet able to out-perform the host CPU. The principle reason for this is that the current parallelisation of the `qs_collocate_density` kernel, and to a lesser extent the DBCSR sparse matrix library have memory utilisation that increases with the number of threads. As a result we are still only able to use around half of the available cores on the Xeon Phi. To overcome this we propose to investigate using the 'offload' model, where only certain sections of code that are efficiently parallelised are run on the Xeon Phi, similarly to the model used in CUDA. A prime candidate for this would be the FFT computation. We plan to report on the ongoing optimisation work and the use of the offload model in a PRACE white paper to be produced at the end of the project in December 2013.

3.18 Porting and optimising SeisSol on the Intel MIC architecture

Project leader: Volker Weinberg, LRZ
PRACE expert: M. Allalen, G. Brietzke, V. Weinberg, LRZ
PRACE facility: EURORA
Other facilities: Initial tests on KNC prototypes at LRZ, runs on SuperMUC (SandyBridge based thin-node islands) for comparison.
Research field: Geophysics
Application code: SeisSol
Project Type: PA Type C
PA number (for Type C project only): 2010PA1759

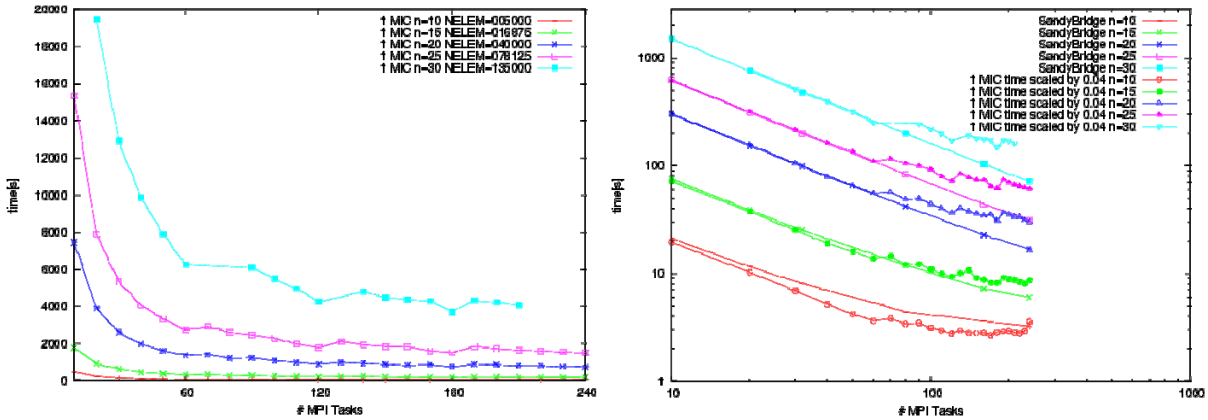
Project objectives:

Porting and optimising the MPI based real-world seismological application SeisSol on the Intel MIC architecture.

Work done and results:

We have ported and analysed the performance of the geophysical application SeisSol on the Intel Xeon Phi based cluster EURORA at CINECA. SeisSol is a code that can be used to simulate earthquake rupture and radiating seismic wave propagation in complex 3-D heterogeneous materials. It is written in Fortran and uses MPI for parallelisation. The main performance bottleneck is the multiplications of relatively small sparse matrices, which dominate the runtime execution.

To fit the rather small memory size of 8 GB on 1 MIC coprocessor we used the CUBE mesh generator by S. Rettenberger, TUM, and the software package METIS to generate computational meshes reaching from 5000 (n=10) to 135000 (n=30) elements. Figure 18 shows the timing results on 1 MIC coprocessor (a) and compares them to the scaling on the SandyBridge-EP based system SuperMUC at LRZ (b). Mind that the theoretical peak performance per physical core on SandyBridge-EP (16 cores/node @ 2.7 GHz, 8 DP Flops/cycle) is 21.6 GFlops and on MIC (60 cores/coprocessor @ 1.03 GHz, 16 DP Flops/cycle) is 16.4 GFlops. On up to 60 MIC cores the scaling behaviour is almost identical, in the regime of hardware hyperthreading on MIC (60 < tasks <= 240) linear scaling breaks, but some additional gain in performance is observed for larger meshes. However, the overall "out-of-the-box" performance on the MIC coprocessor is a factor of 25 slower than on SandyBridge.

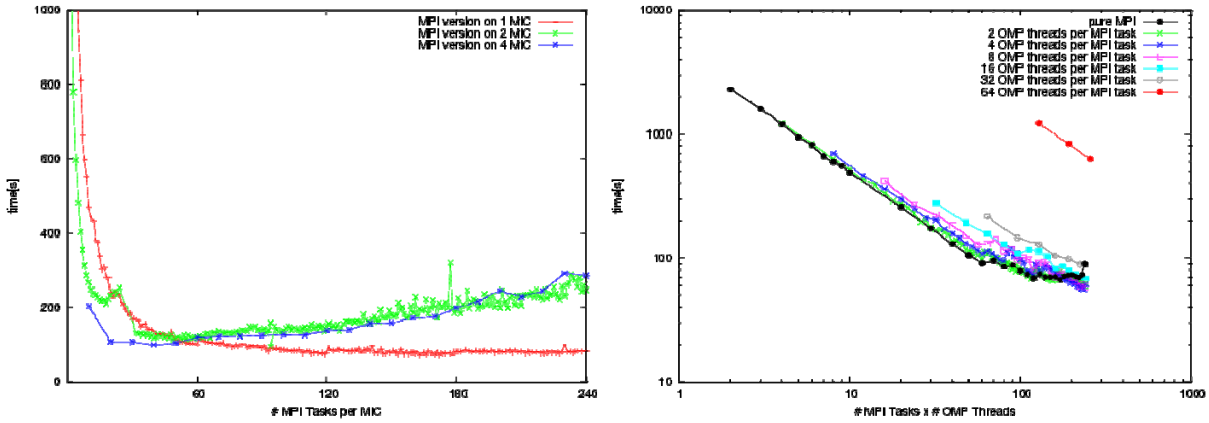


(a) (b)

Figure 18 Timing of SeisSol for 5 different mesh sizes. Fig. (a) shows the execution time on a single MIC co-processor in dependence of the number of MPI tasks. Fig. (b) compares the measurements on MIC with the scaling behaviour on SuperMUC. Mind that the timing results on the MIC coprocessor have been multiplied by 0.04 to fit in the same figure.

Using various pinning settings like *I_MPI_PIN_DOMAIN=core* or *node* or executing the code in native mode using *mpiexec.hydra* directly on *tmpfs* on the MIC coprocessor did not change the performance significantly. Figure 19 (a) shows the timing measurements on multiple MIC coprocessors as a number of the MPI tasks per coprocessor. Using more than one coprocessor did not improve the performance. Negative scaling was observed when using more than 60 tasks per coprocessor.

We have also tested an experimental hybrid MPI and OpenMP version coming with the main code branch of the SeisSol working group. Performance results are shown in Figure 19 (b) and are compared with the pure MPI version of the code. The hybrid version shows lower performance than the MPI-only version, with the performance getting worse with increasing number of OpenMP threads. One possible explanation could be the suboptimal pinning of the threads on the MIC and data partitioning in the job which does not minimize the total effective interprocessor communication.



(a) (b)

Figure 19 Fig. (a) shows the timing of SeisSol on multiple MIC coprocessors for the smallest mesh size as a function of the MPI tasks per coprocessor. Fig. (b) displays the timing of a hybrid version of SeisSol using MPI and OpenMP as a function of the product of the number of MPI tasks and the number of OpenMP threads in comparison with the pure MPI version of the code.

Conclusion:

Concerning the ease of use and the programmability Intel MIC is a promising architecture for scientific computing compared to other accelerator based systems like GPGPUs, FPGAs or former CELL processors or ClearSpeed cards. The MPI based version of SeisSol as well as the experimental combined MPI and OpenMP version could be quickly ported to the new

architecture. The possibility to log on to the coprocessor for diagnostic inspection (e.g. to watch the core and memory usage interactively with tools like *top*) was a great help compared to GPGPU based systems. Scaling on up to 60 MPI tasks on one single coprocessor was almost linear and similar to the scaling on SuperMUC, however the performance was slower by a factor of 25. Also the small GDDR memory size restricted the test cases to rather small meshes. Likely causes for the suboptimal performance are the failing of efficient auto-vectorisation techniques inside the sparse matrix-matrix multiplication routines due to indirect vector-indexing and cache-size limitations. Therefore considerable efforts would need to be made to exploit the 512-bit vector-units of the MIC-architecture. The preliminary scaling tests on multiple coprocessors did not show improvement. Alternative implementations of the compute-kernels by generating vector code using vector-intrinsics are already work in progress and have shown very good scaling on SuperMUC [13]. However, we hope that the next product of the MIC family, announced as "Knights Landing" by Intel, with integrated on-package memory and also functioning in standalone CPU mode, will deliver better "out-of-the-box" performance together with future compiler releases.

3.19 Forward-modeling techniques applied to acoustic or controlled-source electromagnetic data exectued on the Intel Xeon Phi

Project leader: Prof. Borge Arntsen
PRACE expert: Evgney Votyakov, CASTORC
PRACE facility: EURORA
Other facilities (if any):
Research field: Geophysics
Application code: Elastic 3D Full-waveform inversion (FWI3D) and a benchmark portofolio of Finite Difference Time-Domain (FDTD) workloads.
Project Type: PA Type C
PA number (for Type C project only): 2010PA1761

Project status:

The project PI required no enabling support from the PRACE expert, so no technical reporting is included in this deliverable.

3.20 FMPS on MIC

Project leader: Ralf Schneider, HLRS
PRACE expert: Florian Seybold, HLRS; David Horák, VSB; Lubomír Říha, VSB; Václav Hapla, VSB
PRACE facility: EURORA
Other facilities (if any):
Research field: Structural Mechanics
Application code: FMPS
Project Type: PA Type C
PA number (for Type C project only): 2010PA1762

Project objectives:

The Finite Method Programming System (FMPS) is a modular software system, which can be used for the simulation of large deformation coupled thermomechanic processes using the finite element method (FEM). Applications of FMPS include

- quasistatic large deformation analysis of elastic-plastic solids,
- large deformation analysis of viscoplastic solids, and

- instationary heat transfer.

It is used by academic institutions including HRLS as well as consulting engineers.

In this project two PRACE experts groups worked on the parallelisation for a distributed MIC cards system of the solver part:

- HLRS implemented a hybrid MPI/OpenMPI parallel Conjugate Gradient (CG) method using the SlicedELLPACK format for the sparse matrix description.
- VSB implemented a MPI parallel Deflated Conjugated Gradient (DCG) method which interfaces FMPS via the PETSc library.

Work done and results:

HLRS

A performance critical operation within the CG method used to solve finite element problems is the sparse-matrix-vector-multiplication. A suitable sparse-matrix-representation for the Intel MIC architecture is the SlicedELLPACK-format (Alexander Monakov, Anton Lokhmotov, and Arutyun Avetisyan. Automatically tuning sparse matrix-vector multiplication for GPU architectures. In HiPEAC, 2010), which is used in this work.

In the first step, a shared memory implementation of the CG method using Fortran and OpenMP has been implemented and optimised for the Intel Xeon Phi architecture. The same implementation can also be used Sandy Bridge like CPU systems.

The shared memory implementation has been tested using discrete Poisson matrix problems of different degree-of-freedoms (DOFs) on an Intel Xeon Phi card and a Sandy Bridge eight core CPU. Analysing the performance and bandwidth with respect to different DOFs of these tests, one can clearly see a cache effect at around 100,000 DOFs on the CPU, but not on the Xeon Phi card. The Xeon Phi shows clearly better performance than the CPU at around 10 million DOFs upwards, although it seems to saturate at only 27% of the peak bandwidth of 350 GB/s, while the Sandy Bridge CPU shows 68% of its peak bandwidth of 51 GB/s at 100 million DOFs.

In the next step, the shared memory implementation of the CG method has been MPI parallelised for a distributed memory system, yielding a hybrid OpenMP/MPI implementation of the CG solver. Strong scaling test runs of the hybrid OpenMP/MPI solver implementation have been conducted using a problem formulated by FMPS with a fixed number of 185610 DOFs, measured on different machines/configurations:

- Cray XC30 system with Sandy Bridge nodes, one MPI process per NUMA node, eight OMP threads per MPI process, using the Cray MPI library.
- Eurora system with Sandy Bridge nodes, one MPI process per NUMA node, eight OMP threads per MPI process, using the Intel MPI library.
- Eurora system with Intel Xeon Phi cards, one MPI process per card, 236 OMP threads per MPI process, using the Intel MPI library.

One can see a mild speedup at the Cray XC30 system and at the Eurora System when using only CPUs. Using MICs at the Eurora System yields a bad (below one) speedup. This suggests a MPI implementation with quite some optimisation potential concerning communication, e.g. overlapping communication with matrix-vector-multiplication, which could not be realised yet. Also measurements in balanced symmetric mode could not be done yet. It is interesting that the implementation yields quite a different speedup behaviour with regard to CPU only usage and MIC only usage at Eurora, which might suggest a bigger sensitivity of the Xeon Phi cards to distributed memory communication, probably due to the detour of the data via the Sandy Bridge nodes at Eurora.

In addition to the parallelisation steps, an interface method between the matrix storage formats of the FMPS assembly routine (CSR like format) and the newly implemented FMPS solver routine (SlicedELLPACK format) has been created. While the CSR format stores only the upper right part of a symmetric matrix, the SlicedELLPACK format stores the upper right and lower left part of a symmetric matrix in order to enable high bandwidth data access of a matrix-vector-multiplication routine on an Intel Xeon Phi. Due to this difference between the storage formats, the interface implementation involves a binary search to facilitate speedy conversion.

VSB

Parallelization of efficient algorithms for the solution of linear systems

$$Ax = b$$

can be implemented mostly using SPMD technique – distributing data portions among Nc processing units

$$A = \begin{bmatrix} A^1 \\ \vdots \\ A^{Nc} \end{bmatrix}, b = \begin{bmatrix} b^1 \\ \vdots \\ b^{Nc} \end{bmatrix}.$$

This allows algorithms to be almost the same for sequential and parallel cases; only the data structure implementation differs.

The main objective of this subtask is the reduction of the number of CG iterations by means of a simple deflation using aggregations based on domain decomposition – the so-called deflated CG (DCG) method. The implemented DCG should run on nodes accelerated with MICs.

The error is propagated globally by means of an additional coarse problem of the form

$$W^T AWy = W^T Ar,$$

where the columns of the matrix W are a basis for the deflation subspace. For larger coarse problems, for example the SuperLU_Dist parallel solver running on subcommunicators can be used. The simplest way of defining this mapping W for a mesh-based system of equations is by agglomerating the nodes of the mesh into subdomains and then defining a polynomial reconstruction over each of them. In our case the entries in W are unity for the DOFs of this region, and zero for all other DOFs. The DOFs with Dirichlet boundary conditions have to be excluded from the W construction. At the algorithmic level it can be viewed as substracting one more (low-dimensional) search direction Wy from the original CG search direction, so that $p = r - \beta p - Wy$.

For the numerical testing the elastic car engine was discretized to 2,500,000 DOFs and partitioned into 102 or 1,014 subdomains with Metis. The DCG method performs significantly better than CG – the deflation based on 102 subdomains reduced the number of iterations from 14,962 to 2,402 and the compute time from 1,006 sec to 533 sec, the deflation based on 1,014 subdomains reduced then the number of iterations from 16,021 to 2,693 and the compute time from 977 sec to 566 sec. The benchmarks were run on the Cray system HECToR at EPCC.

The coarse problem solution can be accelerated using accelerated nodes. The efficiency of the direct solution of dense coarse problem using the Magma LU solver on CPU only, CPU+GPU, CPU+MIC was tested with following results. The LU factorization using CPU+MIC has speedup 2.0, CPU+GPU has speedup 4.0 when compared to CPU only implementation. For large number of solves, e.g. 1,000, the time for CPU+MIC is 82.9 sec, for CPU+GPU 227.5 sec and for CPU only 232.9 sec, so CPU+MIC is three times faster than

CPU only or CPU+GPU. The use of accelerated nodes using MICs can bring significant time savings. The benchmarks were run on the Bull cluster Anselm at VSB.

Conclusion:

While the hybrid OpenMP/MPI parallelised and optimised CG method implemented by HLRS yields good performance at the shared memory part, the distributed memory part still struggles with communication patterns offering potentials for optimisation, also resulting in a below one speedup when using MIC cards.

VSB shows that the DCG method reduces the number of iterations and the compute time and the use of Intel Xeon Phi accelerated nodes for a coarse problem solution can also bring significant time savings.

3.21 Massively parallel Poisson equation solver for hybrid Intel Xeon - Xeon Phi HPC system

Project leader: Prof. Stoyan Markov, NCSA
PRACE expert: Prof. Stoyan Markov, NCSA
PRACE facility: EURORA
Other facilities (if any):
Research field: Astrophysics, Life Sciences
Application code: Poisson Solver
Project Type: Internal
PA number (for Type C project only):

Project Objective:

The treatment of electrostatics is crucial for the modeling of biologically important interactions at atomistic and molecular level by means of molecular–dynamics simulations. We investigate the possibility to do this by discretely solving the Poisson's equation

$$(\phi_{xx})_{ijk} + (\phi_{yy})_{ijk} + (\phi_{zz})_{ijk} = -f_{ijk},$$

where (i, j, k) denote the three-dimensional lattice indices and $(\phi_{\alpha\alpha})_{ijk}$ is an approximation of the second partial derivative of the scalar potential $\varphi(\vec{r})$. The mesh size along the x, y and z directions is h=0.5Å; i= 1,2,3,…..m; j=1,2,3…n; k= 1,2,3,….p; $m = x/h$; $n = y/h$; $p = z/h$; $\rho = \frac{q}{h^3}$, where q is a point electric charge. For the 27-point stencil finite difference approximation, for the unknowns φ(i,j,k) is obtained the following equation

$$144h^2\rho = -600\varphi_{i,j,k} + 60[\varphi_{i,j,k-1} + \varphi_{i,j,k+1}] + 60[\varphi_{i-1,j,k} + \varphi_{i+1,j,k} + \varphi_{i,j-1,k} + \varphi i,j+1,k + 18\varphi i-1,j-1,k+ \varphi i-1,j+1,k+ \varphi i+1,j-1,k + \varphi i+1,j+1,k +18\varphi i-1,j,k-1+ \varphi i+1,j,k-1+ \varphi i-1,j,k+1+ \varphi i+1,j,k+1+ \varphi i,j-1,k-1+ \varphi i,j+1,k-1+ \varphi i,j-1,k+1+ \varphi i,j+1,k+1+ 3\varphi i+1,j-1,k-1+ \varphi i-1,j-1,k-1+ \varphi i+1,j+1,k-1+ \varphi i+1,j+1,k+1+ \varphi i-1,j-1,k+1+ \varphi i+1,j-1,k+1 + \varphi i-1,j+1,k-1+ \varphi i-1,j+1,k+1 \quad (1)$$

The maximum absolute error of this approximation for $\varphi(\vec{r})$ is $O(h^{-6})$ (1)

Taken along the X-direction, then the Y-direction and at last the Z-direction from eq. (1) we get a large system of linear equations

$$Au(s) = b(s).$$

Detailed description of the system of linear equations can be found in [14].

Biconjugate gradient stabilized method. Prototypes of programs to calculate the Poisson equation using Biconjugate gradient stabilized method (BiCGSTAB, Van der Vorst). A classical BiCGSTAB algorithm is described in [17]. The two multiplications Ap and As used in the algorithm are the most time consuming part. The number of arithmetic operations "multiplication and addition" is 27MNP (27 times M times N times P) for each of the two multiplications.

Parallel BiCGSTAB solver. We have created a new parallel BiCGSTAB algorithm (see [14][15]) to find the solution of the system of linear algebraic equations mentioned above. In order to minimize the MPI communications and achieve good enough performance scalability a hybrid MPI/OpenMP programming model is used. We assume that the number of Intel Xeon Phi coprocessors is at least equal to the number of rows of matrix A, which is P. Each slice is calculated by a Xeon Phi. If the number of coprocessors is at least equal to P, and the number of threads is less or equal to N, then the number of arithmetic operations is proportional to 54 M. Hence, **the speedup** of Parallel BiCGSTAB solver **is proportional to P*N**. If the number of Xeon Phi modules is equal to K, and that of the number of the threads N is greater than 240, then the **acceleration** is proportional to **PN/(Res+1)(L+1)),** where Res = int(P/K) and L = int(N/240).

Work done and results:

Implementation of the BiCGSTAB solver for native execution on Xeon Phi coprocessors: We have developed a parallel implementation of this algorithm in the C programming language with the OpenMP parallelisation model for Xeon Phi systems. We have written and tested an MPI communication scheme for halo exchange in a previous work, so the focus of the current research is the optimization of the Xeon Phi part of the code. Here we present characteristics and results with the code written for native execution on the Xeon Phi. The most important characteristics of an application that has to perform well on the Xeon Phi are the vectorization and scaling capabilities of the code [16]. We have put a lot of effort in vectorization of all possible loops by pragma directives, clean loop bodies and by maintaining better structured data. In addition we have aligned the data at 64 bytes, according to Xeon Phi architecture, as recommended in [16]. Nevertheless, our observation is that data allocation is relatively slow and poorly scalable.

Results: We have tested our code on different problem sizes on different number of threads. The code scales well all up to 240 threads for the bigger system sizes.

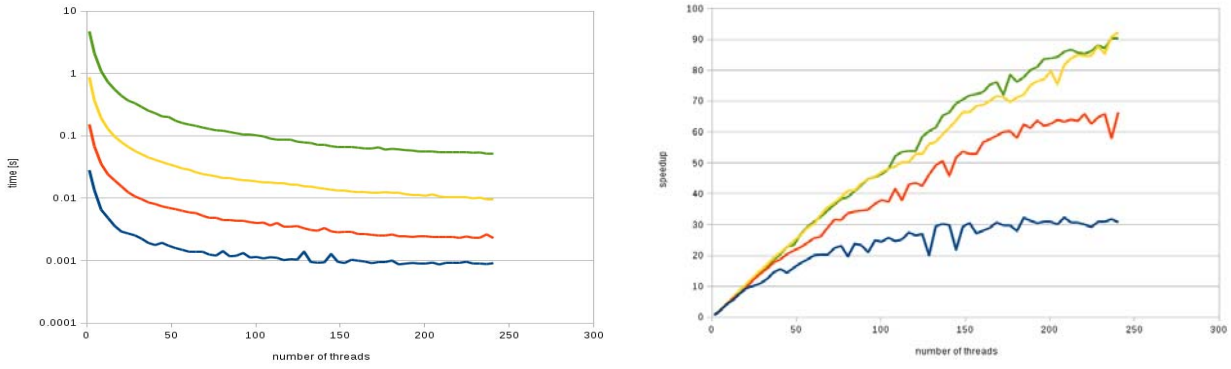


Figure 20 Execution time (left) and speedup (right) as a function of the number of the Intel Xeon Phi threads for one iteration for different problem size (dimensions of the grid) 30x30x30 (blue line), 60x60x60 (red line), 120x120x120 (yellow line) and 240x240x24 (green line).

For bigger problem sizes 120x120x120 and 240x240x240 grid points we observe good scaling. The time per iteration drops with the increase of the total amount of threads and speedup is nearly linear as shown in Figure 20.

Conclusions:

In conclusion we would like to stress that vectorization is crucial to performance on the Xeon Phi coprocessor but it is not a free gift. One must pay attention to the way one stores and accesses data. Often it is necessary to allocate extra variables only to maintain clean data access. The vectorization can be enforced by explicit directives in the code, but one should be careful, because it might change the meaning of the code and affect the results, as advised in [16].

3.22 Porting GPAW to Intel Xeon Phi

Project leader: Jussi Enkovaara, CSC
PRACE expert: Jussi Enkovaara, CSC
PRACE facility: EURORA / BSC's System
Other facilities (if any):
Research field: Chemistry and materials
Application code: GPAW
Project Type: Internal
PA number (for Type C project only):

Project objectives:

GPAW is a widely used software package for various simulations of the electronic structure of nanomaterials within density-functional theory and time-dependent density-functional theory. GPAW performs and scales well on traditional CPUs, and the software is part of the PRACE application benchmark suite. Currently a CPU only version of GPAW is able to scale to thousands of cores and it is used in several research groups worldwide. Having a Xeon Phi accelerated version of the code capable of good scaling behaviour and orders of magnitude speed-ups would be hugely beneficial for the community. In this project we will port GPAW to Eurora, and investigate its performance with Intel Xeon Phi processors. As we have no previous experience on Xeon Phis, the possible performance bottlenecks are not known in advance, and the main objective of the project is to gather information for possible future optimization work.

Work done and results:

GPAW is implemented in a combination of Python and C. All the high level algorithms are implemented in Python, and the computationally intensive numerical kernels are implemented in C (or utilize libraries).

In large calculations $O(N^3)$ parts of the simulation dominate the compute time. These are mainly matrix-matrix operations, and as they offer the highest potential speed-up via optimized library routines, we focused first on matrix-matrix products which are typical for GPAW. GPAW operates mainly with sparse matrices whose representative dimensions within a single node would be 400-800 x 30 000-150 000.

As first step we investigated the automatic offload capabilities of Intel MKL library. For the above type of matrices automatic offloading did not seem to offer any speedup (for square matrices we were able to observe speed ups of 2 to 4). Thus, we moved next to the native execution model.

In order to enable the native execution mode on Xeon Phis, we have ported the Python interpreter to Xeon Phi. The porting of Python is not completely trivial, and we have documented the procedure in the GPAW installation guide [18]. Other external dependencies of GPAW that need to be ported are NumPy Python package and the LibXC library, however, their porting is straightforward. The porting of GPAW itself is also relatively straightforward, however, as the main code version uses only MPI parallellization we have used the multithreaded development branch of the code. The whole build process for Xeon Phi is documented in the aforementioned installation guide.

With the native execution mode, we focused first on matrix – matrix products. The results are summarized in the table below. For Xeon, 16 threads were used and for Xeon Phi 244 which gave the best performance. In both cases the affinity setting KMP_AFFINITY=compact is used.

	Xeon	Xeon Phi
dgemm + setup	9.6 s	6.3 s
dgemm	8.9 s	2.1 s

Table 11 Performance of matrix products.

As can be seen, the actual library routine is sped up by factor of ~4, however, the setup which is minor in Xeon takes more time than the actual calculation on Xeon Phi. The reason for high setup time on Xeon Phi is not completely clear at the moment, but could be related to the fact that setup is performed in Python code.

The fact that the algorithm with the potential for the highest performance can be sped up by only factor of 1.5 is not very promising, and a full calculation of C_{60} molecule is considerably faster on Xeon than on Xeon Phi. One should, however, note that the multithreading in GPAW has been originally designed only for 10-20 threads, and cannot thus utilize Xeon Phi with full efficiency. We have been able to optimize some basic finite-difference kernels so that Xeon Phi is 1.5-2 times faster, however, real calculation involves also some setup steps which seem to suffer from similar performance problems as the matrix multiplication part.

Conclusion:

At the moment, the performance of GPAW on Xeon Phi is inferior to ordinary Xeons. Even though some individual library routines and kernels can be 1.5 – 4.2 faster on Xeon Phi the high setup times and the fact that some kernels can utilize efficiently only 10-20 threads make the full algorithm slow on Xeon Phi. In standard CPU based systems the overheads from using Python in high level algorithms are typically a few percent, but one should assess whether the high setup times result from using Python on Xeon Phi. One possibility is to rely more on explicit offloading (not just automatic offloading with libraries), *i.e.* execute Python parts on Xeon and kernels and library routines on Xeon Phi. Furthermore, all the kernels in GPAW should be redesigned in such a fashion that hundreds of threads can be used. As the number of kernels is relatively small, this task is not that formidable than it appears in first place.

3.23 Offloading ElmerSolver kernels to Xeon Phi

Project leader: Mikko Byckling, CSC
PRACE expert: Mikko Byckling, CSC
PRACE facility: EURORA
Other facilities (if any): CSC Prace prototype system
Research field: Finite Elements, Numerical Analysis
Application code: Elmer
Project Type: Internal

PA number (for Type C project only):

Project objectives:

The project goal is to offload two main ElmerSolver kernels, Finite Element (FE) assembly and iterative solution of the linear system, to run on a single Xeon Phi card.

In order to efficiently offload the Elmer FE assembly kernel, the vectorization of the FE basis function computation and construction of the local stiffness matrix needs to be improved. Due to historical reasons, in Elmer the basis functions are computed for each Gauss point separately, which is an inherently scalar operation. In addition, when the number of Gauss points is large, this creates unnecessary overhead due to increasing number of calls to internal functions. A more efficient approach is to compute the values of the basis functions for all Gauss points in bulk and then use BLAS2 and BLAS3 operations to compute the local stiffness matrix.

In Elmer, iterative solution of the arising linear system is commonly done either with the conjugate gradient (CG) or with the Biconjugate Gradient Stabilized (BiCGStab) methods. In order to minimize threading overhead and thus maximize the potential of the many-core environment of Xeon Phi, both of the methods were re-implemented in a NUMA aware way, which to our knowledge is new. The idea is to localize memory access wherever possible, avoid false sharing on global arrays by accessing work vectors on chunks of 64-bytes and to avoid unnecessary fork/join overhead by letting the threads run concurrently throughout the whole algorithm.

Work done and results:

Finite element assembly: We implemented vectorized versions of the element basis functions, the related transformations from local to global function space and a simple proof-of-concept Poisson solver into a development version of Elmer. The results comparing a dual Xeon E5 platform with a single Xeon Phi 7110 card are presented in the following. Time to perform launch the offload kernels and perform data transfers was not taken into account. As a test problem, we used a 3D Poisson problem with 1M hexahedral finite elements with varying number of Gauss integration points per element.

Gauss points	**1**	**16**	**32**
Standard implementation			
8	52.01s	5.90s	3.02s
64	378.46s	23.84s	21.26s
Vectorized implementation			
8	11.40s	1.40s	0.76s
64	42.53s	2.76s	2.73s

Table 12 FE assembly, Dual Xeon E5 2670, assembly time (in seconds) versus number of threads.

Gauss points	**1**	**60**	**240**
Standard implementation			
8	698.42s	11.88s	7.04s
64	5034.97s	94.09s	47.57s
Vectorized implementation			
8	167.10s	3.04s	2.37s
64	566.46s	9.19s	6.36s

Table 13 FE assembly, single Xeon Phi 7110, assembly time (in seconds) versus number of threads.

Iterative linear system solution: We implemented NUMA aware versions for both the CG and BiCGStab methods with OpenMP. The results comparing a dual Xeon E5 platform with a single Xeon Phi 7110 card are presented in the following. Time to perform launch the offload

kernels and perform data transfers was not taken into account. As a test problem we used a linear system arising from a 3D linear elasticity equation with a mesh size of 7k, 32k and 100k hexahedral elements (small, medium and large test cases, respectively).

Problem	**1**	**16**	**32**
Standard implementation			
3D elasticity, small	1,13s	0,25s	0,33s
3D elasticity_med	8,59s	2,51s	2,67s
3D elasticity_large	36,34s	10,39s	10,72s
NUMA aware implementation			
3D elasticity, small	1,15s	0,20s	0,07s
3D elasticity, med	8,83s	2,27s	1,28s
3D elasticity, large	36,49s	10,76s	5,69s

Table 14 CG, Dual Xeon E5 2670, solution time (in seconds) versus number of threads

Problem	**1**	**60**	**240**
Standard implementation			
3D elasticity, small	10,23s	3,22s	7,46s
3D elasticity, med	72,53s	6,17s	11,24s
3D elasticity, large	303,4s	9,87s	15,85s
NUMA aware implementation			
3D elasticity, small	10,19s	0,35s	0,59s
3D elasticity, med	73,12s	1,86s	1,69s
3D elasticity, large	306,2s	7,14s	4,51s

Table 15 CG, single Xeon Phi 7110, solution time (in seconds) versus number of threads.

Problem	**1**	**16**	**32**
Standard implementation			
3D elasticity, small	2,06s	0,49s	0,62s
3D elasticity, med	14,78s	4,64s	5,57s
3D elasticity, large	66,61s	19,75s	22,29s
NUMA aware implementation			
3D elasticity, small	2,06s	0,32s	0,12s
3D elasticity, med	15,20s	4,99s	2,31s
3D elasticity, large	66,63s	18,90s	9,92s

Table 16 BiCGStab, Dual Xeon E5 2670, solution time (in seconds) versus number of threads.

Problem	**1**	**60**	**240**
Standard implementation			
3D elasticity, small	17,81s	2,17s	8,56s
3D elasticity, med	138,62s	6,41s	14,00s
3D elasticity, large	579,36s	20,87s	25,32s
NUMA aware implementation			
3D elasticity, small	17,74s	0,50s	0,74s
3D elasticity, med	139,43s	3,23s	2,27s
3D elasticity, large	579,33s	10,91s	7,19s

Table 17 BiCGStab, single Xeon Phi 7110, solution time (in seconds) versus number of threads.

Conclusion:

Implementing ElmerSolver kernels on Xeon Phi was straightforward but surprisingly challenging. The results for FE assembly were a bit disappointing, since Xeon Phi was not able to match the performance of a dual Xeon E5 platform. Investigating the issue, we found out that the gluing of local stiffness matrix to the global matrix is currently not properly

vectorized and also requires a fair amount synchronization between the threads. These reasons seem to be holding Xeon Phi back from reaching the performance levels of a regular Xeon. On the other hand, the vectorized version of the assembly is almost 10x faster on a regular Xeon (see Table 14 and Table 15 for one thread), meaning that performance has be improved in the existing implementation. Investigation is ongoing on how to implement the gluing process efficiently both in terms of vectorization and multithreading.

As for the linear solution of the linear system, when implemented in a NUMA aware way, a single Xeon Phi is able to match and surpass the performance levels of a dual Xeon E5 platform. Again, even when run on a regular Xeon platform, the NUMA aware code is 2x faster than the previously implemented version. We are currently doing research on how to perform preconditioning in parallel in order to guarantee convergence of the methods for ill-conditioned problems.

3.24 Private-Cache-Aware Parallel Sparse Matrix-Matrix Multiplication on the MIC Architecture

Project leader: Cevdet Aykanat (Bilkent University)
Contributor: Kadir Akbudak (Bilkent University)
PRACE expert: Cevdet Aykanat (Bilkent University)
PRACE facility: EURORA
Other facilities (if any): Local system which has the same MIC coprocessors that EURORA has. (Intel Xeon Phi 5110P)
Research field: Mathematics and Computer Science
Application code: Sparse Matrix Multiplication
Project Type: Internal
PA number (for Type C project only):

Project objectives:

Sparse matrix-matrix multiplication (SpMM) is an important operation in a wide range of scientific applications such as finite element simulations based on domain decomposition (e.g., finite element tearing and interconnect (FETI)), molecular dynamics (e.g., CP2K) computational fluid dynamics, climate simulation, interior point methods. All of these applications exploit parallel processing technology to reduce running times. There exist several software packages that provide SpMM computation for distributed memory architectures such as Trilinos, Combinatorial BLAS and for GPUs such as CUSP and CUSPARSE.

The objective of this PRACE project is to investigate the performance of the Xeon Phi processor for SpMM operation. In order to attain maximum performance on the Xeon Phi architecture, our objective is also to investigate matrix ordering and thread allocation schemes for better utilization of temporal locality to reduce the cache misses in the fully-coherent cache implementation of the Xeon Phi architecture. We will design, develop, and implement a "private-cache-aware parallel SpMM library (cSpMM)" to run on a single node of the MIC cluster.

Work done and results:

We first investigate implementation of outer-product-parallel SpMM of the form $C = A \times B$ on the Xeon Phi architecture. Outer-product-parallel sparse SpMM is based on one-dimensional columnwise and one-dimensional rowwise partitioning of the input matrices A and B as follows:

$$\bar{A} = AP = [A_1^c \; A_2^c \; \ldots \; A_K^c \;] \quad \text{and} \quad \bar{B} = PB = \begin{bmatrix} B_1^r \\ B_2^r \\ \vdots \\ B_K^r \end{bmatrix}$$

Here, K denotes the number of parts and P denotes the permutation matrix obtained from partitioning. The use of the same permutation matrix for column reordering of A and row reordering of B enables conformable columnwise and rowwise partitioning of matrices A and B. In this input partitioning, each outer product $C^k = A_k^c \times B_k^r$ will be assigned to a thread that will be executed by an individual core of the Xeon Phi architecture. The nice property of the outer-product formulation is that all nonzeros of the A and B matrices are read only once from the device memory. However, multiple writes to the device memory will be needed since the output matrix C is computed as follows in terms of the results of thread computations: $C = C^1 + \cdots + C^k + \cdots + C^K$.

In this project, we propose and implement a hypergraph-partitioning-based method to attain temporal locality in the above-mentioned multiple writes to device memory. In this scheme, input matrices A and B are partitioned recursively and conformably until the size of each output matrix C^k drops below the size of Level 2 cache of the cores of the Xeon Phi architecture. The objective in this partitioning is to allocate A-matrix columns and B-matrix rows that contribute to the same C-matrix entries into the same parts as much as possible. This in turn corresponds to forcing the execution of the outer-product computations that contribute to the same output C-matrix entries on the same core. The proposed hypergraph model encodes this objective successfully thus leading to exploiting the temporal locality in multiple writes to the device memory.

In order to evaluate the validity of the proposed model, we compare the performance of our method to that of the binpacking method that only considers balancing the computational loads of threads. Note that number of resulting parts becomes much larger than the number of threads thus enabling the utilization of dynamic part-to-thread scheduling for further load balancing in both methods. Figure 21 shows performance variation of these two schemes with increasing number of threads for the multiplication of the small *feti-B02* matrix with 612 rows, 152.826 columns, and 920.433 nonzeros. Note that $B = A^T$ in *FETI* application so that the SpMM operation is the form of $C = AA^T$. As seen in the Figure 21, the proposed scheme performs considerably better than the baseline algorithm.

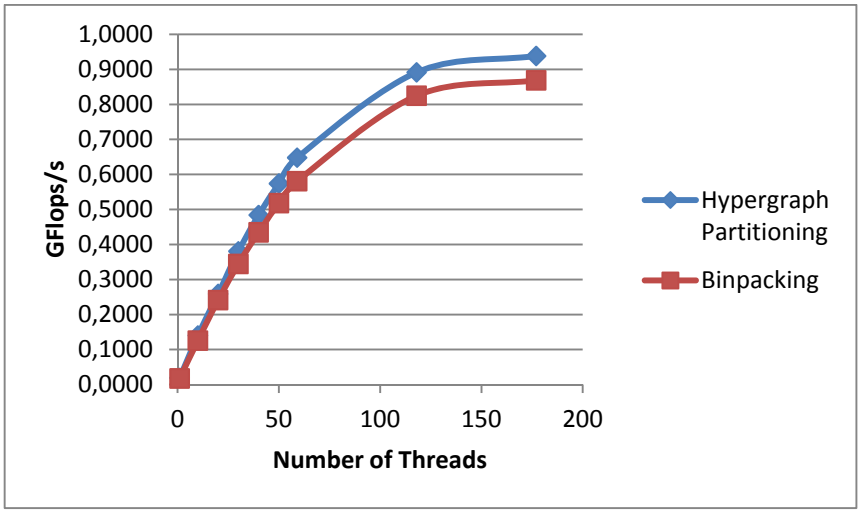


Figure 21 Outer-product-parallel SpMM performance for *feti-B02* matrix

Conclusion:

An outer-product SpMM algorithm was designed, developed, and implemented for the Xeon Phi architecture. Initial experimental results show that the proposed hypergraph-partitioning-based input-matrix partitioning method achieves exploiting temporal locality in multiple writes to the device memory.

3.25 Porting and verification of the ExaFMM library to Intel Xeon Phi-based distributed architecture

Project leader: Dr. Valentin Pavlov, NCSA
PRACE expert: Dr. Valentin Pavlov, NCSA
PRACE facility: EURORA
Other facilities (if any):
Research field: Astrophysics, Life Sciences
Application code: ExaFMM
Project Type: Internal
PA number (for Type C project only):

Project objectives:

ExaFMM ([19][20][21]) is a highly scalable implementation of the Fast Multipole Method (FMM) – an O(N) algorithm for N-body interaction with applications in gravitational and electrostatic simulations. The authors report scaling on large systems with O(100k) cores with support for MPI, OpenMP and SIMD vectorisation.The library also includes GPU kernels capable of running on a multi-GPU system. The objective of the project is to enable the use of ExaFMM solver in the MIC architecture by performing porting, verification, scalability testing and providing configuration suggestions to its potential users.

Work done and results:

Based on the project objectives, the following major activities are identified:

Porting of the ExaFMM code to Intel MIC architecture: The ExaFMM library is a C++ implementation of the FMM algorithm. It is actively being developed and this research is based on the version of the source code that was current at the start of the project, continuously merging with the upstream source code repository [22]. The last such merge was on 11/05/2013.

The library uses MPI for inter-node communication and has several options for intra-node communication – OpenMP, TBB and MassiveThreads. In the P2P and direct kernels, it includes hand-crafted SIMD vectorisation code and a C++ template class for working with vectors of 16 floats (8 doubles) that utilizes the Intel MIC intrisics for working with its 512-bit registers.

Compilation of the library for Intel MIC was straightforward. After initial compilation the possibilities for optimising the code specifically for the MIC architecture in several aspects were investigated:

1. Kernel offloading

The algorithm works by successively dissecting the domain into octants and building a multi-level oct-tree. This oct-tree is traversed in several passes and each box and pair of boxes is subjected to evaluation of a certain kernel. In the GPU version of the library, the core tree building and traversal is performed on the CPU, while the computational kernels that work inside each box and between pairs of boxes are CUDA kernels and are executed on the GPU.

We tried to match this technique for the MIC architecture and have the controlling CPU work on the tree building and traversal while the computational kernels are being offloaded to a MIC device, but it proved quite inefficient. In a typical $O(10^6)$ simulation there are tens of thousands of boxes (offload calls) while the kernels work on the order of 10^5 computations. Communication delays are significant while the payload is not that great. Thus, the idea of offloading the kernels was abandoned as not appropriate for this kind of architecture. Native execution mode was favoured as the better alternative.

2. Multi-threading

The authors claims support for TBB, OpenMP and MassiveThreads for intra-node parallelism. For this project we decided to base our investigations on OpenMP, being the simplest possible and providing no additional levels of indirection. It turned out that the library support for OpenMP was broken and either did not spawn multiple threads or when it did, the program crashed. We fixed the problems and the code now works with OpenMP. However, it should be noted that given the nature of the core algorithm (oct-tree traversing), the library is built using recursive functions. In this scenario it uses task-based parallelism for spawning additional threads for traversing the tree in parallel, rather than using multiple-threads for performing the computations in the kernels themselves.

3. SIMD Vectorisation

The great computational power of the MIC architecture comes from the 512-bit registers that allow simultaneous execution of 16 single-float or 8 double-float operations in SIMD fashion. Thus, in order to see improvement in performance, one has to make sure that the computation is properly vectorised. The library contains a definition of a vector object of 16 floats (or 8 doubles) that uses the MIC intrisics for working with 512-bit registers, but this vector class was only used in a hand-crafted vectorisation of the P2P (direct) kernel. Our investigations showed that by slightly restructuring the non-vectorised P2P loop and allowing the compiler to auto-vectorise it, we can achieve slightly better performance. The P2P kernel is one of the two kernels that take up the majority of computing time, thus this optimization is desirable. The other kernel that takes up most of the time is M2L kernel, which in the original code is not vectorised at all, since it is not based on loops (no auto-vectorisaion) and not based on 512-bit intrisics (no hand-crafter vectorisaion). By rewriting the most computationally intensive subroutines to use the 512-bit intrisics, we obtained improvement of performance.

We prepared build-time configuration directives that can be used to compile the code in another facility.

Verification of the port: Verification of the port was done by calculating the mean squared error (MSE) of a randomly chosen subset of 1,000 bodies between potentials obtained by running FMM calculations on 1,000,000 bodies and the potentials obtained for the test subset using direct pair-to-pair interaction. For 10-order multipole expansion we got MSE of the order $O(10^{-6})$, which is in accordance with the expectations. The same test was run on non-MIC architecture and showed similar MSE.

Scalability testing and suggested configurations: For scalability testing we used the following approach: we testing the code using a fixed number of particles and time steps on single MIC co-processor using different MPI-OpenMP configurations that result in maximum 240 threads on the MIC. The domain decomposition technique used in the library and the related building of locally essential tree for communication with other ranks restricts the number of MPI ranks to be a power of 2. We used scattering affinity for the threads, which proved to be more efficient than the compact one. The results are shown in Table 18 Execution times in [s] for a system of 1,000,000 bodies, configured for max 1024 bodies per cell and scattering thread affinity. Maximum performance is achieved when running 32 MPI

tasks each spawning 3 threads. We interpret this result as being due to the fact that 1) at least two threads per core are needed to satisfy the MIC scheduling another thread every cycle (as explained in [23], p. 31, last paragraph); and 2) scheduled threads are computationally intensive and saturate their respective cores, so using more than 2 threads per core is not efficient.

MPI Tasks	Number of threads per MPI task							
	1	**3**	**7**	**15**	**30**	**60**	**120**	**240**
1	375.58	143.35	75.37	51.96	44.35	42.02	46.34	62.37
2	193.64	74.30	40.45	30.79	27.41	30.09	39.76	
4	95.81	38.22	22.91	18.59	19.25	25.81		
8	50.33	20.82	13.43	13.28	16.75			
16	23.37	10.26	8.45	10.25				
32	11.80	**7.00**	7.54					
64	25.41	29.72						
128	35.82							

Table 18 Execution times in [s] for a system of 1,000,000 bodies, configured for max 1024 bodies per cell and scattering thread affinity.

Conclusion:

The ExaFMM code is ported and is available in the MIC architecture. It is best suited to work in homogenous native Xeon Phi MPI mode, in a combination of MPI tasks and threads that result in a total number of threads being around 120.

3.26 Porting and verification of the ScaFaCoS/FMM library to Intel Xeon Phi-based distributed architecture

Project leader: Prof. Stoyan Markov, NCSA
PRACE expert: Dr. Valentin Pavlov, NCSA
PRACE facility: EURORA
Other facilities (if any): none
Research field: Astrophysics, Life Sciences
Application code: ScaFaCoS/FMM
Project Type: Internal
PA number (for Type C project only):

Project objectives:

ScaFaCos [24][25]is a parallel library that includes various methods for solving gravitational and electrostatic problems in large particle simulations. One of the methods included in the library - Fast Multipole Method (FMM) [26]- is gaining popularity in recent years due to its asymptotic complexity of O(N) where N is the number of particles and its versatility regarding boundary conditions. The objective of the project is to enable the use of the ScaFaCoS's FMM solver in the MIC architecture by performing porting, verification, scalability testing and providing configuration suggestions to its potential users.

Work done and results:

Based on the project objectives, the following major activities are identified:

Porting of the ScaFaCos/FMM code to Intel MIC architecture: The ScaFaCos/FMM code quite large and complex (~200 MB) and includes a mixture of F77, F90 and C codes, along with several 3rd party infrastructure and communication libraries. Given that the MIC architecture features two modes of execution (native and offload), and additionally a

heterogeneous CPU/MIC MPI execution, we investigated different scenarios for building the code.

For the offload scenario, our initial idea was to locate the computational kernels (which are OpenMP enabled) and include the corresponding pragmas to achieve offloading of those kernels to the MIC coprocessor, while the main CPU runs the intrinsic tree-traversing algorithm. However, while we managed to compile the code in this configuration, we had no success in running the code, hitting seemingly random segmentation faults in any of the offloaded portions. We spent a lot of effort finding and fixing these, but the end result is not satisfactory. More work would be needed in order to finalize such configuration.

On the other hand, given the intrinsic tree-based structure of the algorithm, an *efficient* offloading scenario seems quite improbable. Without getting into too much technical details regarding the algorithm, its sufficient to note that it includes oct-tree bisection of the domain up to several (5, 6) levels, and executing 7 different computational kernels for each box. This can easily amount to a large number of kernel executions, each of them working on a small dataset. In a scenario where the kernels are offloaded, such behaviour will most probably lead to performance degradation due to the data preparation and communication overhead. For native compilation scenarios, we were able to compile and run the code and we have prepared build-time configuration directives that can be used to compile the code in another facility.

Verification of the port: Verification of the port was done by calculating the mean squared error (MSE) of a randomly chosen subset of 1,000 bodies between potentials obtained by running FMM calculations on 1,000,000 bodies and the potentials obtained for the test subset using direct pair-to-pair interaction. For 10-order multipole expansion we got MSE of the order $O(10^{-6})$, which is in accordance with the expectations. The same test was run on non-MIC architecture and showed similar MSE.

Scalability testing and suggested configurations: For scalability testing we used the following approach: we testing the code using a fixed number of particles and time steps on single MIC co-processor using different MPI-OpenMP configurations that result in maximum 240 threads on the MIC. We used scattering affinity for the threads, which proved to be more efficient than the compact one. The results are shown in Table 1. Maximum performance is achieved when running 16 MPI tasks each spawning 15 threads.

The table clearly shows that although the code performs quite fast, it does not scale well in terms of increasing both the number of ranks and the number of threads. In fact, there is negative scalability in some cases. For the threads we attribute this to the convoluted nature of the algorithm, which must fall back to task-based recursive parallelism because of the tree-like internal structure of the data. For the case of MPI ranks, it is yet unclear why the code cannot scale well beyond 16 ranks.

MPI Tasks	**Number of threads per MPI task**							
	1	**3**	**7**	**15**	**30**	**60**	**120**	**240**
1	7.00	17.12	15.65	18.80	15.43	16.52	5.13	4.06
2	3.50	6.70	6.66	6.58	7.02	3.99	2.76	
4	1.76	4.51	5.32	4.69	2.45	1.11		
8	0.89	2.31	2.51	1.21	0.74			
16	0.48	1.12	0.72	**0.32**				
32	0.35	0.55	0.35					
64	0.59	0.74						
128	1.58							

Table 19 Execution times in [s] for 10 steps over a system of 7,929,600 bodies with scattering thread affinity.

Conclusion:

The ScaFaCos/FMM code is best suited to work in MIC architecture in homogenous native Xeon Phi MPI mode. For O(10^7) particles, the code does not scale well beyond 32 MPI tasks. For a single MIC, the best option is to use 16 MPI tasks, each utilising 15 threads. Larger simulations would benefit from using more than 1 co-processor.

3.27 Development of AGBNP2 implicit solvent model library for Intel Xeon Phi architecture and its implementation in GROMACS

Project leader: Peicho Petkov
PRACE expert: Peicho Petkov
PRACE facility: EURORA
Other facilities (if any):
Research field: Life Sciences
Application code: AGBNP2library
Project Type: Internal
PA number (for Type C project only):

Project objectives:

A library, implementing the AGBNP2 [27][28] implicit solvation model, was developed. The model evaluates the Born radii and the atomic surface areas in a parameter-free and conformational-dependent manner, and decomposes the nonpolar hydration energy into a cavity term and an attractive dispersion energy term. The library is intended to be used in Molecular Dynamics (MD) packages for estimation of solvation free energies and studying of hydration effects.

Work done and results:

Based on the project objectives, the following major activities are identified:

Implementing AGBNP2 implicit solvation model: The library was written in C and parallelized with OpenMP. The main objective was to parallelize the code efficiently to run on Intel Xeon Phi in native mode.

Two different structures were used to store data - the atomic properties structure, which is used in parallel over the atom number, and a multiplet structure, that holds 2-, 3-and 4-body cross-section volume data in a threadwise manner.

The library needs as input coordinates, atom types, partial charges and Lennard Jones parameters. The derived constants for the model are calculated and neighbour lists are constructed in parallel over the atoms and the data is stored in an atomic properties structure.

Based on the idea suggested in [27] each thread calculates 2^{nd}-/3^{rd}- and 4^{th}- order cross section volumes and stores them in cache memory if certain geometric conditions are met and the volume is greater than a constant, defined by the model. Then, the self-volumes, the surfaces and the scaling parameters are calculated in parallel over the atoms and held in the atomic properties structure. The V'_{ij} values are stored in a B-tree for each thread and are used for an on-the-fly calculation of the atomic scaling factors wherever needed. Several other atomic properties are also calculated in parallel over the atoms.

After that the free energy terms are computed in our implementation we did not include the hydrogen bonding energy correction of the AGBNP2 model and its contribution to the forces. For calculation of the GB energy a cutoff scheme is used. The forces, acting on each atom, are calculated in parallel using the multiplets structure, whereas the cutoff scheme is applied for calculation of the GB contribution

Optimizing performance for Intel Xeon Phi: For an optimised OpenMP parallelization, private copies of the number of 2-, 3- and 4-body cross-section volume lists and the B-trees are used for every thread without data overlapping. The atomic parameters and properties, stored in the atomic properties structure, are shared between the threads. The main goal of the so implemented parallel algorithm is to loop over private mutliplets or to use parallel FOR loops over shared data.

In accordance with Intel Xeon Phi programming guidelines [29] all data was aligned to 64B. Where possible, loops were modified to enable vectorization. Small on-the-fly functions were inlined. The results are obtained with the O3 optimization level of the Intel compiler and using of Intel's fast memset/memcpy libirc library.

Scalability testing: The most time consuming functions are the creation of the multiplets lists and the calculation of their properties, and the calculation of the inverse Born radii.

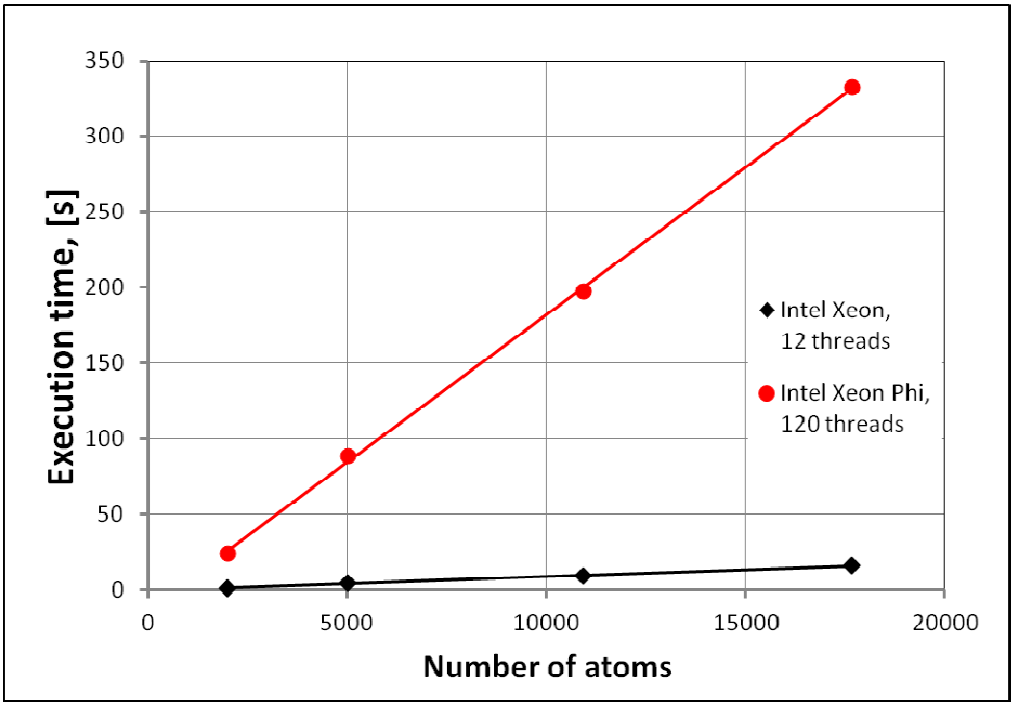


Figure 22 AGBNP2 scaling test results on Xeon Phi.

The implemented library contains about 2500 lines of code, organized in 76 functions and its optimization requires a lot of time and efforts. A major issue was that vectorization could not be enforced in loops with multiple calls to functions. Another problem includes complex manipulations of the indices within the loops body, which also deters vectorization. Although the code mainly contains element-wise array operations, we were not able to achieve any satisfactory results in terms of performance or speed-up on the Intel Xeon Phi. The library scales linearly with the size of the simulated system.

Conclusion

A library, implementing the AGBNP2 [27][28] implicit solvation model, was developed in C with OpenMP parallelization. The code was ported on Intel Xeon Phi in native mode. Unfortunately, the library proved to be quite complex and neither reasonable performance, nor scalability were obtained on the coprocessor. The available time of the project was not sufficient to properly address vectorization issues and other performance/scalability problems.

4 Summary

During the PRACE-1IP extension period, task T7.1 in WP7 focused on the application enabling on the Xeon Phi accelerators. The PRACE prototype EURORA at CINECA was used for most of the Xeon Phi enabling projects and some were allocated to the BSC's Xeon Phi system. A summer school was organised by CINECA on 8-11 July 2013 and attended with 34 researchers.

Two calls was opened / organised for the Xeon Phi enabling proposals, including the PRACE Preparatory Type C Call for EURORA (cut-off: 3 June 2013) which was followed by the PRACE-1IP Extension WP7 Internal Call (Closed on 15 August 2013). A total of 29 proposals were received and 27 Xeon Phi enabling projects were accepted supported during the PRACE-1IP extension period. T7.1 provided the technical reviews, including the home site reviews by CINECA and the Type C reviews, for all the received 29 proposals. Each accepted project was assigned to one or more PRACE experts who were responsible for the contact, the enabling support for the assigned project and the project's reporting. The projects' progress was monitored by monthly telcons. Discussion sessions in PRACE All Hands Meeting and WP7 F2F meeting were also organised for T7.1 during the PRACE-1IP extension period.

A wide range of applications were ported to the Xeon Phi prototypes and optimisations were implemented where possible. The difficulties of porting and optimising the code depend a lot on the code features. In general, compared with the porting on other accelerators, most of the codes were well ported to Xeon Phi with reasonable efforts. However, many codes required more efforts to gain the optimised performance. Some projects reported the initial optimisation results were promising and they are planning to continue future investigations after the PRACE-1IP extension period ends.

Besides the brief reporting from each Xeon Phi enabling project in this deliverable, most of the projects will also produce a white paper to include more technical details. The white papers of the enabling work on Xeon Phi will be reviewed and published in January 2014.

5 Annex

The white papers for the Xeon Phi enabling project are listed below. The white papers will be available for publication on the PRACE-RI web site [1] in January 2014

Title	Author(s)
Performance Analysis and Enabling of the RayBen Code for the Intel MIC	A. Schnurpfeil, F. Janetzko, St. Janetzko, K. Thust (FZJ), M. S. Emran, J. Schumacher (TU-Ilmenau)
Porting of Computational Mechanics Codes to MIC	Mariano Vazquez, Raul de la Cruz, Judit Gimenez, Vicenç Beltran, Felix Rubio Dalmau (BSC)
Enabling the UCD-SPH code on the Xeon Phi	Christian Lalanne (ICHEC), Ashkan Rafiee (University College Dublin), Denys Dutykh (University College Dublin), Michael Lysaght (ICHEC), Frederic Dias (University College Dublin)
Porting and Optimizing a MHD code for XeonPhi	Evghenii Gaburov (SURFsara), Yuri Cavecchi (University of Amsterdam)
Multi-Kepler GPU vs. Multi-Intel MIC for spin systems simulations	M. Bernaschi (CNR), M. Bisson (CNR), F. Salvadore (CINECA)
Enabling SMEAGOL on Xeon Phi: Lessons learned	Alin Elena (ICHEC) and Ivan Rungger (Trinity College Dublin)
SuperLU_MCDT (Many Core Distributed) Solver on MIC Architecture	Ahmet Duran, Serdar Celebi, Mehmet Tuncel and Bora Akaydin (ITU)
Code Optimization and Scaling of Astrophysics Software Gadget on Intel Xeon Phi	Plamenka Borovska, Desislava Ivanova (NCSA)
Code Optimization and Scaling of Massively Parallel Multiple Sequence Alignment Method Based on Artificial Bee Colony on Intel Xeon Phi	Plamenka Borovska, Veska Gancheva, Nikolay Landzhev (NCSA)
Code optimization and Scaling of Multiple Sequence Alignment Software ClustalW on Intel Xeon Phi	Plamenka Borovska, Veska Gancheva, Simeon Tsvetanov (NCSA)
Porting FEASTFLOW to the Intel Xeon Phi: Lessons learned	Ioannis Venetis, Georgios Goumas, Markus Geveler, Dirk Ribbrock(GRNET)
Optimising CP2K for Intel Xeon Phi	Fiona Reid, Iain Bethune (EPCC)
Porting a real-world seismological application to the Intel MIC architecture	M. Allalen, G. Brietzke, V. Weinberg (LRZ)
FMPS on MIC	Florian Seybold(HLRS), Ralf Schneider (HLRS), David Horak (VSB), Lubomir Riha (VSB), Vaclav Hapla(VSB), Vit Vondrak(VSB)
Massively parallel 3D Poisson equation solver for Intel Xeon - Xeon Phi HPC	Stoyan Markov, Peicho Petkov, Damyan Grancharov, Leandar Litov (NCSA)
Exploiting Locality in Sparse Matrix-Matrix Multiplication on the Many Integrated Core Architecture	Kadir Akbudak, Cevdet Aykanat (Bilkent)

Porting and verification of ExaFMM library to MIC Architecture	Dr. Valentin Pavlov, Nikola Andonov and Georgi Kremenliev(NCSA)
Analytical generalized Born plus nonpolar implicit solvent(AGBNP2) library for Intel Xeon – Xeon Phi HPC	Stoyan Markov, Peicho Petkov, Elena Likova, Damyan Grancharov, Nevena Ilieva, Leandar Litov(NCSA)