SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2007-2.2.2.1 - Preparatory phase for 'Computer and Data Treatment' research infrastructures in the 2006 ESFRI Roadmap

PRACE

Partnership for Advanced Computing in Europe

Grant Agreement Number: RI-211528

D6.3.1
Report on available Performance Analysis and Benchmark Tools, Representative Benchmark

Version:	1.0
Author(s):	Peter Michielse (NCF), Jon Hill (EPCC), Guillaume Houzeaux (BSC), Olli-Pekka Lehto (CSC), Walter Lioen (SARA)
Date:	24/11/2008

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="6">PRACE Project</td><td colspan="2">Project Ref. №: RI-211528</td></tr>
<tr><td colspan="2">Project Title: Partnership for Advanced Computing in Europe</td></tr>
<tr><td colspan="2">Project Web Site: http://www.prace-project.eu</td></tr>
<tr><td colspan="2">Deliverable ID: D6.3.1</td></tr>
<tr><td colspan="2">Deliverable Nature: Report, Initial Benchmark Suite</td></tr>
<tr><td>Deliverable Level:
PU</td><td>Contractual Date of Delivery:
30 / November / 2008

Actual Date of Delivery:
30 / November / 2008</td></tr>
<tr><td colspan="2">EC Project Officer: Maria Ramalho-Natario</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 available Performance Analysis and Benchmark Tools, Representative Benchmark</td></tr>
<tr><td colspan="2">ID: D6.3.1</td></tr>
<tr><td>Version: 1.0</td><td>Status: Final</td></tr>
<tr><td colspan="2">Available at: http://www.prace-project.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word 2003</td></tr>
<tr><td colspan="2">File(s): PRACE-D6.3.1-final-24112008</td></tr>
<tr><td>Authorship</td><td>Written by:</td><td>Peter Michielse (NCF), Jon Hill (EPCC), Guillaume Houzeaux (BSC), Olli-Pekka Lehto (CSC), Walter Lioen (SARA)</td></tr>
</table>

	Contributors:	Martin Polak (GUP), Albert Farres (BSC), Miquel Catala (BSC), Jesus Labarta (BSC), Judit Jimenez (BSC), Xavi Saez (BSC), Raul de la Cruz (BSC), Rogeli Grima (BSC), Mauricio Araya (BSC), Rosa Badia (BSC), Eduard Aiguadé (BSC), Jussi Enkovaara (CSC), Pekka Manninen (CSC), Sebastian von Alfthan (CSC), Jussi Heikonen (CSC), Harald Klimach (HLRS), Stefan Wesner (HLRS), Stefanie Meier (FZJ), Wolfgang Frings (FZJ), Lukas Arnold (FZJ), Florian Janetzko (FZJ), Alexander Schnurpfeil (FZJ), Orlando Rivera (LRZ), Matthis Brehm (LRZ), Iris Christadler (LRZ), Maciej Filocha (PSNC), Maciej Cytowski (PSNC), Maciej Szpindler (PSNC), Mark Cheeseman (CSCS), Tim Robinson (CSCS), Neil Stringfellow (CSCS), Jean-Guillaume Piccinali (CSCS), John Donners (SARA), Rob de Bruin (RUG), Arnold Meijster (RUG), Aad van der Steen (NCF), Andy Sunderland (STFC), Charles Moulinec (STFC), Ilian Todorov (STFC), Joachim Hein (EPCC), Xu Guo (EPCC), Mark Bull (EPCC), Alan Simpson (EPCC), Bertrand Cirou (CINES), Jean-Cristophe Trama (CEA), Pierre-Francois Lavallee (IDRIS), Nikos Tsakiris (GRNET), Giorgos Goumas (GRNET), Vegard Eide (SIGMA), Giovanni Erbacci (CINECA), Carlo Cavazzoni (CINECA), Ulf Andersson (SNIC), Aad van der Steen (NCF)
	Reviewed by:	Miroslaw Kupczyk, PSNC, Thomas Eickermann, FZJ
	Approved by:	Technical Board

Document Status Sheet

Version	Date	Status	Comments
0.1	05/September/2008	Draft	Outline version
0.2	26/September/2008	Draft	Chapter 1, 2, 3 details added
0.3	08/October/2008	Draft	Synthetic bmark and Integration bmark suite added
0.4	15/October/2008	Draft	Chapter 3 completed
0.5	24/October/2008	Draft	Complete document, for internal WP6 review
0.9	12/November/2008	Final Draft WP6	Integration of comments from WP6 review, made available to PRACE internal review
0.95	24/November/2008	Final Draft PMO	Integration of PRACE internal reviewers comments
1.0	24/November/2008	Final version	

Document Keywords and Abstract

Keywords:	PRACE, HPC, Research Infrastructure, Applications, Benchmark Suite, Synthetic Benchmarks, Prototypes, Performance Analysis Tools
Abstract:	This document reports on the construction of a benchmark suite, to be used both within the current PRACE project and beyond, when actual Tier-0 systems will be purchased. Apart from the benchmark suite, this document also reports on currently available performance analysis tools and synthetic benchmarks, as these are essential tools for monitoring scalability and optimisation of benchmark codes, and for analysing and comparing the basic components of HPC systems. This document takes its input from various sources. First, there is the list of applications and their requirements, as delivered by tasks 6.1 and 6.2. As these applications belong to the most frequently used on current European HPC platforms, they should form the basis of a PRACE benchmark suite. Secondly, there is the hardware architecture survey, as conducted by WP7 and its consequences for prototype systems to be used within PRACE. As these prototype architectures are considered as important, it makes sense to use these as platforms for benchmark preparations on scalability (to be handled by task 6.4) and optimisation (task 6.5). A third aspect is the available combinations of expertise on applications and expertise on architecture, for which it makes sense to be used as appropriate and efficient as possible. PRACE targets towards a European research Infrastructure, ideally consisting of various hardware architectures. This implicitly means that some applications are more suited to certain architectures than others. This needs to be reflected in the final benchmark suite, with the idea that potentially subsets of the overall benchmark suite may be used for benchmarking different architectures. These aspects together lead to the output as described in this document, which consists of an initial benchmark suite, with applications ported to target architectures, including recommendations on further work and effort estimates for petascaling (task 6.4) and optimisation (task 6.5). Integration of the benchmark codes into a benchmark suite is an important subtask, as it ensures that other tasks and workpackages within PRACE can use the benchmark suite as their starting point. Identification and categorisation of performance analysis tools and synthetic benchmarks is included as well, especially to be used when tasks 6.4 and 6.5 take off. The document and the initial benchmark suite form a strong basis for this future work.

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] PRACE: http://www.prace-project.eu
[2] TRAC: http://trac.edgewall.org/
[3] JuBE: http://www.fz-juelich.de/jsc/jube/
[4] DEISA: http://www.deisa.eu/science/benchmarking
[5] STREAMS: http://www.streambench.org/
[6] EuroBen: www.euroben.nl
[7] SPEC: www.spec.org
[8] PERFECT:
[9] ASC Sequoia: https://asc.llnl.gov/sequoia/benchmarks/
[10] LINPACK (HPL): http://www.netlib.org/benchmark/hpl/
[11] NAS Parallel Benchmarks: www.nas.nasa.gov/Resources/Software/npb.html
[12] STREAM2: http://www.cs.virginia.edu/stream/stream2/
[13] P-SNAP: http://www.c3.lanl.gov/pal/software/psnap/
[14] Selfish: http://www-unix.mcs.anl.gov/zeptoos/projects/
[15] EPPC OpenMP:
http://www2.epcc.ed.ac.uk/computing/research_activities/openmpbench/openmp_index.html
[16] Intel MPI: http://www.intel.com/cd/software/products/asmo-na/eng/219848.htm
[17] SPEC MPI: http://www.spec.org/mpi/
[18] SKaMPI: http://liinwww.ira.uka.de/~skampi/
[19] Sandia SMB: http://www.cs.sandia.gov/smb/
[20] HPCC: http://icl.cs.utk.edu/hpcc/
[21] PARKBENCH: http://www.netlib.org/parkbench/
[22] IOR: http://sourceforge.net/projects/ior-sio/
[23] IOZONE: http://www.iozone.org/
[24] Bonnie++: http://sourceforge.net/projects/bonnie/

[25] B_eff_IO: http://www.hlrs.de/organization/par/services/models/mpi/b_eff_io/
[26] Allinea: www.allinea.com
[27] CEPBA: www.bsc.es
[28] Cray Inc.: www.cray.com
[29] HPCT: http://domino.research.ibm.com/comm/research_teams.nsf/pages/actc.index.html
[30] IPM: http://www.nersc.gov/nusers/resources/software/tools/perf.php
[31] Scalasca: www.fz-juelich.de/jsc/scalasca
[32] VAMPIR: www.vampir.eu
[33] VPA: http://www.alphaworks.ibm.com/tech/vpa
[34] HPL Cell example: http://www.netlib.org/benchmark/hpl/

List of Acronyms and Abbreviations

ASC	Advanced Strategic Computing.
BCO	Benchmark Code Owner.
BLAS	Basic Linear Algebra Subroutines (basic library).
BSC	Barcelona Supercomputer Center (Spain).
BSCW	Basic Support for Cooperative Work, a collaborative workspace software package.
BT	Block Tridiagonal (NPB application).
B/s	Bytes per second.
CAF	Co-Array Fortran
CEPBA	European Center for Parallellisation at Barcelona
CFD	Computational Fluid Dynamics.
CPU	Central Processing Unit.
CSC	Center for Scientific Computing (Finland).
CSCS	Swiss National Supercomputing Centre (Switzerland).
CPI	Cycles per Instruction.
DEISA	Distributed European Infrastructure for Supercomputing Applications. EU project by leading national HPC centres.
EPCC	Edinburgh Parallel Computing Centre.
FFT	Fast Fourier Transform.
FT	Fast Fourier Transform (NPB application).
flop/s	floating-point operations per second.
Gflop/s	10^9 floating-point operations per second (Gigaflop/s).
GMRES	Generalised Minimal Residual Method.
GUI	Graphical User Interface.
GUP	Institute of Graphics and Parallel Processing, Johannes Kepler University Linz (Austria).
HDF	Hierarchical Data Format.
HLRS	High Performance Computing Center Stuttgart (Germany).

HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym of Supercomputing.
HPCC	High Performance Computing Challenge.
HPL	High Performance Linpack benchmark.
IO	Input-Output
JuBE	Juelich Benchmarking Environment.
LAPACK	Linear Algebra PACKage (numerical linear algebra library).
LU	Lower Upper diagonal factorization (NPB application).
MD	Molecular Dynamics.
Mflop/s	10^6 floating-point operations per second (Megaflop/s).
MHz	10^6 Hz.
MPI	Message Passing Interface. A library for message-passing programming.
MRNet	Multicast Reduction Network.
NAS	Numerical Aerodynamic Simulation.
NERSC	National Energy Research Scientific Computing Center.
NPB	NAS Parallel Benchmark.
OpenMP	Open Multi-Processing. An API for shared-memory parallel programming.
OS	Operating System.
PABS	PRACE Application Benchmark Suite.
PAPI	Parallel Application Programming Interface.
PAT	Performance Analysis Tool.
Perl	Practical Extraction and Reporting Language.
Pflop/s	10^{15} floating-point operations per second (Petaflop/s).
PGAS	Partitioned Global Address Space (classification of programming languages).
PRACE	Partnership for Advanced Computing in Europe; Project Acronym.
PSTSWM	Parallel Spectral Transform Shallow Water Model.
QCD	Quantum Chromo Dynamics.
SARA	SARA Computing and Networking Services Amsterdam (the Netherlands).
SPECfp	SPEC floating point benchmark.
SOAP	Simple Object Access Protocol
SP	Scalar Pentadiagonal (NPB application).
Tflop/s	10^{12} floating-point operations per second (Teraflop/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, while national or topical HPC centres would constitute tier-1.
Tier-1	Major national or topical HPC systems.
TPP	Theoretical Peak Performance.

UPC	Unified Parallel C
Wiki	Web page or collection of web pages for creating collaborative web sites.
XML	eXtensible Markup Langauage.

Executive Summary

In order to be a success, PRACE needs to understand the software requirements for future Pflop/s systems. This deliverable takes the key scientific and technical categories of applications, as conducted through a survey of most major European HPC systems and the applications that exploit these, carried out by task 6.1. It reports on the construction of a benchmark suite, to be used both within the current PRACE project and beyond, when actual Tier-0 systems will be purchased. Apart from the benchmark suite, this document reports on currently available performance analysis tools and synthetic benchmarks, as these are essential tools for monitoring scalability and optimisation of benchmark codes, and for analysing and comparing the basic components of HPC systems.

This document takes its input from various sources. First, there is the list of applications and their requirements, as delivered by tasks 6.1 and 6.2. As these applications belong to the most frequently used on current European HPC platforms, they should form the basis of a PRACE benchmark suite. Secondly, there is the hardware architecture survey, as conducted by WP7 and its consequences for prototype systems to be used within PRACE. As these prototype architectures are considered as important, it makes sense to use these as platforms for benchmark preparations on scalability (to be handled by task 6.4) and optimisation (task 6.5). A third aspect is the available combination of expertise on applications and expertise on architecture, for which it makes sense to be used as appropriate and efficient as possible.

PRACE targets towards a European Research Infrastructure, ideally consisting of various hardware architectures. This implicitly means that some applications are more suited to certain architectures than others. This needs to be reflected in the final benchmark suite, with the idea that potentially subsets of the overall benchmark suite may be used for benchmarking different architectures.

These aspects together lead to the output as described in this document, which consists of an initial benchmark suite, with applications ported (and to some extent analysed) to target architectures, including recommendations on further work and effort estimates for petascaling (task 6.4) and optimisation (task 6.5). Integration of the benchmark codes into a benchmark suite is an important subtask, as it ensures that other tasks and workpackages within PRACE can use the benchmark suite as their starting point. Identification and categorisation of performance analysis tools and synthetic benchmarks are included as well, to be used later in the project when tasks 5.4, 6.4 and 6.5 take off. We believe the document and the initial benchmark suite form a strong basis for this future work.

1 Introduction

The Partnership for Advanced Computing in Europe (PRACE, [1]) has the overall objective to prepare for the creation of a persistent pan-European HPC service. PRACE is divided into a number of inter-linked work packages, and WP6 focuses on the software for petascale systems.

The primary goal of PRACE WP6 is to identify and understand the software libraries, tools, benchmarks and skills required by users to ensure that their applications can use a Pflop/s system productively and efficiently. WP6 is the largest of the technical PRACE work packages and involves all of the PRACE partners.

Task 6.3 is responsible for the creation of a benchmark suite, to be used not only within WP6 but also in WP5, when testing and validating prototype systems. The benchmark suite should represent application areas from the potential user bases, and should take into account the available prototype architectures and available expertise within the PRACE project. This means that task 6.3 receives its input from tasks 6.1 and 6.2. With the application benchmark suite, task 5.4 will conduct its testing and analysing of the prototype systems, while tasks 6.4 and 6.5 will be able to cover aspects of scalability to Pflop/s systems and optimisation of applications. Since these efforts can not be done without suitable software tools and thorough understanding of the underlying hardware, task 6.3 covers performance analysis tools and synthetic benchmarking as well. The synthetic benchmark suite will be used by tasks 5.2 and 5.3.

Including the performance analysis tools and synthetic benchmark survey, the audience for this document is not only within other PRACE tasks, but hopefully also a wider HPC audience, as it offers characteristics and analysis for deployment of specific, heavily used applications codes on future Pflop/s systems.

1.1 Structure of the Report

This document is structured as follows. In section 2, besides some important definitions, a refinement of objectives and the methodology to arrive at them, will be discussed. This includes the practical approach of splitting the full task into a number of subtasks, but also the approach to use human resources as efficiently as possible. Section 3 covers the actual assignment of applications to people and prototype architectures, the obtained results with respect to porting, and the forecast with respect to scalability to Pflop/s systems and optimisation. Section 4 covers the actual integration of benchmark codes into a benchmark suite. Sections 5 and 6 discuss available synthetic benchmarks and performance analysis tools, respectively. With respect to conclusions and future work, we have taken the approach to include these aspects as a subsection in each of the sections 3 to 6, as this fits more natural to the actual subjects in the individual sections. Many details on porting of the applications to the prototype architectures can be found in the Annex.

2 Definitions, Objectives and Methodology

2.1 Definitions

The purchase process of large HPC computer systems (either focussed on capacity or capability) is generally supported by the execution of a set of user applications on the systems under consideration. In this respect, we are talking of benchmarking the systems, leading to technical information which allows proper technical comparison of the systems, especially with respect to their performance on real applications.

Technical information typically consists of actual performance details of the systems, which vary from the performance on component level to the performance of the system as a whole. This is reflected in the tests that are in use: tests on a component level (processor, caches, memory, interconnect, IO system) and obviously tests for the system as a whole. Tests on component levels are generally referred to as benchmark kernels or synthetic benchmarks, tests for the whole system as benchmark applications.

In order to avoid confusion, it is important to define clearly what is meant with the relevant terminology. For that reason, throughout this document, the following definitions will be used:

- A benchmark kernel is the collection of a small test program source code, run script, defined number of processors, possibly dataset and reference output, and is meant to test an individual component of the system;
- A benchmark code is the collection of one application source code, run script, defined number of processors, dataset and reference output, and is meant to test the behaviour of the system as a whole;
- A synthetic benchmark suite is the collection of benchmark kernels, to be run standalone;
- A benchmark suite is the collection of benchmark codes, together with the schedule to run the individual benchmark codes (either standalone or in some defined form of throughput). To distinguish this from the synthetic benchmark suite, we may refer to this as application benchmark suite;
- A performance analysis tool is a tool to use for getting performance information when running an application, in particular a benchmark kernel or benchmark code.

Throughout this document, the concepts of porting, petascaling and optimisation will be used frequently. It makes sense to describe these concepts here as well:

- Porting is the process of installation, compilation, linking and execution of an application source code on a specific hardware platform running specific software versions. Successfully ported (to distinguish from later optimisation and scaling efforts) means correct execution of the generated executable on the specific hardware platform running specific software versions, using representative input sets;
- Petascaling is the performance scalability of benchmark codes (including IO aspects) to petascale level architectures, and is typically expressed in the amount of cores which can still be efficiently used for the execuation of the benchmark code. This is most likely

depending on actual input sets. Using ca. 10 Gflop/s per processor core, this means systems with a number of processor cores in the order of 100,000;
- Optimisation is the improvement of typically single-CPU (or: single-core) (standalone) performance (including IO aspects) of a benchmark code, and is typically a combination of memory hierarchy management ("cache optimisation") and CPU floating-point unit scheduling. In this context, source code optimization is meant, rather than external factors such as job scheduling.

2.2 Objectives and Methodology

This section discusses the approach we have taken to arrive at the objectives for task 6.3, including the methodology with respect to the efficient usage of both human and hardware resources. The two main objectives for task 6.3 are:

- To create a benchmark suite which will serve as a starting point for tasks 5.4, 6.4 and 6.5 (this deliverable D6.3.1);
- To eventually create a benchmark suite which becomes the PRACE benchmark suite for Tier-0 procurement (D6.3.2 at the end of the PRACE project).

Further refinement of these objectives, in order to design a detailed work plan for task 6.3, has been done in the following way:

1. Definition of a representative set of benchmark codes, including representative datasets with respect to required size for petascaling;
2. Porting of benchmark codes to prototype hardware architectures, initial execution results, preparation for task 5.4 (benchmark evaluation on prototype systems), task 6.4 (petascaling) and task 6.5 (optimisation), identification of potential performance and scalability bottlenecks (including licensing);
3. Survey on integration of benchmark codes into a benchmark suite, and actual implementation as input for other WP/tasks in PRACE (in particular 5.4, 6.4 and 6.5)
4. Survey on available performance analysis tools (structured testing on benchmark codes will be covered in D6.3.2, which may result in collaboration with vendors of these tools);
5. Survey on synthetic benchmarks (not yet structured testing on platforms, nor adapting to cover petascale architectures, which is reserved for future work in 5.4 and 6.3). The results of the synthetic benchmark survey will be used by tasks 5.2 and 5.3.

Objectives 3, 4 and 5 are basically independent of the actual benchmark codes they should work with. They are also independent of each other, which means that for each of these individual work plans can be designed. This has actually been done, leading to three so-called subtask leaders, each responsible for reaching one of the objectives 3 to 5. Sections 4 to 6 will cover the results obtained for these objectives. For objective 2, the situation is more complex. First, objective 2 takes objective 1 as input. This process will be described in section 3.1. Secondly, the actual technical work for objective 2 will be continued after deliverable D6.3.1 and reported in D6.3.2, but also within tasks 6.4 and 6.5, which deal with petascaling and optimisation of the benchmark codes. For that reason, in agreement with tasks 6.4 and 6.5, we have chosen for a horizontal approach, which means the following:

- For the execution of tasks 6.3, 6.4 and 6.5, each benchmark code will be worked on under the responsibility of a so-called Benchmark Code Owner (BCO). The BCO is a person who in most cases belongs to the staff of one of the PRACE partners;
- The BCO will steer the actual (porting, petascaling and optimisation) efforts, such that the benchmark code will run on each of the designated prototype hardware architectures. This includes the scheduling of work among the contributing PRACE partners to the benchmark code;
- During the porting, optimisation and scaling process, the BCO communicates with the application owners on all aspects of the application: source code, dataset, output, etc. In particular, actual results will first be communicated to the application owner, and through the public status of the deliverable report also to hardware or software vendors, and the rest of the HPC community;
- Reporting of the actual benchmark results (porting, petascaling, optimisation) must be within PRACE and the EU, without reporting constraints set by the application owner. In case the application owner does set such constraints on this, we will not transfer the application into a benchmark code within the PRACE benchmark suite;
- In case licensing of the application is relevant, as a rule of thumb we have accepted licenses that allow free usage within PRACE as a minimum possibility.

Apart from a lot of technical effort to the benchmark codes and subtasks, quite some organisational effort has been needed on a central level to cover the definition and distribution of benchmark codes, to monitor progress and to obtain results. Each BCO has had a similar organisational task on its benchmark code level, just as each of the subtask leaders for their assigned subtask. This has led to a tree-structure of communication, with its root at the level of WP6 management. The whole PRACE project may benefit from this structure.

The concept of BCOs and contributors, the integration of individual benchmark codes into a benchmark suite and the future work within PRACE basically define a distributed working environment, in which various people contribute to shared entities. This means that, both within PRACE WP6 and later on also in other WPs, it is necessary to use software tools to support such a distributed working environment. For the moment, this has been done using the TRAC system, as used at CSC Finland. TRAC is a web-based software project management and bug/issue tracking system emphasizing ease of use and low ceremony. It provides an integrated Wiki, an interface to version control systems, and a number convenient ways to stay on top of events and changes within a project. For more details, we refer to [2].

3 Benchmark Selection

3.1 Relevance to PRACE

Considering the goals of PRACE (HPC ecosystem, various Tier-0 systems, sustainability, facilitating science), a benchmark suite to technically support these goals will need to consider the following aspects:

- coverage of relevant application areas;
- representative applications within the covered application areas;
- coverage of (the range of) hardware platforms (prototypes) which are relevant for PRACE;
- opportunities to test system components with synthetic benchmarks;
- petascaling opportunities of benchmark codes with relevant datasets;
- optimisation opportunities of benchmark codes.

These aspects have to be taken into account when designing both a synthetic benchmark suite as well as an application benchmark suite. This means that we will develop synthetic benchmark and application benchmark suites, which are both formal in approach and flexible in usage. After all, if the resulting HPC ecosystem is to support various Tier-0 architectures, we must be able to distinguish between these architectures with respect to the applications targeted for the Tier-0 systems, and hence in the ability to use subsets of the overall benchmark suites (synthetic and application).

3.2 Selection Approach

First, we have considered earlier work in WP6. Deliverable D6.1 has been working on the identification and categorisation of applications and initial benchmark suite. It has taken various angles to reach a list of so-called core applications, and a list of possible extensions. These are contained in tables 1 and 2. Note that the actual acronyms of the applications are explained in Annex 7.2.

It is clear that as many as possible applications of the core list should be integrated in a PRACE benchmark suite, as they represent a significant user community of the systems at PRACE partners.

Application name	Application area
QCD	Particle physics
VASP	Computational chemistry, condensed matter physics
NAMD	Computational chemistry, life sciences
CPMD	Computational chemistry, condensed matter physics
Code_Saturne	Computational fluid dynamics
GADGET	Astronomy and cosmology
TORB	Plasma physics
ECHAM5	Atmospheric modelling
NEMO	Ocean modelling

Table 1: The proposed list of core applications in D6.1.

Application name	Application area
AVBP	Computational fluid dynamics
CP2K	Computational chemistry, condensed matter physics
GROMACS	Computational chemistry
HELIUM	Computational physics
SMMP	Life sciences
TRIPOLI4	Computational engineering
PEPC	Plasma physics
RAMSES	Astronomy and cosmology
CACTUS	Astronomy and cosmology
N3D	Computational fluid dynamics

Table 2: Possible extensions to the core list of applications in D6.1.

A second consideration is the actual choice of prototype architectures, as being consolidated by the Technical Board as an advice to the Management Board (MB) of the PRACE Project, and prepared by WP2 and WP7. This has led to a MB decision on the following prototype architectures, as (near-) production systems, in table 3.

The third consideration to select the applications to be transferred into a benchmark code is a practical one. It basically consists of the combination of knowledge of the particular application, expertise with certain hardware platforms and access to prototype architectures. Typically, we have tried to identify PRACE partners which combine all three aspects as preferred BCO. For most applications, both from the core list as well from the extended list, this has been successful. Contributors to a benchmark code typically qualify if they satisfy at least one, and preferably two or even three of these aspects.

Architecture type	Actual system	Location
MPP-BG	IBM BlueGene/P	FZJ, Germany
MPP-Cray	Cray XT5	CSC, Finland
SMP-FatNode-pwr6	IBM p575 Power6	SARA, Netherlands
SMP-ThinNode-x86	Bull – Intel Xeon/Nehalem cluster	FZJ, Germany and CEA, France
SMP-ThinNode+Vector	NEC SX-9 + x86 …	HLRS, Germany
SMP-FatNode+Cell	IBM Power6 with Cell	BSC, Spain

Table 3: Actual prototype architectures in PRACE.

Taking this approach, we have mapped both the applications of the core list and the applications of the extended list to the set of prototype architectures, with BCOs and contributors. This has led to table 4:

Application	BCO	MPP-BG	MPP-Cray	SMP-TN-x86	SMP-FN-pwr6	SMP-FN+Cell	SMP-TN+vector	Contributors
QCD	FZJ	X	X		X			EPCC, CSC
VASP	BSC	X			X	X	X	NCF, GUP, HLRS, (PSNC)
NAMD	EPSRC	X	X		X	maybe		CSC, GRNET, GUP
CPMD	BSC	X			X	X	X	CSC, CINECA, SIGMA, HLRS, PSNC
Code_Saturne	EPSRC	X	X		X	X	maybe	HLRS, BSC
GADGET	LRZ	X		X	X			CINECA, DL, CSCS
TORB	BSC	X			X	X		
ECHAM5	CSCS	X	X	maybe	X		X	CSC, HLRS
NEMO	NCF	X	X		X		X	DL, CSCS, SIGMA
CP2K	CSC	X	X		X			EPCC, CINECA, CSCS, SIGMA
GROMACS	CSC	X	X		X			SNIC, NCF, CSCS
N3D	HLRS		X	X	X		X	
AVBP	GENCI	X		X	X			
HELIUM	EPSRC	X	X		X			
TRIPOLI_4	GENCI	X		X				
PEPC	FZJ	X	X		X			
GPAW	CSC	X	X		X			CINECA
ALYA	BSC					X		
SIESTA	PSNC					X		
BSIT	BSC					X		

Table 4: Application to benchmark translation, with BCO distribution.

Table 4 shows that all applications from the core list have come back as benchmark codes, on at least 3 target prototype architectures, completed with 3 applications from the non-core list: CP2K, GROMACS and N3D. These are the first 12 (green) rows of the table. SMMP, RAMSES and CACTUS have disappeared from the extended (yellow) list, as it turned out to be that there was no PRACE partner that could volunteer as BCO. Instead, GPAW (computational chemistry), ALYA (computational mechanics and fluid dynamics), SIESTA (computational chemistry, molecular dynamics) and BSIT (computational geophysics) have joined the application set, mainly to make sure that enough coverage of the SMP-FN+Cell platform could be guaranteed (yellow rows). An additional advantage of this is that two other application areas are introduced: computational mechanics and computational geophysics.

After finalisation of table 4, each BCO and its contributors have started the work on the benchmark codes and hardware architectures. The qualitative results of these efforts will be discussed in the next section.

3.3 Actual Results

Technically, task 6.3 is preparing benchmark codes for further research on petascaling and optimisation. Keeping this in mind, task 6.3 is covering porting of the agreed set of applications to the agreed set of prototype architectures, preparing the scene for future work. This section therefore does not contain high detail on scalability and optimisation, but does contain porting statistics and to some extent porting details. It also considers details with respect to petascaling and optimisation potential, including effort estimates.

Table 5 shows code characteristics with respect to programming languages, libraries, programming models and IO models.

Benchmark code	Languages	Libraries	Programming Model	IO characteristics
QCD	Fortran 90, C		MPI	no special
VASP	Fortran 90	BLACS, SCALAPACK	MPI (+ pthreads)	no special
NAMD	C++	Charm++, FFTW, TCL	Charm++, MPI, master-slave	no special
CPMD	Fortran 77	BLAS, LAPACK	MPI	
Code_Saturne	Fortran 77, C99, python	BLAS	MPI	read at start, write periodically
GADGET	C 89	FFTW, GSL, HDF5	MPI	
TORB	Fortran 90	PETSC, FFTW	MPI	read at start, write periodically
ECHAM5	Fortran 90	BLAS, LAPACK, NetCDF	MPI/OpenMP	read at start, write periodically
NEMO	Fortran 90	NetCDF	MPI	read at start, write periodically
CP2K	Fortran 95	FFTW, LAPCK, ACML	MPI	checkpoints and output, intense
GROMACS	C, assembler	FFTW, BLAS, LAPACK	MPI	read at start, write periodically, relaxed
N3D	Fortran 90	EAS3, Netlib (FFT)	MPI + NEC-microtasking	read at start, write periodically
AVBP	Fortran 90	Hdf5, szip, Metis	MPI	read at start, write periodically
HELIUM	Fortran 90		MPI	read at start, write periodically
TRIPOLI_4	C++		TCP/IP sockets	read at start, write periodically
PEPC	Fortran 90		MPI	read at start, write periodically
GPAW	Python, C	LAPACK, BLAS	MPI	read at start, write at end
ALYA	Fortran 90	Metis	MPI/OpenMP	read at start, write periodically
SIESTA	Fortran 90	Metis, BLAS, SCALAPACK	MPI	read at start, write periodically
BSIT	Fortran 95, C	Compression lib	MPI/OpenMP	read at start, write periodically

Table 5: Benchmark code characteristics.

Table 5 does not show big surprises. By far the most common programming language is some instance of Fortran, while MPI is by far the most popular programming model. Pure OpenMP (shared memory parallel) codes are absent in this list, four of the codes (counting NEC microtasking for OpenMP) implement a combination of MPI and OpenMP. IO characteristics typically are a result of the time-dependent nature of the underlying physical model, which gives rise to time stepping through the computational domain, resulting in periodic output. From a future perspective, an interesting conclusion that may be derived from table 5 is that scaling application performance to petascale systems will have to deal, one way or another, with both Fortran and MPI.

Figure 1 is the actual result of the porting activities. It shows the amount of assigned codes, as specified in table 4, and the amount of actually ported codes until now. This is typically work in progress, the most important reason being the fact that the full set of actual agreed prototype architectures is not yet available. We have been creative in using similar

architectures to cover as much porting work as possible, to be able to start the work already before the actual prototype systems became available. Another aspect we face here is the fact that people skills for working on SMP-FN+Cell in particular are scarce.

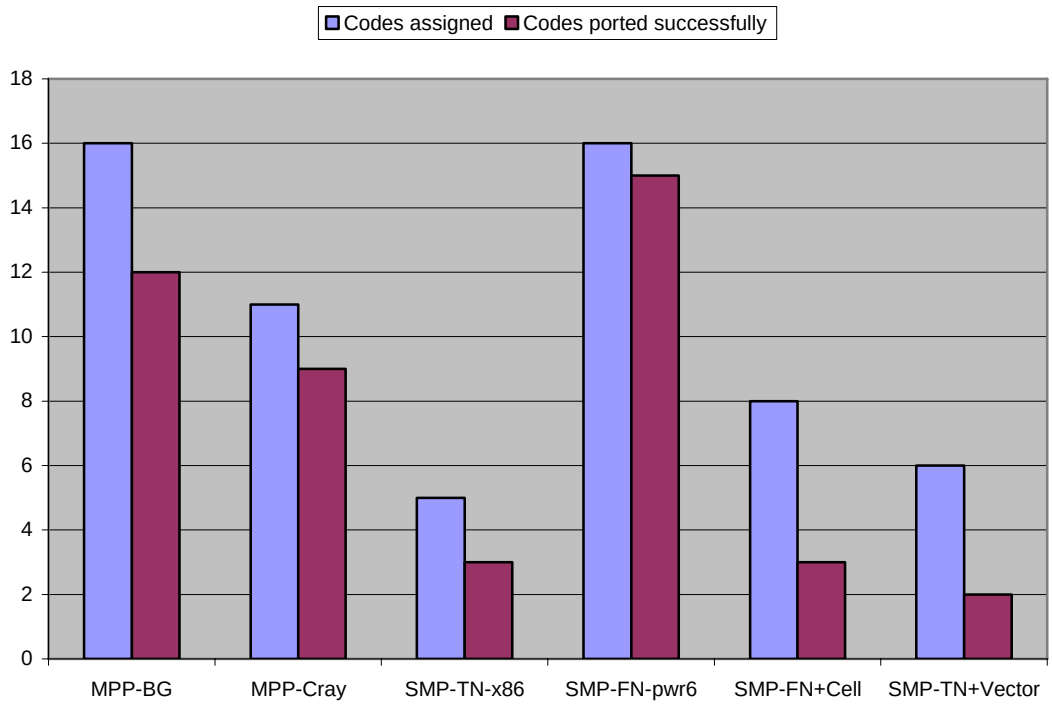

Figure 1: Overview hardware platform porting results.

Figure 1 does not give any details on which codes have been ported to which prototype architectures. Table 6 shows these porting details. In fact, table 6 is comparable to table 4 in the sense that it has the exact same sparsity pattern as table 4. Green colors denote successful porting, yellow means that porting is in progress, and orange means that porting has not started yet (and only in one case that porting has been stopped because of technical problems – ECHAM5 on MPP-BG).

Application	MPP-BG	MPP-Cray	SMP-TN-x86	SMP-FN-pwr6	SMP-FN+Cell	SMP-TN+vector
QCD	Done	In progress		Done		
VASP	Done			Done	Yet to start	Yet to start
NAMD	Done	Done		Done	Yet to start	
CPMD	Done			Done	In progress	Yet to start
Code_Saturne	Done	Done		Done	Yet to start	Done
GADGET	Done		Done	Done		
TORB	Done			Done	Yet to start	
ECHAM5	Stopped	Done	In progress	Done		Yet to start
NEMO	Done	Done		Done		In progress
CP2K	Done	Done		Done		
GROMACS	Done	Done		Done		
N3D		Yet to start	In progress	Yet to start		Done
AVBP	Yet to start		Done	Done		
HELIUM	In progress	Done		Done		
TRIPOLI_4	Yet to start		Done			
PEPC	Done	Done		Done		
GPAW	Done	Done		Done		
ALYA					Done	
SIESTA					Done	
BSIT					Done	

Table 6: Summary on porting efforts for benchmark codes and prototype architectures.

Apart from porting efforts to the prototype architectures, initial insights in the potential for scaling to petascale systems and single-CPU optimisation have been obtained. Table 7[1] contains the scalability potential of each of the benchmark codes, including an estimate on the amount of effort in person months (pm's). We have defined the range none-low-medium-high with respect to scalability as follows, and have assumed one core to deliver a minimum of 10 GFlop/s peak performance.

None (red): No speed-up above 2500 cores;
Low (orange): Speed-up obtained up to 5000 cores;
Medium (yellow): Speed-up obtained up to 10000 cores;
High (green): Speed-up obtained for more than 100000 cores.

Speed-up at a certain number of cores is defined as still getting execution time improvement when comparing the execution time on that number of cores to the execution time on half the number of cores.

From table 7, the following initial observations can be made:

- Within the set of computational chemistry codes (VASP, NAMD, CPMD, CP2K, GROMACS, GPAW) the potential varies from low to high. At first sight, this may seem surprising, as they all cover the same (broad) application area, although individual codes may use different approaches. It will make sense to investigate how low-scalable codes may benefit from algorithms and implementations used in high-scalable codes;
- The amount of effort estimated to improve scalability to medium or high seems to be reasonable: on average around 4 to 5 person months. This will be taken forward in task 6.4 for selected promising applications in close collaboration with the code owners. Further work on task 6.4 will take this forward.

Benchmark code	Expected scalability	Estimated effort	Comments and areas of attention
QCD	high	0-1 person months	
VASP	high		Depends on FFT and BLAS implementations
NAMD	medium-high	8-10 person months	Investigate master-slave (3 pm), investigate shared memory (7 pm)
CPMD	high	2 person months	Well parallelised already, some tuning needed
Code_Saturne	medium	3 person months	Preprocessing stage and IO
GADGET	medium-high	2 person months	Investigate potential OpenMP constructs and MPI implementation
TORB	high	3-5 person months	Adapt code internals (up to now 999 processes is max.)
ECHAM5	low-medium	2-8 person months	OpenMP optimisation, data output mechanism
NEMO	low	3 person months	Domain decomposition load imbalance, solver implementation, MPI
CP2K	low	5 person months	Load imbalance needs to be solved
GROMACS	medium	8 person months	Optimise communication patterns
N3D	low-medium	1-6 person months	Very platform dependent - MPI AlltoAll implementation
AVBP	medium-high	2 person months	Focus on MPI implementation (AllReduce area)
HELIUM	medium	3-4 person months	Focus on MPI implementation (synchronisation constructs)
TRIPOLI_4	high	6 person months	Independent particles, Monte-Carlo approach, IO to be modified
PEPC	high	1 person month	Data structure to be investigated
GPAW	medium-high	3-6 person months	Implement SCALAPACK usage, parallelise over electronic states
ALYA	medium-high	2 person months	Explicit solver ok, implicit solver requires effort, IO to be modified
SIESTA	medium	2-3 person months	Focus on MPI implementation
BSIT	high	1 person month	Embarassingly parallel, need to consider queue management system

Table 7: Expected scalability potential and estimated effort for benchmark codes.

A comparable exercise with respect to optimisation can be done. This leads to the results in table 8. This shows the effect that some codes have been optimised for single-CPU performance already quite extensively. Also, BCOs have focussed sofar on scalability potential rather than on optimisation potential. It is important to consider both effects, as

[1] Not all cells in tables 7 and 8 have been filled yet, as initial analysis after porting is currently work in progress.

dramatic single-CPU optimisation has in general far-reaching consequences for scalability expectations.

Benchmark code	Expected optimisation	Estimated effort	Comments and areas of attention
QCD	low	0-1 person months	Collection of five kernels, well-defined
VASP			
NAMD	low		Investigate simultaneous multi-threading
CPMD			
Code_Saturne	medium	3 person months	Include library routines, cache optimisation, vector constructs
GADGET	medium	3-6 person months	Modify algorithm towards floating-point work
TORB	medium		Vectorisation of loops for Cell
ECHAM5	high	1 person month	Investigate work done by hardware vendors
NEMO	medium	1-2 person months	Flat profile, cache optimisation
CP2K	medium	2-3 person months	Mostly cache optimisation
GROMACS	low-medium	1 person month	Mostly done through assembler routines already
N3D	low-medium	1-4 person months	Vectorisation done, include library routines, improve cache behaviour
AVBP	low	1-2 person months	Mostly done already
HELIUM	medium	2-4 person months	Few routines high in the profile, cache behaviour of long loops
TRIPOLI_4			
PEPC	medium	0-1 person months	
GPAW	low-medium	2-4 person months	Already relying on BLAS, further cache utilisation improvement
ALYA	medium	4 person months	Prepare solvers for vectorisation on Cell
SIESTA	high	2 person months	Optimisation for Cell
BSIT	medium	2 person months	Explicit finite difference method optimised, fine-tuning possible

Table 8: Expected optimisation potential and estimated effort for benchmark codes.

In summary, the results presented here are only initial results, and should serve as a good starting point for many other tasks within the PRACE project. Since a massive amount of information on porting these applications to the prototype architectures is available, and which is very useful, we have included this information in Annex 7.2.

3.4 Conclusions and Future Work

As has been mentioned before, porting the applications to the target prototype architectures is work-in-progress. The BCOs and their contributors have been able to already port a significant part of the applications to many of the assigned prototype architectures, where each application has been ported to at least one platform. This means that already a significant part of the sparse matrix has been filled. This work will continue to complete the sparse matrix on applications and prototype architectures.

Another aspect is the fact that already ported applications will enter the stadium of petascaling and optimisation, by tasks 6.4 and 6.5. Here we expect the concept of BCO to be very valuable, as work easily flows from task 6.3 into 6.4 and 6.5, and later backwards when integration of the scaled and optimised applications into the final benchmark suite for PRACE will need to be done.

With respect to the future final benchmark suite for PRACE, there is the issue of usage and licensing. It is planned that the PRACE benchmark suite will be used after the PRACE project, when real Tier-0 systems will need to be benchmarked. Within task 6.3, this may be an activity which can become quite important to make sure that the actual heavily-used applications will remain part of the PRACE benchmark suite.

4 Integration in Benchmark Suite

4.1 Introduction

In order to facilitate usage within other tasks in WP6 and within other work packages within PRACE, the benchmark codes from WP6 need to be integrated into a suite as part of task 6.3. Therefore, all applications that are part of tasks 6.3, 6.4 and 6.5, along with a series of synthetic benchmarks, are to form part of such an integrated benchmarking suite (PRACE Application Benchmark Suite – PABS) for use by the PRACE partners, in particular tasks 5.2, 5.3 and 5.4. This chapter describes how this integration will be implemented. A specific example is included in Annex 7.3.

4.2 Integration Framework

The technology used will be JuBE (figure 2 and [3]) from Jülich Supercomputing Centre (FZJ), which is based on Perl and XML. JuBE creates a definition for each platform, application and result set and allows automated compilation, running, and comparison of results against expected standards and reporting of results, as required the definitions in section 2.1 of this document. JuBE is also used by DEISA and it is anticipated that PRACE can use parts of the benchmark suite already created for overlapping applications and platforms. This enables PRACE taking advantage of experience of effort across the two projects.

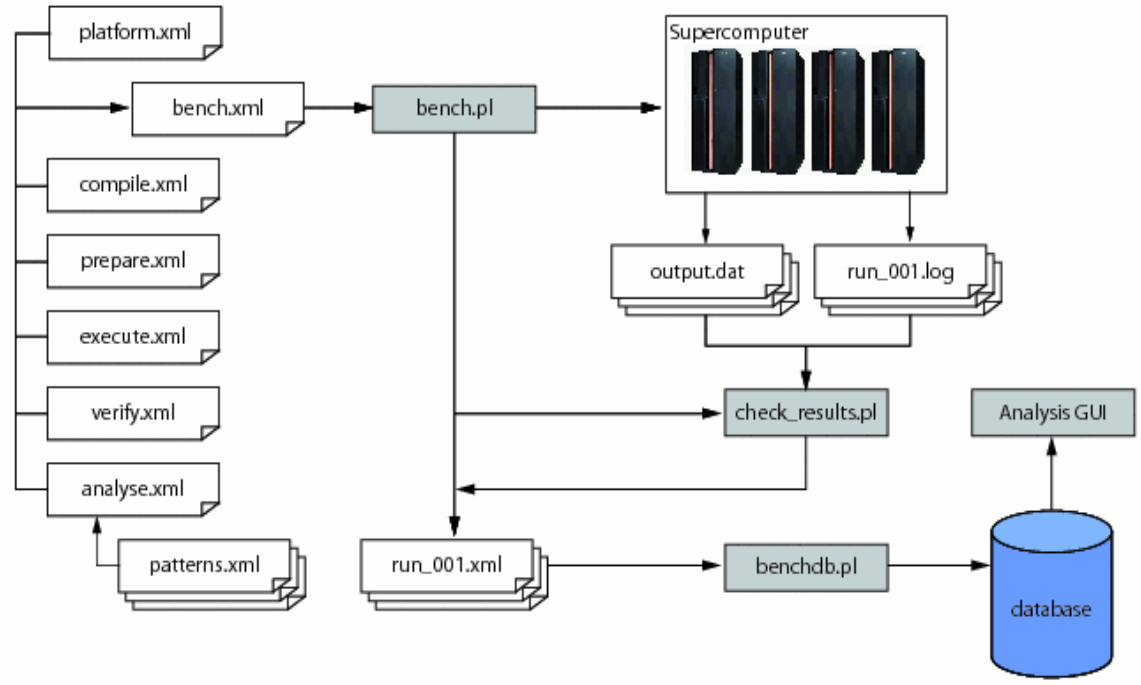

Figure 2: Overview of the JuBE framework.

The integration of the proposed benchmark codes requires that the BCO creates the definition files for each platform that the application is ported to. These definition files, along with the JuBE suite, are stored in the TRAC Subversion system [2]. Within the JuBE system for

PRACE there is a file documenting how to obtain the benchmark code and the datasets required.

JuBE requires that a pre-defined folder layout is followed, as shown in figure 3. Each application in the benchmark suite is stored in a folder with the correct name in the "applications" folder. Inside this folder are the "src" folder which contains the source code (obtained according to the instructions in the "HowToGet.txt" file) for the application and the "run" folder which contains any scripts needed to complete the benchmark for that application. The other folder of note is the "platform" folder, which contains the platform.xml file which defines the software for each platform and a folder for each platform which contains necessary skeleton files (such as job submission scripts) for each platform.

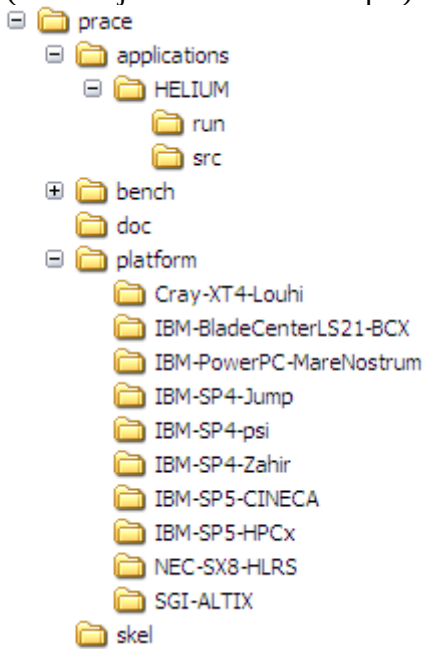

Figure 3: Folder layout for PABS.

In the example in figure 3, the actual application is "Helium", while a set of hardware platforms is used. The hardware platforms will need to include the PRACE prototype architectures.

As mentioned, there are a number of XML files that need to be created for each application in the benchmark code. For the actual hardware platform description, there is:

- Platform.xml – contains details particular to each platform, such as compilers, library locations and template job submission scripts. This is kept in the subdirectories of the "platform" folder (see figure 3).

For the benchmark code (application), there are a number of relevant files:

- bench.xml – this is the overarching XML file which details the benchmarks for each application code. There is one such file for each platform the application can be executed on;
- compile.xml – tells JuBE how to compile the application, detailing flags, compilers, etc.;
- prepare.xml – details how to set up a particular benchmark, such as altering an input file;
- execute.xml – details how to execute the application (or submit the job to the batch system);

- verify.xml – details how to verify that a particular benchmark has been successfully executed. The verification is carried out by a Perl script (check_results.pl);
- analyse.xml – details how to extract meaningful data, such as wall clock time, from the application output and builds a results table from this data. This makes use of the patterns.xml file.

All the above are kept in the relevant applications folder (Helium in the example in figure 3).

For the benchmark output, there is:

- patterns.xml – details the patterns to search for in order to analyse the output for the required data. A different file can be sued for each benchmark. This is also kept in the relevant application folder, but the "skel" folder also contains some useful files for this (e.g. analysing HPM data).

Actual execution of a benchmark code on a particular prototype architecture might comprise running the application over a number of processors. For example, NAMD, one of the core applications, runs a 2 million atom benchmark over 32, 64 and 128 processors. Running the benchmark will automatically compile the source code, submit the jobs to the scheduling system of the prototype, and once completed, will process and verify the output and produce a summary of the information. In the case of NAMD, the important summary statistic is the time taken per step, which JuBE can search for in the output and present the figures in a table, as shown in figure 4:

```
namd_IBM-SP6-Huygens_apoa1_i000006
=======================================
  Subid                  ncpus      nodes taskspeode threadsask timepermin timepermax timeperavg timepercnt   walltime
  ------------------ ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
    n1p32t1_t001_i01         32          1         32          1       1.33       1.35       1.34          3    1321.93
    n2p32t1_t001_i01         64          2         32          1       0.62       0.63       0.62          3     676.05
    n4p32t1_t001_i01        128          4         32          1       0.32       0.32       0.32          3     428.80
```

Figure 4: Screen-grab of JuBE output for NAMD benchmark.

The running of such a benchmark is done with a simple command from within the application directory:

```
perl ../../bench/jube platform.xml
```

platform.xml should be replaced with the correct bench.xml for the particular platform in question. This command compiles the code, copies the executable to the right place and submits the jobs to the backend of the system.

Once the jobs have completed, the follwing command will analyse the data and present the results:

```
perl ../../bench/jube –update –result ID
```

again from the application directory. The ID is the ID number of the run (given in the output of the first command).

4.3 Conclusions and Future Work

The JuBE framework presents a consistent and useful way of packaging the diverse application and synthetic benchmarks being used by PRACE. Although creating the infrastructure is perhaps more effort than simply creating a script for each code to compile the application, it will allow easy use of the benchmark by other PRACE work packages. In addition, because of the overlap with DEISA, we reduce the potential of duplicated work over two European projects.

The initial task of integrating the benchmark suite is to add all synthetic and application benchmarks as defined in this document and required for other PRACE work packages. This work forms part of this deliverable. Future work will include adding more benchmark tests for each application in order to benchmark petascale systems as well as adding applications on the prototypes that come into service after the completion of this deliverable. With respect to JuBE, database integration is under development.

5 Synthetic Benchmarks

5.1 Introduction

This chapter gives an overview of synthetic benchmarks, their rationale, and their applicability. In addition, we evaluate their merits and shortcomings in order to arrive at a well-designed set of synthetic benchmark programs (synthetic benchmark suite) that comprehensively measure key performance metrics. Ultimately, this should lead to the understanding of the strong and weak points of the computers thus assessed and from there a notion of what may be expected from the performance of applications executed on these machines.

5.2 Synthetic Benchmarks: Overview

Computer benchmarks are designed to provide insight in the performance of a computer with respect to a code (or: program) that is executed on it. The procedure of benchmarking is very simple: one executes the code and measures the wallclock time that is spent in the execution. Unfortunately, in this way we only learn about the performance of a particular code run on a particular computer. It tells us nothing about the performance of other codes on the same computer or the same code on another computer. Worse, when code *A* is faster on computer *I* than on computer *II* this is no guarantee that the same will be true for code *B*. It may depend on the type, amount, and variety of operations that codes *A* and *B* contain and of the architectures of computers *I* and *II*, respectively. In the following we will explain how to address this problem, what the possibilities are to gain consistent knowledge about the performance of computer systems, and what are the constraints of that knowledge.

5.2.1 *Synthetic benchmarks: the why*

A so-called synthetic benchmark consists of one or more programs that do not represent a real application but rather attempts to assess a particular property of a computer system in order to understand why it performs as it performs. Such a property may be, for instance, the speed of a combined floating-point multiply-add operation or the bandwidth from the main memory to a CPU core. In general, a synthetic benchmark must be simple enough to relate the content of the benchmark code directly to some hardware characteristic of the processor or, in case of parallel programs, the set of processors involved and the network that connects them. Because of the complicated nature of computer systems it is not possible, or at least highly impractical, to extract all the desired knowledge about the system from one program that covers all of its performance aspects. The consequence is that one strives to put together a coherent set of programs that each address an aspect of the behaviour of a computer system. The complete set of programs together should provide insight about the strengths and weaknesses of a specific system. This makes it easier to compare it with other computers on these same points. The knowledge thus obtained will make it possible to make some predictions about the performance that we might expect with full applications beforehand. This, in turn, can have the effect that we can decide to include or exclude a computer system in the set of systems we want to consider in procurements and/or it may simplify the other

tests to be conducted on such candidate systems. So, to summarise the main reasons why synthetic benchmarks exist:

1. They enable us to discover the strong and weak points of a computer system (and sometimes even why this is so).
2. They enable us to compare certain performance characteristics of computer systems and possibly their relative suitability for the tasks we want them to perform.
3. They enable us to decide whether certain systems are of interest to us even without extensive testing with full applications.
4. They enable us to discriminate between relevant and irrelevant properties of the systems we want to consider for the tasks we want them to perform.

In all, synthetic benchmarking, when properly conducted, will provide us with general knowledge about computer systems and their (im)possibilities and it can give us a better founded decision with regard to systems we plan to select. Also, synthetic benchmarks can considerably reduce the time and money spent in computer procurements both on the side of the institution that wants to buy such a system as well as on the side of the candidate vendors. This aspect should not be underestimated as, especially for the smaller computer vendors, a complete procurement procedure including a large benchmark can be a huge investment that might discourage them to get involved, however good their product might be.

5.2.2 *Synthetic benchmarks: the what*

Having given a rationale for the existence of synthetic benchmarks, we need to define how to design the programs that measure the desired characteristics and what we expect to learn from them. Because we restrict ourselves to HPC in the technical/scientific arena we disregard the many benchmarks that have been designed to measure aspects like graphics performance or database queries. Rather we concentrate on those that provide information about the speed of basic operations and fundamental algorithms that are the computational basis for the applications that ultimately will be executed on the target systems: large HPC systems with a Theoretical Peak Performance around one Pflop/s or beyond. The Theoretical Peak Performance (TPP) is the upper bound of the performance of a system measured by the number of floating-point operations per second that can be attained by it. For a computer *X* it can be expressed as:

$$TPP_X = F \times N_{fp} \times N_c \times P$$

where F is the clock frequency of the processor core, N_{fp} is the number of floating-point results/clock cycle that a core can deliver, N_c is the number of cores per processor, and P is the number of processors in the system. It is also known as the performance number the vendor will guarantee never to exceed. In practice the TPP of a system will never be attained. The main reasons for this are that the number of floating-point results/cycle will normally be considerably less than what is theoretically possible and, second, in systems where we have many processors, i.e, the type of systems we are concerned about here, there will generally be an appreciable communication overhead that degrades the overall performance.

Synthetic benchmarks are directed at exposing the difference between the TPP of a system and the actual observed performance and especially at what causes this difference. The ratio between the TPP and the observed performance is called the *efficiency* of the system. Unfortunately, the efficiency is not constant for a given system but varies with the type of

computation being done. A good synthetic benchmark should therefore also be designed such that they show these differences in efficiency as clearly as possible. A direct consequence of this behaviour of computer systems is that there is not *one* performance number that totally characterises them but rather a *spectrum* of performance data resulting from the different computational tests in a benchmark. Another consequence is that ranking of the systems according to the benchmark results is never straightforward and sometimes not even possible.

So, what should be in a synthetic benchmark to make it at all useful? First, we should try to design programs that make us understand why certain computations have the efficiency that we observe. Second, we can identify basic operations and algorithms that are the main constituents of full applications and therefore have a predictive value with regard to the applications they are part of. When a certain algorithm performs exceedingly well and it represents most of the time in the execution of an application it is evident that the application itself will do well on that system. A well-known example is the set of applications that are dominated by the solution of dense linear systems. The solution of a dense linear system is, in turn, dominated by the matrix-matrix multiplication algorithm. On almost all systems this algorithm can be performed with high efficiency, often close to 90% or even higher. It is therefore evident that all such applications will perform well on systems that have a high efficiency for the matrix-matrix multiplication. A good synthetic benchmark will try to cover the space of fundamental operations and algorithms that appear in all important application classes and thus may help in explaining, at least in part, what may be expected of the performance for these application classes.

The basic loss of efficiency at the processor level stems from the huge mismatch between the speed of the functional units in the processor core and the speed of the memory. The memory is not capable of providing enough operands to the functional units to keep them constantly busy and therefore they are in many cases waiting for operands instead of producing results. To reduce the effect of the slow memory many well-known devices have been added, like various levels of caches and multi-threading capabilities that hide this so-called memory latency. This is the reason why it is not sufficient to measure the speed of an operation only for one fixed data length: adding the elements of two arrays of length N = 6,000 will result in a totally different speed than when the arrays have a length of 10,000,000 as can be seen in the following example in table 9:

$c_i = a_i + b_i$ $i = 1,...,N$	N = 6,000 (Mflop/s)	N = 10,000,000 (Mflop/s)
IBM POWER6	767.26	204.76
SiCortex 5832	201.21	96.126

Table 9: Influence of problem size for two processor cores.

For N = 6,000 all operands come from the cache, for N = 10,000,000 the operands must all be fetched from memory. Note that the performance penalty for the POWER6 processor is more than a factor 3.5, while it is just over a factor of 2 for the SiCortex (MIPS) processor. This is mainly due to the much higher clock frequency of the POWER6 processor: 4.7 GHz against 500 MHz for the SiCortex with a corresponding mismatch between the speed of the memory and the processors. It is therefore evident that one should measure a range of lengths for such operations when one wants to assess what happens within an application. A single value will not do.

The influence of the relative speed of the memory and the various levels of cache memory can be measured in very much detail. A program that is only concerned with this metric is CPU-Z. It gives detailed bandwidth results for all memory levels as well as latencies, i.e., the time needed by the cache or memory to react to a memory request at all. A program that yields less detail but gives bandwidth results for various modes of reading from/writing to the caches is cachebench. The problem with both programs, however, is that the outcomes are very hard to relate to actual operations or algorithms. It is therefore preferable to use actually occurring operations and to measure the bandwidth for these operations directly. This is for instance done in the STREAMS benchmark [5] for a limited number of cases and in the EuroBen benchmark [6] for a larger set of operations. From these measurements one can infer in how far speed of the operations is limited by the bandwidth from the cache or memory.

On the other hand, the benchmark programs must not be too complicated to trace the performance back to specific system components. Synthetic benchmarks that are problematic in this respect are the SPEC benchmarks [7] and PERFECT benchmark [8]. The latter is comprised of so-called “compact applications”, i.e., applications that are reduced to a minimum in terms of input sets and output sets. It is indicative of the problems of this benchmark that it has never properly been decided in the run rules when a program had normally terminated. For instance, in a proposal for the run rules a program was regarded to have (successfully) terminated execution when the output set was written to disk. The objection made at the time was that in large systems it might well be possible that the output set would reside in the disk cache and not on disk and it could depend on the size of the output set whether one would regard the program as finished or not. The main problem, however, with both the PERFECT and the SPEC benchmarks is that it is virtually impossible to understand *why* a particular component program performs as it does. In this respect we are in no better position than when one would measure the performance of ones own application(s): it does not add to our insight but yields performance numbers of which we cannot say how they will predict the behaviour of a system in other situations. Worse, in case of the SPEC benchmarks, often not even the individual outcomes are used, but the geometric mean of all programs in the set. This effectively erases all characteristic behaviour that might show up in the constituent programs and give us information about the (special) features of the system under consideration and make it stand out positively or negatively.

There has always been a strong tendency to reduce the performance of a computer system to a single number. This is very understandable both from the side of the users as well as from the vendor's side: the users just have to rank the candidate systems according to this metric, while such a number can be an excellent selling argument for a vendor when it is better than that of his competitors. It will be clear by now that this type of ranking scheme does not acknowledge the multi-faceted nature of HPC systems. Despite this widely accepted fact, these types of single-metric rankings remain popular. Motivations for such a choice are that other benchmarks are more expensive and/or more complicated and are of limited relevance. The risk for ending up with a sub-optimal system, especially with HPC systems, is however very large and should be avoided. On the other hand one has to prove that a well-designed synthetic benchmark will have sufficient advantages over the simplistic approach to encourage the relevant parties to adopt it. This means that the synthetic benchmark should have, whenever possible, the following properties:

1. The programs in the benchmark should not be too low level.
2. The programs in the benchmark should not be too complicated.
3. The programs should expose the properties of the machine in a context that is relevant for applications on HPC systems.

4. The programs should cover the space of basic operations and algorithms as used in scientific/technical applications on HPC systems.
5. The installation and execution of the benchmark should be simple and system independent., i.e., portable.
6. The runtime of the total benchmark set should not be excessive (many hours or days).
7. The run rules and the benchmarking circumstances should be unequivocal.

Property 7 includes the possibility to check in a simple way whether a program has executed correctly. With the aforementioned properties it is possible to define what, at minimum, should be incorporated in the set of benchmark programs to satisfy our needs.

5.3 An Outline for a Synthetic Benchmark

We are now in a position to be more concrete about what should appear in a synthetic benchmark set and how it might be organised. First, we need to identify what basic algorithms are used in what applications. Furthermore, we must include basic operations, both with respect to computation and communication to get a hold of the upper/lower bounds on the speed of these operations. This, in turn, will give us the opportunity to assess whether the operations that make up the basic algorithms indeed perform as expected, thus providing us with a consistency check. It should be noted here that we look at the hardware and the compiler (or library) as a whole: it might well be that another compiler on the same systems would perform better or worse, dependent on the quality of the code generation. This is another reason to check both on the basic operations and basic algorithms built from them. So, let us make an attempt to define what concretely should at least appear in the synthetic benchmark set and why.

Basic operations:	• Constant copy: $a(i)_{i=1,\ldots,n} = c$, • Array copy: $a(i)_{i=1,\ldots,n} = b(i)_{i=1,\ldots,n}$, • Dyadic operations, like: $c_{i=1,\ldots,n} = a_{i=1,\ldots,n} + b_{i=1,\ldots,n}$, • Dot product • Vector update • 2-D rotation • 2nd difference operation • Low-order recursion • High-order polynomial evaluation
Basic algorithms:	• Matrix-vector multiplication (dense and sparse) • Solution of a linear system (dense and sparse, the sparse systems in both a regular and an irregular form) • Evaluation of eigenvalue problems (dense and sparse) • MD update kernel • Fast Fourier Transforms • Random Number Generation • Sorting.

Basic I/O patterns:	• Data transfer performance
	  o Variable block sizes
	  o Access patterns: sequential, strided, random
	  o Concurrency: shared, individual files
	• Metadata performance
	  o File creation & deletion

Basic communication patterns:	• Communication latency and bandwidth
	  o Point-to-point communication: synchronous, asynchronous, one-sided
	  o Collective communication: Broadcast, reduce, alltoall etc.
	  o Communication patterns: Halo, ring, bisection etc.
	• Overlap potential for communication and computation

Table 10: Desirable components represented in a synthetic benchmark set.

The number of basic operations and algorithms listed in table 10 is fairly large and one should realise that for a particular problem type more than one implementation might be needed. For instance, for regular, symmetric sparse linear systems stemming from a Finite Difference scheme a Conjugate Gradient algorithm can be used, however, for a irregular, non-symmetric system as occurs in Finite Element problems a Krylov method, like GMRES is required. These methods behave quite differently and should consequently both be present. Multigrid solvers again are very different from the two methods mentioned before and well might be included also. On the other hand, one should take care not to try to include each and every algorithm ever used in the complete HPC application range but rather concentrate on those that usually take the largest fraction of time in the applications one is interested in.

Where possible, the programs that implement them should preferably be self-checking in order to see immediately whether they have run correctly or not. Furthermore, the programs should be run for a range of problem sizes to assess the influence of the caches for (semi-) numerical operations and algorithms, of buffer size and block size in I/O programs, and of different communication protocols in communication programs.

To be a fair reflection of what is contained in application codes the synthetic programs should implement the basic algorithms according to best practices and, when new types of algorithms emerge, these should replace or extend the programs already in the benchmark. In other words, the benchmark set has to evolve according to new ways of problem solving in the HPC application field. In addition, when new languages like UPC, CAF, or other PGAS languages become main vehicles for implementing applications, one should also include relevant kernels coded in these languages.

The use of best practices brings some problems with it: when applications are optimised for a certain HPC platform, it is quite probable that numerical library routines will be used because they can often be considerably faster than the equivalent code written in one of the standard high level languages, like Fortran or C(++). The programs in the benchmark therefore must be able to accommodate the use of such library routines with the consequence that slight variations of some standard benchmark codes may be required. For instance, programs implementing dense linear algebra algorithms will hardly be affected, or not at all, because the calling sequences of the relevant routines (BLAS, LAPACK) are accepted to such a degree that they are present for virtually any platform. Unfortunately, this is not true for FFTs and Random Number Generation for which there are no generally accepted routine interfaces that are identical across all existing HPC platforms.

Having stated the desirable properties of a good synthetic benchmark set, we are now in a position to review a number of synthetic benchmarks that exist at this moment and see whether they meet our requirements.

5.4 Review of some Synthetic Benchmarks

Presently, many synthetic benchmarks are around. Not all of these try to address all the areas that are of interest: computation, communication, and I/O. In fact, for this last area there are hardly any useful and/or well accepted synthetic benchmarks and this should be a main concern in this project. For communication on the other hand there are a few benchmarks around that show a considerable overlap. We will discuss some benchmarks below in order to assess whether they can be used as a starting point or a constituent for a joint synthetic benchmark set.

5.4.1 *Computationally oriented benchmarks*

The benchmarks in this subsection concentrate on the computational aspects. This is not to say that no communication is performed, but rather that it is not regarded as the topic to be benchmarked. In this context it is just a vehicle to enable the correct execution of the benchmark codes.

- **ASC Sequoia benchmarks** [9] A very disparate set of benchmark codes. Mostly full, large applications. It contains codes in Fortran, C/C++, Python and a variety on communication mechanisms including OpenMP, MPI, hybrid OpenMP/MPI, and Pthreads. Not all codes seem to be reliably or optimally implemented. The majority of programs do not suit our purposes because of their complexity.
- **LINPACK (HPL)** [10] Solves a dense linear equation of an arbitrary order. It is useful to obtain an upperbound on the performance of a system but not for anything else. In that sense it is very similar to the TPP of the system and the correlation between both values is generally over 95%.
- **NAS Parallel benchmarks (NPB)** [11] These benchmarks are primarily computationally oriented although they address parallel computers. NAS stands for Numerical Aerodynamic Simulation and the programs in the benchmark have therefore a strong relation to this field. There is an extensive description of the NPB that can serve as a paper-and-pencil version of the benchmark together with rules for allowable extensions and libaries. Also a reference implementation is available that can serve as a basis for running the benchmark. When reporting the results one should describe in detail what modifications have been applied. The benchmark consists of 5 so-called kernels, i.e., programs consisting only of an important part of an application, instead of a complete application and three types of linear solver programs. For each of the kernels and complete programs input sets of increasing size are prescribed named A, B and C. This corresponds to the requirement for a range of input sizes as formulated earlier as a necessary property for a synthetic benchmark. Recently the so-called multi-zone version of a subset of the NPB benchmarks has been published: the LU, BT and SP on collections of loosely coupled discretization meshes. Parallelisation over the meshes is implemented in MPI: parallelism within the meshes is implemented in OpenMP. The number of meshes (and therefore the scalability of the MPI parallelism) is fixed for each dataset size. For two of the applications, all the meshes are the same size. For the third (BT), the

meshes are of different sizes, which results in a difficult load balancing problem. Unfortunately, the programs in the NPB are too complicated to relate them well to machine properties. They may have their use in a restricted part of the CFD application area but the mapping of the NPB results to actual CFD applications will generally be fairly speculative. It is therefore hardly fit to serve as a synthetic benchmark for our purposes.

- **PERFECT** Set of 13 compact applications. Development is frozen. Not useful for our purposes as the interpretation of the results in relation to other applications is next to impossible.
- **SPEC** There is a large variety of benchmarks from the SPEC organisation ranging from a Java benchmark to a database benchmark. For the HPC community the SPEC CFP 2006 floating-point benchmark is the most relevant. The separate communication oriented SPEC MPI benchmark is discussed below. SPEC CFP 2006 consists of 17 codes in Fortran, C(++) or a combination of them. All are complete and sometimes quite large applications ranging from QCD to weather codes, each code with its own fixed input set. As such, this benchmark is not fit for our purposes: the codes are too complex to extract behaviour information from the programs that allows for general assertions about machine performance in other areas. Of course one or more of the programs may be in the application areas one is interested in but the fixed input sets will make predictions for one's own purposes difficult.
- **STREAM** This benchmark measures the bandwidth from (cache)memory to the CPU for four cases: vector copy, vector add, vector scale, and a linked triad ($c_i = a_i + s \times b_i$, $I = 1,...,N$). The benchmark supports OpenMP parallelisation, enabling aggregate memory bandwidth measurements on multicore systems. There is also a version that extends the tests to multiple processors by means of MPI. As such STREAM has been included in the HPCC benchmark (see section 5.4.3). Similar measurements are performed in the EuroBen benchmark, however, with a larger range of data lengths. A very useful test for obtaining bounds on the possible performance of basic operations.
- **STREAM2** [12] An extended version of the STREAM benchmark which evaluates memory bandwidth as a function of vector length providing information on performance on all levels of the memory hierarchy. However STREAM2 lacks the OpenMP and MPI parallelisation support found in STREAM.
- **P-SNAP** [13] A benchmark measuring operating system interference or noise. In very large scale parallel systems this jitter can have a cascade effect which decreases both the predictability of runtimes as well as the overall performance. The benchmark executes a calibrated spin loop in each MPI task and records the actual time taken to execute each iteration of the loop.
- **Selfish** [14] Another OS Jitter measurement benchmark which measures the amount of interrupts detected over a time interval.

5.4.2 *Communication oriented benchmarks*

The benchmarks in this subsection focus on measuring the performance of the communication subsystem. The focus is on MPI benchmarks as it is by far the most popular parallelisation scheme in HPC today and will likely retain its position for years to come. However, as the number of cores per node is constantly increasing there is growing interest in the hybrid OpenMP+MPI programming model. Thus OpenMP performance should also be measured.

- **EPCC OpenMP Microbenchmarks** [15] It measures the overheads associated with various OpenMP directives. It compares the execution time of a code fragment executed in parallel and compares this to a reference sequential execution time. Three classes of

overhead are measured: synchronisation, loop scheduling and creating/copying arrays. Fortran and C versions are available. The metric reported is overhead time associated with a directive. This benchmark can be very useful in interpreting the results of other basic benchmarks where OpenMP overhead is encountered.

- **Intel MPI Benchmark** [16] Very extensive benchmark of essential MPI functions. Point-to-point as well as collective operations and barrier synchronisation are assessed. Also a collection of MPI-IO functions are included. Very useful for obtaining bounds on the communication performance of a system.
- **SPEC MPI** [17] The SPEC MPI2007 benchmark contains 13 applications over a large range of application areas. The same problems as for the SPECfp 2006 benchmark programs hold: the set of programs is too complicated to draw sensible conclusions from, except when an application from the benchmark matches particularly well with one of one's own applications. However, drawing general conclusions about a machine by means of this benchmark is not possible.
- **SKaMPI** [18] A comprehensive set of MPI benchmarks, similar to the Intel MPI Benchmark suite. SKaMPI provides a scripting interface which can be used to extend its functionality by adding measurements for additional MPI routines and custom communication patterns.
- **Sandia SMB** [19] A set of benchmarks evaluating message throughput and host processor overhead of high-performance network interfaces. Currently only the host processor overhead microbenchmark is available. The benchmark measures a system's ability to concurrently perform communication and computation when using the asynchronous MPI_Isend and MPI_Irecv communication routines. This can affect significantly the performance of programs designed to exploit overlapping communication and computation.

5.4.3 *Combined benchmarks*

Combined benchmarks are collections of various subbenchmarks measuring different aspects of the system. They simplify comprehensive system evaluations by providing a uniform interface for compilation and execution of the subbenchmarks as well as usually providing commonly formatted output.

- **EuroBen** The EuroBen benchmark set contains three subsets addressing single-CPU, OpenMP, and MPI performance. The subsets themselves are divided in a "module 1" and "module 2", respectively. Module one measures the performance of basic operations and intrinsic mathematical functions while module 2 measures the performance of basic algorithms, like the solution of dense and sparse linear systems, FFTs, random number generation, etc. Although not complete with respect to basic algorithms, it gives performance information about the building blocks of a wide range of application areas. As such it would fit in the criteria for a good synthetic benchmark. The MPI subset contains measurements of basic MPI communication functions, be it less extensive than the Sandia SMB.
- **HPCC** [20] The HPCC benchmark was expressly put together for finding out important characteristics of large HPC systems. It includes other benchmarks already discussed: HPL and STREAM to assess the maximum attainable floating-point speed for dense linear systems, including the communication, and the processor-memory bandwidth, respectively. Furthermore, a matrix-matrix mulitiplication which correlates very highly HPL and therefore does not provide much extra information, an FFT program and a

parallel matrix transposition. The latter program is again strongly related to the FFT outcome as in a parallel FFT a matrix transposition is the most important factor in the interprocessor communication. An interesting feature of the FFT implementation is that it is hybrid: locally OpenMP can be used while the interprocessor communication is done with MPI. For instance, a parallel matrix transposition is included in both the FFT and the Wavelet Transformation programs in EuroBen. In addition, there is a program performing random memory updates to obtain information about the worst case memory bandwidth and a program that tests a variety of MPI communication patterns, similar but not identical to those in EuroBen and IMB.

- **PARKBENCH [**21] In principle PARKBENCH adhered to the same ideas that are the basis of the EuroBen benchmark: it should be hierarchical, starting with basic operations and increase the complexity of the codes stepwise with basic algorithms and possibly beyond. In PARKBENCH the hierarchy consists of low-level benchmarks, kernels, and compact applications. The low-level and part of the kernel benchmark programs are very similar to a part of the EuroBen benchmarks, module 1 and 2, respectively. However, the programs not concerned with dense linear algebra are in fact the NPB kernels that address the topics PARKBENCH wants to cover: e.g., the PARKBENCH FFT kernel benchmark is in fact the NPB FT code. As already remarked before, the NAS Parallel Benchmark codes are too complex to draw consistent conclusions from and are therefore of less interest when building a reliable synthetic benchmark. The same remarks apply to the Compact Application part of PARKBENCH. It consists for the larger part of the CFD codes in the NPB and adds the shallow-water code PSTSWM which is also a component of the PERFECT benchmark (see above). PARKBENCH, version 2.1 is still available from netlib but development and maintenance are frozen since 1996.

5.4.4 *IO Benchmarks*

IO Benchmarks measure the performance of the disk subsystem. The focus should be on measuring both the predominant POSIX and emerging MPI-IO interfaces separately as their performance does not necessarily correlate with each other.

- **IOR** [22] The IOR benchmark is designed to measure the parallel read and write access performance of various parallel I/O interfaces in HPC systems. Access patterns (sequential vs. random), file size, block size, and concurrency (one file per process vs. shared file). Currently POSIX, MPI-IO and HDF5 interfaces are supported, enabling performance comparisons between different I/O interfaces on a specific platform. This is especially pertinent for evaluating the quality of MPI-IO -implementations, which seems to vary.
- **IOZone** [23] The IOZone benchmark is a widely used tool for assessing a variety of POSIX file operations. IOZone is useful for assessing serial and parallel performance within a single SMP node using threads. However, the ssh/rsh-based multi-client mode for running in parallel on a distributed memory system is not as scalable and portable as MPI-based parallelisation.
- **Bonnie++** [24] A filesystem benchmark similar to IOZone. In addition to bandwidth tests, it offers a set of tests to evaluate metadata operations such as file creation, deletion and file status (fstat) lookups.
- **B_eff_io** [25] A benchmark which aims at producing a characteristic average number for the amount of the achievable I/O bandwidth on a system using a number of different access patterns found in parallel applications.

5.5 Interpretation of results

To obtain results that are comparable across different systems, a set of run rules is required that regulates the way the benchmark codes should be executed. Most of these rules are obvious but to forgo inconsistencies in the results we should explicitly state them. Furthermore, we cannot ignore the existence and increasing occurrence of computational accelerators. We may expect them to be present in some form in the Pflop/s-prototype systems that PRACE is targeting and it is to be expected that at least some of the codes in the set of synthetic benchmarks could be run on such accelerator hardware without excessive effort to modify them for the particular accelerators at hand. This, however, also necessitates precise rules about what code modifications are acceptable.
In the following section we assume that for all programs in the benchmark set, a generic program code is available, the collection of which we call the base implementation of the benchmark.

5.5.1 *Run rules*

Standard CPU configurations

With regard to systems with general CPUs one should at least perform an “as-is” benchmark run. With “as-is” is meant that no modifications in the base implementation is allowed other than the minimum to obtain a correct result. When indeed such modifications are required, they should be reported. Furthermore, the following testing circumstances for each run should be provided:

a) System type
b) Number of processors, cores/processor, and processors/node (if applicable)
c) Amount of memory/processor or memory/node (if applicable)
d) Compiler version
e) Compiler optimisation flags
f) Libraries used
g) Operating system version
h) Date and time of run

Both compiler flags and libraries should be generally available to the HPC community, i.e., they may not be special flags/versions only used for the benchmark but be accessible to anyone who wishes to reproduce the benchmark run. It is, however, permitted to exchange benchmark code in the programs by calls to optimised libraries performing the same algorithms. Calling sequences and parameter types for the library versions should, however, be identical.

In addition to an as-is run, optimised runs could be performed. Here the same rules as given above hold except for the fact that one may modify the code for higher performance. This does *not* extend to altering the algorithms that are used in the codes. Even when the computational result would be the same one should adhere to the original algorithm. For instance, it is not allowed to use Strassen's algorithm for matrix-matrix multiplication when the original code contains the usual $O(n^3)$ multiplication algorithm. Merely a more efficient implementation may be attempted. In case of an optimised run the same information as for as-is runs should be provided. In addition, the code modifications should be detailed with respect to the base implementation.

Of course it is not allowed to take advantage of the knowledge of the outcome of a program, i.e. skip (part of) the computations if this would be possible. Although this is conceivably not the case in the as-is runs it might be possible in the optimised runs.

For all programs that are executed the complete output should be available. If a program did not execute correctly or not at all, this should be reported.

Accelerator-enhanced configurations

It is highly probable that the computational accelerators in their host systems may not be able to run a full benchmark set. They are mostly added to accelerate a limited set of algorithms extremely well while they are not accommodated to or even unable to perform algorithms outside this set. It is also clear that in most cases it will not be possible to use the standard source code as given in the base implementation. Depending on the type of accelerator and the libraries that are available for it, it may be necessary to insert variable types and library calls that enable the execution of the algorithms of choice. For some accelerators BLAS, LAPACK, and FFT libraries are available and in this case the modifications need not be extensive. In other cases it may be necessary to include routine calls for the transportation of data to/from the accelerator from/to the host system perhaps with double- or multi-buffering for efficiency's sake. As we can consider the execution of programs from the benchmark set on accelerators not as as-is runs but as optimised runs, this is in principle no problem as long as the program's semantics not are violated. So, the same rules as for optimised runs on standard-CPU systems are valid, be it that the description of code modifications and the supporting libraries will in general be much more extensive.

As is remarked before, it is in the nature of the accelerators that not all programs in the benchmark set can be executed. Although it may be possible in principle to port some programs to an accelerator, it may demand unreasonable efforts to do so. Still, whenever possible, it is highly commendable as it also can give us valuable information about the limitations of certain accelerators for the programs that have been ported. For instance, the lack of supporting hardware for reduction operations could give information about what algorithms *not* to implement on the accelerator that misses such a facility and it may help in identifying unrealistic claims from accelerator vendors.

5.5.2 *Benchmark results*

Benchmark output

There are essentially three metrics that are useful for assessing the performance of a computer system: the wallclock time, the number of floating-point operations per second (flop/s) for computational-oriented programs, and bytes per second (B/s) for communication and I/O-oriented programs. Of these three the wallclock time is the fundamental metric on which all other metrics depend. From the three metrics mentioned additional ones may be derived: e.g., instructions/cycle for code efficiency or bytes per flop (B/flop) for system balance. This means that the programs in the synthetic benchmark set at least should have wallclock time and flop/s or B/s as output, depending on the metric of interest. The output thus becomes a component of the *performance profile* for the system that is being benchmarked. The performance profile being the set of benchmark results that serve as a basis for the conclusions about the strong and weak points of a computer system. The results of the individual programs can be used for internal comparison, like the fraction of the theoretical peak performance that can be attained, or external, comparing it with the results from other

systems for the program at hand. A database with the results for the systems thus measured can be of great help in the interpretation because of trends for various types of architectures that might be discerned. The basic metrics in the database can be combined to obtain the derived metrics one might be interested in.

Correctness checks
The programs need to have internal correctness checks to be sure that the correct amount of data has been processed and has yielded the correct results. Obviously, only programs that produce correct results will be admissible to the database of results.

5.6 Conclusions and Future Work

From the previous sections it will be clear that a PRACE synthetic benchmark set (PRACE-SBM) should consist of programs that targets four system properties:

- Computation
- Memory bandwidth
- Interprocessor communication
- I/O

The PRACE-SBM can partly be composed from existing benchmarks that already suit our purpose of learning about the system properties and building the performance profile. However, not all issues required for a complete picture are addressed with equal quality or at all. To streamline compiling, execution and interpretation of the results, the benchmark components will be integrated into the PRACE benchmark suite. However the current version of the JuBE framework, which is the suite is based on, is designed mainly for running application benchmarks which generate a small number of metrics (usually just the wall clock run time is sufficient). On the other hand a synthetic benchmark such as Euroben may generate hundreds of individual results which should be processed and stored.

This leads us to the following recommendations, which also defines a work program for components still missing and how to go about adaptability and maintainability of versions.

1. Compose an initial PRACE-SBM from components from existing benchmarks with acceptable relevance and quality.
2. Integrate the PRACE-SBM components into the PRACE benchmark framework (JuBE) in collaboration with the benchmark integration subtask.
3. Collaborate with JuBE developers in improving the framework to better suit PRACE-SBM.
4. Fill in the blanks that have not yet have been addressed in the initial benchmark set.
5. Monitor the application space in order to keep up with current/new algorithmic practices and develop new programs that implement them as kernels. This also means that obsolete kernels should be removed as the programs should reflect the state-of-the art in HPC computation, communication and I/O practices.

For a first practical approximation for the PRACE-SBM we will start with merging (part of) the EuroBen and HPCC combined benchmarks, the EPCC OpenMP benchmark, the IOR and Bonnie++ I/O benchmarks, the STREAM2 and P-SNAP benchmarks, and the SKaMPI and SMB MPI benchmarks.

Future development of the PRACE-SBM benchmark components should focus on the following:

- *Computation*
  - Implement sparse eigenvalue, multi-grid, and molecular dynamics kernel programs for single-CPU, OpenMP, and MPI versions.
- *Memory bandwidth*
  - Implement a memory bandwidth benchmark combining the OpenMP parallelisation functionality of STREAM and the support for variable vector length of STREAM2.
- *Interprocessor communication*
  - Implement benchmarks for testing MPI+OpenMP hybrid programs.
- *I/O*
  - Implement an MPI-parallelised test for metadata performance.

The result of this work will be reported in D6.3.2.

6 Performance Analysis Tools

6.1 Introduction

In High Performance Computing, application performance is the ultimate goal and test for the success of the combination of hardware architecture, system software and application code. In order to analyse performance, tools are used to identify and characterise performance bottlenecks encountered with existing and proposed algorithms. This section will build on our experience in the use of current tools on existing systems. It is unlikely that Performance Analysis Tools (PATs) will scale without modification to ~10,000 or more processes, i.e. to be able to assess Pflop/s performance.

For the remainder of this section, we will use the following descriptions for monitoring, profiling and tracing:

- Monitoring: Performance data collection via hardware counters (the number of executed floating point instructions, the number of cache misses statistics on branch instructions, etc.). This also covers application memory usage and application I/O activity;
- Profiling and tracing: Profiling collects aggregated information about certain events, whereas tracing records information about individual events. Events are function calls, communication or other activities. Profiling typically yields total runtime per function, number of calls and a call tree.Tracing allows one to investigate the dynamic behavior of a single function over many iterations. Profiling and tracing often require instrumentation (modification) of the target application. This can be done by using compiler flags, by either manual or automated source level modification, by directly instrumenting a binary executable or by using instrumented runtime libraries.

Since Performance Analysis tools are very diverse by nature, there is no uniform way to describe or assess them. In the following sections we will describe a number of tools or toolkits. To exemplify the tools we standardize on the well-known Linpack benchmark (HPL) that is used for the TOP500 list. The rationale for choosing HPL is the portability to all platforms and its working is well understood. Since the main purpose is exemplification, we did not try to fully optimise or even scale the HPL run.

This section does not intend to be an exhaustive market survey of all available PATs. Our selection approach has been to initially evaluate a mix of PATs: from hardware vendors, from software vendors and Open Source (typically developed at universities and/or research institutes). We have mapped these PATs to available hardware architectures and to available expertise, which has led to the set of investigated PATs in the following subsections. With respect to future work in this subtask of task 6.3, PATS will be heavily used in tasks 6.4 and 6.5, leading to more details for 6.3 as well. Based on these experiences, we expect to be able to really feed into the development cycles of hardware and software vendors, and in the Open Source development of PATs.

6.2 Allinea Optimisation and Profiling Tool (OPT)

6.2.1 *Introduction*

Developer
Allinea Software, more information in [26].

6.2.2 *Availability*

Commercial product, license required, free 30 day trial available.

6.2.3 *Supported Platforms*

Allinea OPT is available for almost every flavour of Linux, for Itanium, Opteron, EM64T, Xeon, PowerPC and the IBM Cell BE. BlueGene/P support has recently been added. A complete list of the supported platforms is available from the website.

6.2.4 *Assessment Environment*

- **IBM BG/P**
  A one-rack (4096 cores) IBM BlueGene/P system at STFC Daresbury Laboratory (4096 cores) has been used as the platform for the evaluation.
- **Allinea OPT v1.4.2**
  Recent updates have added support for IBM BlueGene/P, SGI Altix and IBM Cell BE platforms.
- **Linpack**
  The open-source package Linpack is a numerical solver for a dense system of linear equations.

Description of the tool
OPT is a development tool for improving the performance of MPI and scalar applications. It gathers profiling information by instrumenting the MPI communication layer. OPT is a grid-enabled application that uses the web-service protocol SOAP to allow profiling users to access OPT remotely and securely with a minimal amount of communication bandwidth. OPT's graphical interface uses remote (or local) OPT servers to launch applications, store performance data and analyse user applications.

Design Features
- Allinea OPT has been designed for use on large-scale parallel systems.
- Supported languages: Fortran, C and C++.
- Easy generation of different data formats.
- Grid capable. This allows users to access remote profiling data almost as rapidly as a local server.
- Multiple runs can be compared to assess code scalability.
- Interoperable with other profiling tools e.g. PAPI hardware counters or gprof.

- Subsets of processors and time intervals of interest can easily be selected in order to keep levels of profiling data manageable.

Profiling Methodology
OPT consists of three components (which can reside on separate machines):

1. A library that is linked to your application code and records performance data.
2. A database component (the profiling server) to store trace data, and combines them into profiling data. This consists of a PostgreSQL Database.
3. A graphical user interface that is able to interpret both the trace and the profiling data and displays it in user-specified formats.

Once that the OPT server software and the OPT profiling library is installed on the system (an installation GUI is provided for this) there are three separate stages involved in profiling users' application codes. Firstly, the application source code is instrumented by wrapping the existing MPI calls. Either the whole code can be profiled, in which case no changes to the source code are required, or the application can be edited to use the OPT APIs in order to undertake selective profiling or add profiling extra features. On the IBM BG/P platform the following additions to the compile/link line were required:

- The OPT header-file include directory
- The OPT static library directory, OPT and OPT-support libraries
- The GNU C++ stdc++ library
- The dynamic-linking library

Instrumented executables can be run in exactly the same way as 'normal' executables, either interactively or in batch mode. Jobs can be launched either from the command line or the OPT GUI. During execution profiling data is collected in the profiling database.

Once the job has completed the MPI profiling data can be viewed in several different formats. A selection of these views are summarized briefly here:

Timeline View
This is a chronological display of the users program. The profiling information from each processor is listed as a separate line and within each line the coloured boxes represent MPI communications or other MPI function calls. Areas of interest in the timeline can be navigated via zoom and mouse drag or time intervals can be entered manually. Communication lines representing messages can also be toggled on/off. Individual message occurrences can also be highlighted and detailed performance data can be obtained (see figure 5). The timeline view can be useful for highlighting asynchronous behaviour or load imbalances between processors. The timeline is best when we consider only a small section of the actual run time of a program – so long as this section is representative of overall performance.

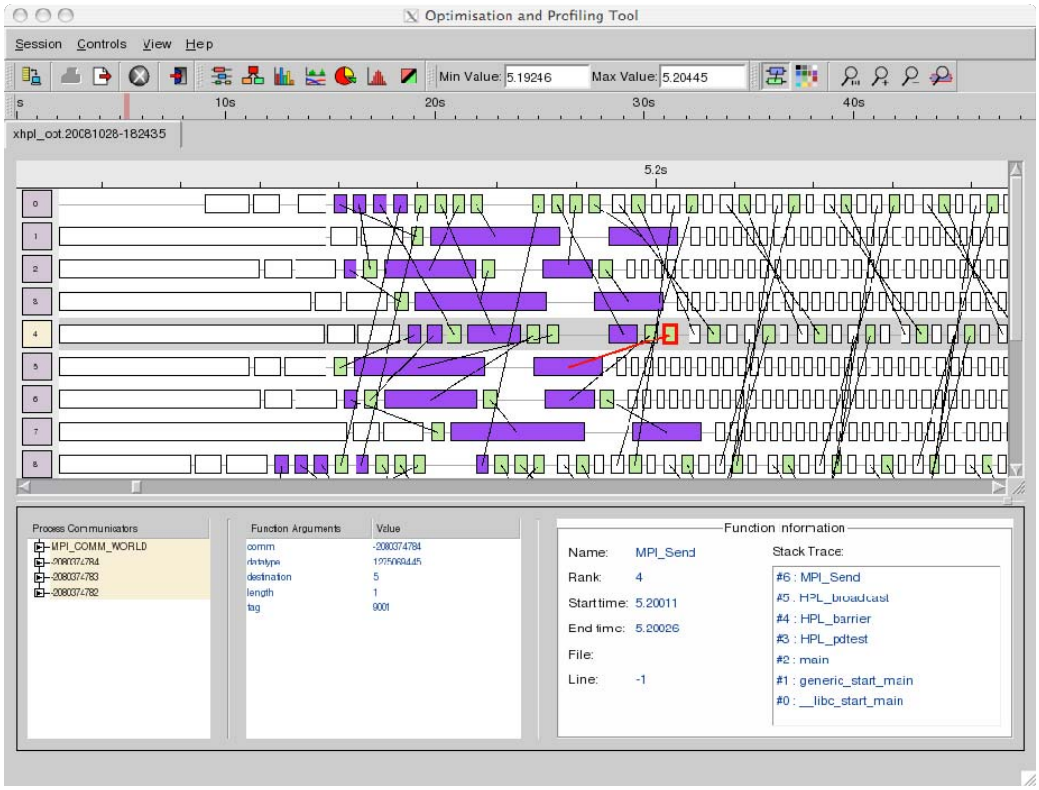

Figure 5: Screenshot of timeline view of OPT Linpack run on IBM BG/P.

Figure 5 shows an occurrence of MPI_Send from process 4 to process 5, which has been highlighted for detailed analysis.

Call Graph View
This view allows users to determine how MPI functions were called and to discover the source of an MPI function call. This enables the user to track problems in balance, raw or cumulative resource usage. gprof data can also be viewed from this display.

Histogram View
This view arranges metric values from processes into buckets and gives a view of selected measurements by plotting a histogram (figure 6). By viewing the data in this format users can easily identify load imbalances between selected processors.

Message Profile View
The Message Profile view provides a summary view of point-to-point communications between different processors. The information is provided in the form of a grid (figure 7). Metrics such as bytes transferred, number of mpi calls and time spent in mpi communications can be selected. The number of bytes communicated between processors is the displayed metric.

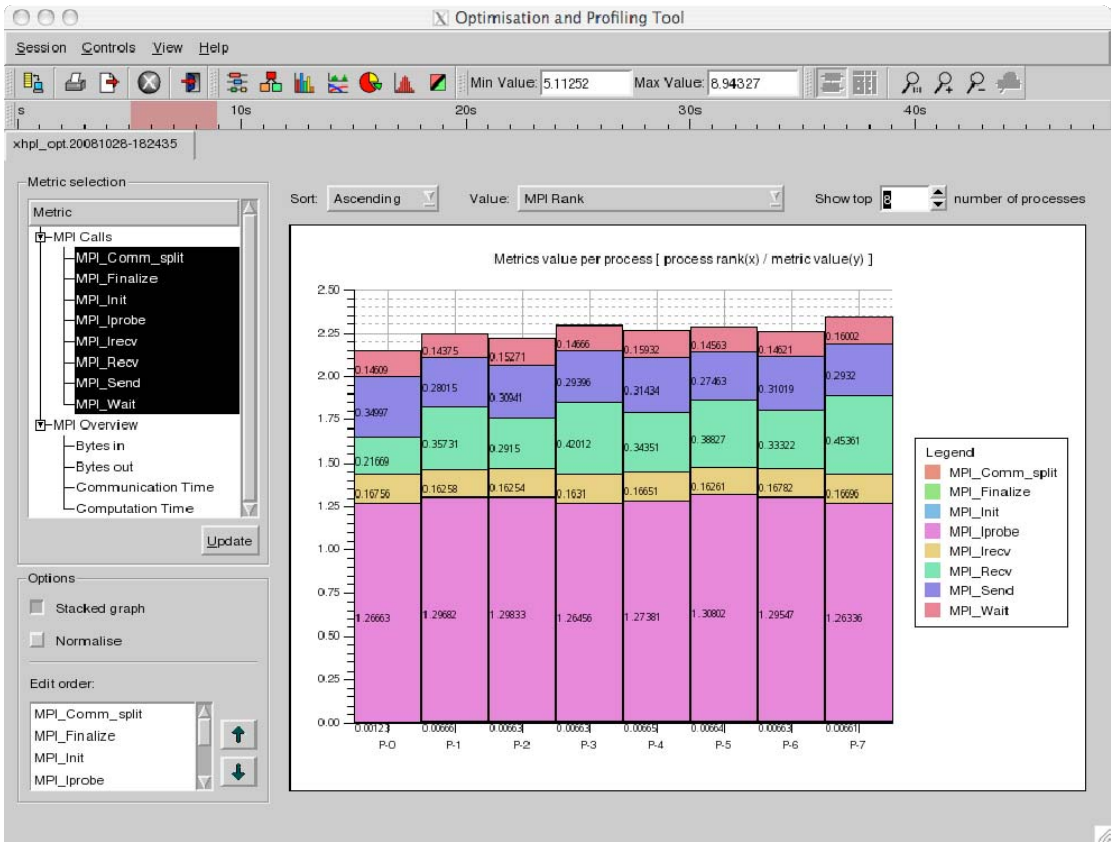

Figure 6: Screenshot of histogram view of OPT Linpack run on IBM BG/P.

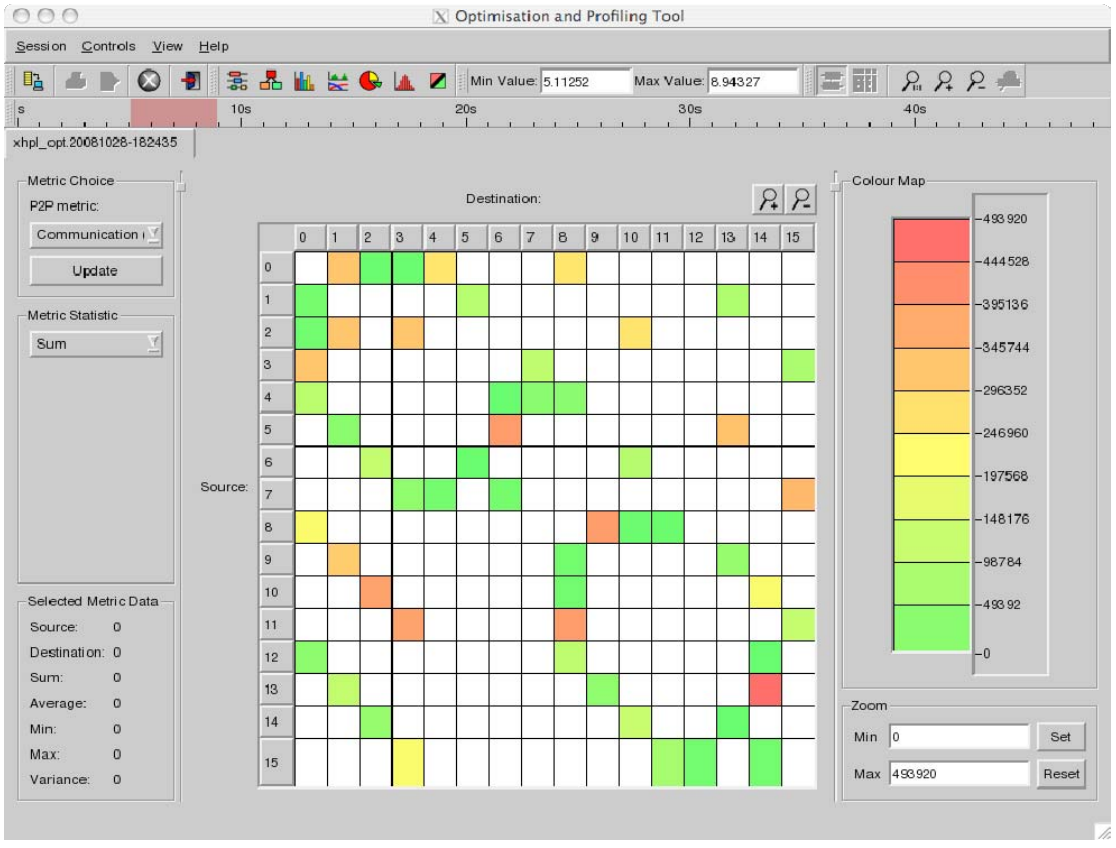

Figure 7: Screenshot of Message Profile view of OPT Linpack on IBM BG/P.

Evaluation

Advantages

- A highly detailed analysis of communication can be obtained, right down to individual messages between processors;
- Load imbalances between processors can be identified very easily;
- The chronological timeline views allow users to focus on areas of interest;
- Several options available in the user interface for filtering/selecting both communication routines and parameters of interest;
- Jobs can be run, traced and viewed all from within a single graphical user interface;
- The OPT function calls can be used to stop and start logging where required. This technique can mitigate some of the performance disadvantages listed below;
- The documentation is very detailed and specifies installation procedures for several HPC platforms, including the IBM BG/P;
- Allinea customer support was very responsive to queries.

Disadvantages

- Logging the communications can be slow and heavily impacted on the performance of the Linpack benchmark code. The reported speed of Linpack dropped from around 6 Gflop/s to around 1 Gflop/s;
- The GUI becomes less responsive when communications events increase (e.g. with larger processor sets);
- The setup is relatively complicated. For example the user must install a PostgreSQL database server on the target platform. However this is usually bundled with the OPT software and installation on IBM BG/P was relatively straightforward;
- The tracefiles can become very large and loading these into the GUI becomes time consuming. This even became a problem for 16 processors of the HPL benchmark lasting less than a minute;
- There is no OpenMP or mixed-mode support;
- OPT profiles only MPI communications routines, not computational routines. However the tool can be combined with gprof and PAPI to provide such information.

Future work
More details and experience need to be gained when scaling up to many cores. This will be done by applying Allinea OPT to the Helium benchmark code on IBM BG/P.

6.3 CEPBA-Tools: Paraver & Dimemas

6.3.1 *Introduction*

The CEPBA-Tools environment for performance analysis is a set of tools being developed and maintained by BSC [27]. Two major tools constitute the core of the environment: Paraver and Dimemas.

Paraver is a flexible browser for traces. It can display timelines (with very scalable display mechanisms) of a wide variety of metrics (activity, hardware counter derived metrics, communication bandwidths,…). It also provides detailed statistics (averages, histograms) of those metrics for any desired interval of the trace. Paraver comes along with MPItrace, the instrumentation package that generates traces for MPI + OpenMP parallel programs.

Dimemas is a coarse grain simulator to estimate the impact on the performance of MPI applications of parameters such as network latency, bandwidth, contention, processor speed, etc. It interoperates with Paraver (can make the prediction starting from a Paraver trace of a real run) and generates Paraver traces of what would be the behavior under the target system characteristics.

6.3.2 *Availability*

Paraver and Dimemas along with the trace generation and handling utilities are distributed in binary form. BSC is preparing an Open Source Distribution in a near future.

6.3.3 *Supported Platforms*

Versions of the instrumentation libraries to generate traces are available on the following platforms: Linux-x86/AMD/PPC clusters, AIX-PWR4/PWR5, Cell, BG/L, Altix, SX8, CRAY XT3/XT4.

Paraver and Dimemas as such run on standard Unix/Linux machines. This includes servers and laptops. A typical way of operation is to obtain the trace on a parallel machine. A normal trace analysis with Paraver would require the machine to have at least 1GB of memory.

6.3.4 *Scalability*

The main limitation in the Paraver analysis is the trace size. Paraver can easily handle traces of up to 200MB on a standard laptop. Above this size, response times may get too large. Different techniques have been developed to summarize traces of tens of GB such that the resulting traces can be visualized with Paraver.

Non linear rendering techniques included in Paraver result in the possibility to visualize traces with a large number of processors.

As an example of the above techniques, figure 5 shows a timeline of 10,000 cores running Linpack for 1,700 seconds. The three views of the same region of the trace show the duration of the dgemms (left), their IPC (center) and their L1 misses (right). We can see the extreme precision of the analysis, showing for example differences in IPC of less than 3%. It is also possible to identify 4 processes (one node) out of the 10,000 that showed slightly higher L1 misses than the rest.

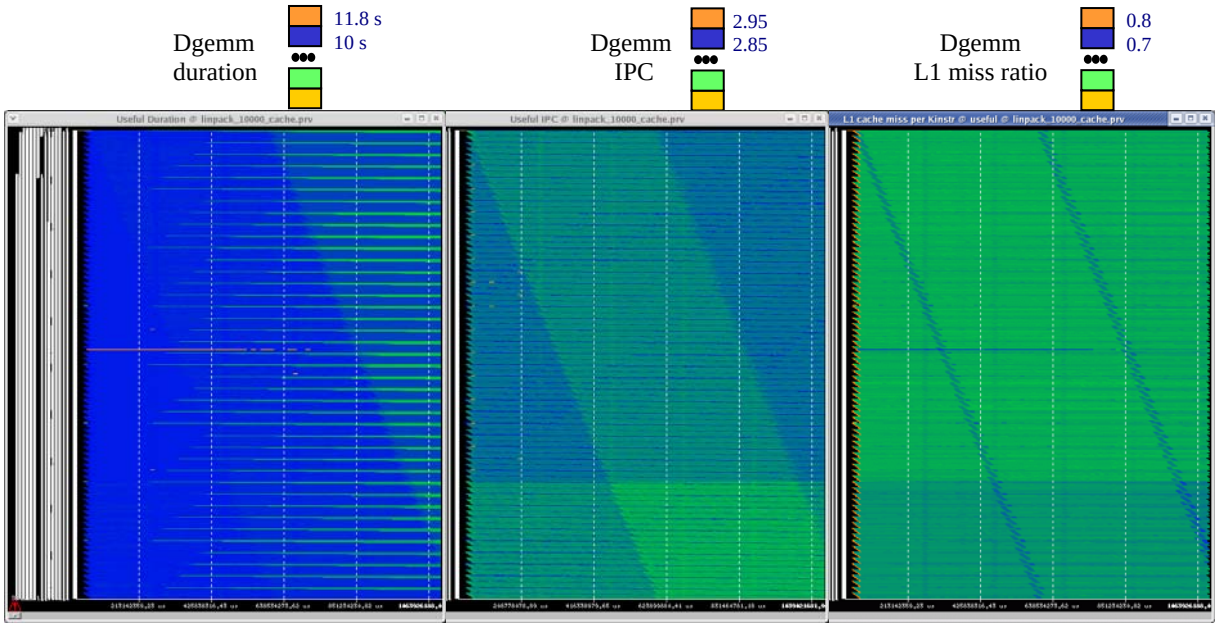

Figure 8: Linpack@MareNostrum, 10k cores x 1700 seconds.

6.3.5 *Paraver Details*

Methodology

A normal Paraver analysis consists of several phases:

- Obtaining the original trace
- Obtain representative subtraces to be visualized with Paraver.
- Detailed analysis with paraver

Tracing

Paraver comes along with the MPItrace instrumentation package. The basic mechanism to inject probes relies on the LD_PRELOAD environment variable. MPItrace handles pure MPI as well as MPI+OpenMP programs. Support for Pthreads is also available.

Normal production binaries can thus be instrumented. In order to instrument a run it is only necessary to modify the mpirun invocation in the submission script by calling a trace script in front of the user application. The actual instrumentation is controlled by an xml file where the analyst can specify the hardware counters, level of detail in the tracing, how to handle the intermediate files, etc. Both the trace script and the control xml are provided with the distribution.

MPITrace generates one .mpit file per process. A merger utility is provided to match and merge the different files into a single one.

Subtraces

When starting Paraver and trying to load a file larger than what it can properly handle for visualization Paraver will offer several possibilities to compute some preliminary statistics on the trace and manipulate it to reduce its size. This includes the possibility to filter it. Typically, the process consists of generating a summarized trace that spans all the duration of the original one but where only a subset of the records is included. An xml file is used to

control how to perform the filtering. Typically one would discard all communication records and just keep computation records of sufficient duration.

This summarized trace can be loaded with Paraver and its periodical structure identified. A utility lets the analyst specify on this trace which region of the original file seems to be a relevant section (i.e. a couple of periods) and thus generate a fully detailed trace but only for that interval.

This process is typically performed with Paraver installed on the parallel machine where the trace was obtained. The summarized traces can then be copied to a laptop where the analysis can be done locally.

Analysis
Once a representative trace is obtained it is possible to perform extremely detailed analyses. These can be either visualization of how performance metrics (duration, IPC, bandwidth,….) evolve with time or the corresponding profiles (MPI calls, User functions) or histograms (duration, hardware counts, derived metrics,…). Each such view pops up when loading a configuration file previously saved by an expert.

Assessment
The tool is extremely powerful and in the hands of an expert can help identifying or pinpointing the performance problems in an application. The great flexibility in terms of generating the displayed metric offers the possibility to perform very precise measurements. The possibility to generate histograms of any such metric and to go from there to the timeline is a very powerful mechanism to identify regions of specific behaviour. Although the tool has a long learning curve, it proves very useful in cooperative analyses, where an expert analyst presents her observations to an application developer and jointly discuss on the observed behaviour.

Although being possible to analyse very large runs on large configurations, the process of trace generation and initial handling is cumbersome. Traces of several tens of GB can be handled if the target machine is large and has storage enough, but the time spent in the merging and sub-trace selection process, can be a nuisance.

Once the appropriate sub-trace is available, the initial views can help understand the general behaviour. A typical observation is that the GUI has too many windows and several clicks on different windows have to be done to achieve a given effect. On the other side, the possibility of loading several traces and copying time scales form one to another provides a very natural way to study the scalability of applications.

Even if it is possible to dig down into possible causes of performance problems, there is no standard methodology for a novice user that just provides a basic general description of the behaviour. It is easy for a user to get lost with so many possible views and histograms. Sometimes the doubt arises of what is exactly measured by a configuration file.

6.3.6 *Dimemas Details*

Methodology
A normal Dimemas analysis would have the following steps:

- Convert a Paraver trace to Dimemas trace
- Validate the Dimemas simulation
- Perform parametric studies

Trace conversion
The trace that we have been analysing in the previous Paraver session can be converted to a Dimemas trace with the prv2trf tool provided with the environment. It is also possible to directly obtain a Dimemas trace form an instrumented run of the application.

Validation
A Dimemas simulation is controlled by a .cfg file. The dimemas-java utility provides a GUI to create one such control file. It contains a description of the target machine, including number of processors per node, latency and bandwidth of the network, level of contention, etc. Paraver performs the simulation reporting the predicted execution time for the specified trace on the target architecture. It also can generate a Paraver trace with the detailed timing of the predicted behavior.

A first recommended study is to simulate the converted trace on a target machine with the nominal parameters of the actual machine where the trace was obtained. This simulation will typically generate a trace quite similar to the original Paraver one. By loading them in Paraver we can compare them.

Parametric sweep
By performing different simulations varying latency, Bandwidth or contention level we can estimate their impact on the application execution time. By looking at the generated Paraver predictions we can identify the impact of those factors and whether it is uniform along the application or certain parts are more sensitive to one or the other. This type of analysis gives immediate answers to typical questions by developers such as “should I try to pack data and generate less but larger messages?”, or “should I use asynchronous communications?”, or “…...”

Assessment
The tool provides very good perception of the behavior of the application and how sensitive it is to communication. It is a great complement to the original analysis of Paraver in order to identify the main issues to address in the optimisation of a code.

6.3.7 *Future Developments*

Current developments try to address some of the issues that should reduce the time spent by the user on analyses and to increase the semantic content of the information provided by the tools. On the Paraver side this includes:

- Automatic analysis: algorithmic work has already been done in areas of time analysis and clustering techniques that should now be integrated in the tool. The time analysis will ease the process to generate representative subtraces by automatically identifying the periodic structure of the trace. To further identify structure in the trace we are applying clustering techniques to hardware counter information. These techniques can be used to extrapolate hardware counters and to apply CPI stack models that give a high level identification of the bottlenecks of the computation phases.
- Integration of sampling techniques with tracing: Sampling techniques are used in standard profiles. Some tracing tools also use sampling, but we do believe that the potential of mixing both techniques is much higher than what it is currently done. We are developing techniques to obtain extremely detailed information of the behaviour of a program along time without having to incur very high overheads.
- Methodology and training: we have to develop a set of configuration files to guide a novice user in the first analyses. These metrics and models should present an abstract view of the performance of an application. It should be possible for the automatic environment to directly present these views and statistics to the user.
- Online analysis: the techniques we have developed for the offline trace analysis should be implementable in an online run time analysis. We are currently implementing preliminary prototypes of such functionality based on MRNet.

Regarding Dimemas we are developing multiscale prediction techniques that allow us to consider detailed instruction simulator and other prediction techniques for precise estimation of the impact of processor architecture on the sequential computation burst between MPI calls

Further, BSC is preparing an Open Source Distribution of the CEPBA-Tools environment. This should allow for users with specific needs to implement the features they require. Given the huge flexibility of the visualizer we envisage that the major usefulness would be at the level of tracing packages in order to emit the specific information into the tracefile.

6.4 Cray Performance Analysis Tools

6.4.1 *Introduction*

In the 1970's and 1980's, Cray Research Inc. HPC systems ("supercomputers") represented a very large market share. Engineers at Cray realised that designing and developing fast hardware was important, but also that the notion on how efficient this hardware was used, and consequently improving efficiency, was important. The originals of Cray Performance Analysis Tools (CrayPat) go back to this period. This section covers the current versions of these tools (v4.3), which we have assessed on the Cray XT5 system at CSC (which represents one of the prototype architectures within PRACE). CrayPat is owned and developed by Cray Inc. [28].

6.4.2 *CrayPat Details*

CrayPat provides access to a variety of experiments and reports that can indicate how a program is behaving during execution and where possible performance bottlenecks may lie. CrayPat consists of five major components, of which all but Apprentice2 are command line based:

- pat_build -- used to instrument the program to be analysed.
- pat_report -- a report generator, used to generate reports from the performance data captured during program execution and to export data for use in other programs.
- Cray Apprentice2 -- an optional graphical analysis tool, used to visualize and explore the performance data captured during program execution. It can also be used off-line on Linux systems.
- pat_hwpc -- an alternative to pat_build and pat_report, used specifically to perform simplified hardware counter analysis experiments and generate reports from the resulting data.
- pat_help -- an online help system, which contains extensive usage information and examples. This help system can be accessed by entering pat_help at the command line.

Features
Because CrayPat is developed for the Cray supercomputers, it is by construction well suitable for these environments. It is versatile to use and is shown to be able to analyse parallel software running on several thousands of CPUs. Because of this scalability, and the highly controllable degree of invasiveness, CrayPat is well suited for analysis of large parallel programs.

The recorded data can be aggregated or shown for different items separately. Data can be recorded from either each processing elements, or just from the given ones. Also the communication between processing elements can be recorded. This enables a very thorough analysis of the program and finding the possible performance bottlenecks.

The estimated instrumentation overhead is also recorded and can be substracted from the results in order to get more realistic and accurate results.

Moreover, CrayPat is able to analyse the I/O and memory usage of the software.

Instrumentation
The basic usage of CrayPat consists of a few general steps. The toolkit is initialized by loading a corresponding module, (usually) recompiling the program and instrumenting it using pat_build command.

Instrumentation is done by using CrayPat's command line based tool, pat_build. The executable is re-linked but not recompiled. Thus no source code modification is required for the instrumentation, but the original object files must be available. CrayPat uses static binary instrumentation, and supports performance data collection in several ways:

- Tracing: Record timestamps and arguments for all instrumented functions;

- Sampling: Samples hardware counters or callstack at fixed intervals;
- Profiling: Performs a specific sampling experiment where user + system time are sampled for all functions in a program.

Experiments are defined before program execution by using runtime environment variable. Tracing experiments can only be performed for the executables instrumented for tracing, sample-type experiments for all other executables.

The instrumented program generated by pat_build is a stand-alone version, and the original binary is preserved. Thus the created instrumented executable can be run normally, and the run-time library for measurements is transparent to the user. Automatic instrumentation at group (function) level is provided for several groups, such as mpi, io, heap, math SW, etc. CrayPat also provides several environment variables that can be used to control the program.

If the instrumentation methods provided by pat_build are not sufficient, further and more fine grained instrumentation can be performed using CrayPat Application Program Interface (API).

By default, the resulting data files are written to the execution directory. This directory must reside on a file system that supports record locking, such as the Lustre file system.

Hardware Counter Data

CrayPat runtime environment variables enable one to collect a wide variety of performance analysis data using the same instrumented program. It can be set to collect different sets of hardware performance counter data, either by groups or by choosing individual hardware performance counters. These are usually presented as a combination of actual PAPI counter values and metrics derived from these.

An alternative method is to use pat_hwpc command, which performs simplified hardware counter analysis and also automatically generates reports from the resulting data. pat_hwpc uses an uninstrumented version of the program, as is does the instrumentation by itself. It is usually the quickest way to acquire basic hardware performance information.

OpenMP

For programs that use the OpenMP programming model, CrayPat can measure the overhead incurred by entering and leaving parallel regions and work-sharing constructs within parallel regions, show per-thread timings and other data, and calculate the load balance across threads for such constructs.

For programs that use both MPI and OpenMP, profiles by default compute load balance across all threads in all ranks, but you can also see load balances for each programming model separately.

On Cray XT systems, the user is responsible for inserting API calls himself. There is a variety of C and Fortran functions that can be used to instrument OpenMP constructs for compilers that do not support automatic instrumentation.

Reporting the results
The reporting tool, pat_report, performs data conversion and combines information from binary with raw performance data into text report of performance results. It also formats data for the graphical analysis tool Cray Apprentice2.

The standard raw data .xf report files require the original instrumented executable to be available to provide mapping from addresses to function names and source line numbers, whereas Apprentice2, or .ap2 files are self-contained. Therefore, converting the results to .ap2 files is recommended. By default, pat_report does this automatically, if an .xf file or a directory containing .xf files is specified.

pat_report provides text reports of various formats, depending on the given parameters. Its main features include

- Profile by groups
  - Threshold
  - Load balance information
  - Imbalance metrics
- Function Profile
  - Flat profile
  - Call Tree view
  - Callers view
  - Hardware counters information
- MPI Profiler
  - MPI Load balance
  - MPI Stats by bin
- I/O Statistics
  - Read and Write Statistics
- Heap Statistics
  - High water mark
  - Memory leaks

In addition to the standard reports, pat_report can create highly customized reports tailored to specific needs. This is done by specifying the data to be included in the report, specifying how the data is to be aggregated and labeled, and specifying how the resulting information is to be displayed. Reports can be tailored for spreadsheets as well.

Apprentice2
Cray Apprentice2 is a post-processing performance data visualization tool. It is not a component of CrayPat, nor is it restricted to analysing data generated on any particular Cray system. Rather, it is a platform-independent post-processing data visualization tool. After the program is instrumented for a performance analysis experiment, executed and one or more performance analysis data files are generated, Apprentice2 can be used to explore the experiment data and generate a variety of interactive graphical reports. As a GUI tool it requires that the workstation support the X Window System.

Apprentice2 can display a large amount and variety of data, but it is dependent on the options selected when the program was instrumented and on the runtime environment variables specified when it was executed. Environment variable PAT_RT_SUMMARY can be used to summarize and aggregate the data.

Apprentice2 provides various reports, including:

- Overview Report
- Load Balance Report
- Traffic Report of various kinds
- Activity Report
- Call Graph
- Function Report
- I/O reports of various kinds
- HW counter reports or various kinds

6.5 DewizPat – Automatic Communication Pattern recognition

6.5.1 *Introduction*

Over the years GUP has developed its tools NOPE and ATEMPT for generating traces from running MPI programs and visualizing communication between processes based on events between them using a logical clock timescale. Therefore users can already better understand their codes or find bugs visually (e.g. receives without corresponding sends).

The fact that applications run on thousands of cores at a time, makes it impossible to find those patterns through "simply having a look", there is ongoing work on automatically finding repeating or typical patterns within those traces and also linking them to source code lines. Recent research focuses especially on efficiently searching for inefficient use of communication. This section gives some detail on NOPE and ATEMPT.

6.5.2 *NOPE and ATEMPT: Details*

In figure 9, taken from the GUP Eclipse Traceviewer, we show a trace from a synthetic benchmark code that performs a scatter in an inefficient way, by just using MPI_SEND and MPI_RECEIVE. The x-axis counts logical clocks in the progress of the code and the y-axis shows the 8 processes involved. A red colored event shows a process doing a send to a blue colored event being the correct receiver of that communication. In the left part of the Image one can see different properties of the event being currently selected.

The pattern recognition code is able to perform the following tasks:

- analyse the whole program trace for repeating message patterns;
- find inefficient ones (as the one from the screenshot);
- find not only local patterns, but also find global (compact) ones, i.e. at first sight there might be only 2 processes communicating, but after a more thorough analysis you actually find out, that there is something bigger going on between more processes;
- report the patterns being identified.

Currently, the output is only textual, because we care about actual automatic recognition. It delivers the patterns found and the processes involved and also which source code lines were involved in producing that patterns. Another remark that can be made is that the analysis is done very efficiently and has been tested on traces with several million events from the LLNL BG/L. But for simplicity the screenshot should give you a better idea, what it is about.

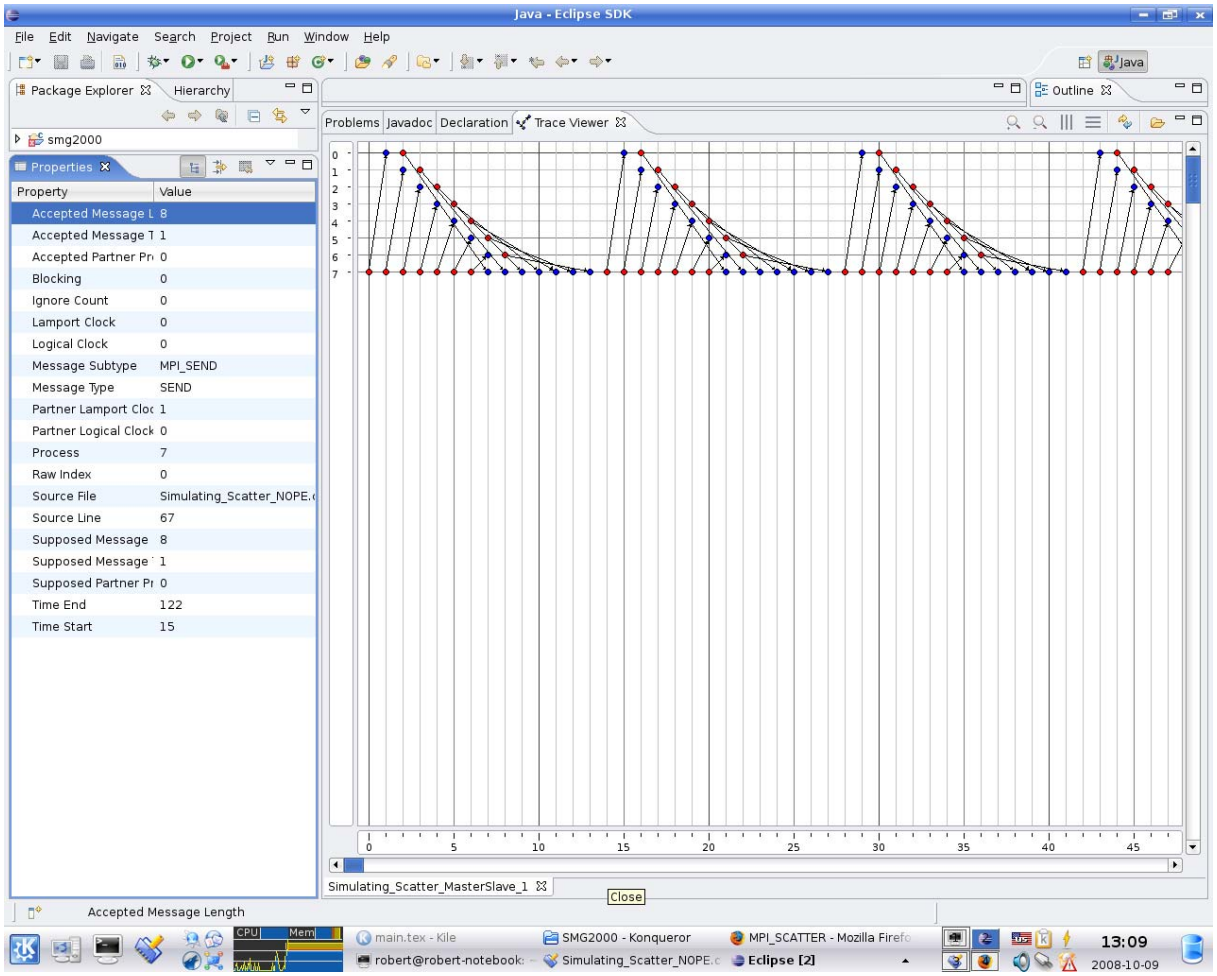

Figure 9: Screenshot of a NOPE generated Trace using the Eclipse Traceviewer.

This tool is still ongoing research and therefore only partly usable, because the NOPE tool for generating the traces can currently only deal with MPI_SEND, MPI_RECEIVE and broadcasts.

6.6 IBM HPCT: High Performance Computing Toolkit

6.6.1 *Introduction*

The IBM HPC toolkit is a versatile environment for performance analysis of sequential and parallel applications. It consists of a collection of performance data collection tools and graphical user interface peekperf. Its main features are:

- It offers an integrated environment for simultaneous investigation of all aspects of performance (e.g. cpu, memory, threads, message passing).
- No source code modifications are required.
- Performance data information is presented in a manner that highlights the relation between the performance metrics and the source code statements and data structures.
- Depending on the platform, the toolkit may contain only components that are available to the platform.

Currently supported platforms are: PPC970, Power4, Power5, Power5+, Power6: AIX 5L or Linux, BG/L, BG/P: Linux. See also [29].

6.6.2 *IBM HPCT Details*

This section covers details of the HPCT. Actual examples, obtained on the IBM PowerSeries 575 POWER6 running Linux (SLES10 SP2) at SARA (which represents one of the prototype architectures within PRACE), are given in Annex 7.5.

Hardware Performance Monitor (HPM)
The HPM Toolkit consists of:

- A utility hpmcount, which starts an application and provides at the end of execution, wall clock time, hardware performance counters information, derived hardware metrics, and resource utilization statistics like Mflop/s rates, cache misses at all levels of cache, number of load instructions resulting in TLB misses, and other measurements that that are supported by the hardware. Hpmcount is an inherently sequential program, looking only at the hardware performance counters of a single process (and its children). Using poe or mpirun one instance of hpmcount is running for each MPI task and it is either possible to create per task output files or aggregated output files.
- An instrumentation library libhpm which enables a developer to selectively instrument interesting parts of his code using an API.
- A utility hpmstat. Hpmstat is a simple system wide monitor based on hardware performance counters. The usage is very similar to the vmstat command.
- A graphical user interface PeekPerf for visualization.

On Linux the kernel has to be recompiled with the perfctr patch. In case of a CERT advisories with respect to kernel vulnerabilities this means that this patch has to be reapplied. On Power6 there are 6 performance counter registers and 195 different event groups. On Linux it is not

possible yet to monitor multiple groups simultaneously (multiplexing). At present the Linux hpmstat does not work. An example is given in Annex 7.5.1.

MPI Profiler
In order to instrument the application, it has to be relinked with an additional instrumented MPI wrapper library: `-lmpitrace –llicense`. As for the HPM library, this library allows for selectively instrumenting interesting parts of the code by inserting calls to special configuration and utility functions provided by the tracing library API. The instrumented application also generates trace files that can be visualized using PeekPerf. An example is given in Annex 7.5.2.

Xprofiler
Xprofiler is a visualization tool for `gmon.out` profiling data created by applications compiled with the `–pg` flag. Xprofiler is a nice GUI that gives the information that is also provided by the well-known `gprof` command line tool for displaying call graph profile data. Xprofiler does not yet work for 64-bit executables, which is a pity on large memory nodes (the machine we used for the assessment has both 128 and 256 GB nodes). An example is given in Annex 7.5.3.

PeekPerf
The PeekPerf GUI is the control center of the HPC Toolkit, it allows for controling the instrumentation, execution, visualization and analysis of all collected performance data within the same user interface. The dimensions of performance data provided in the current framework are:

- CPU (HPM)
- Message Passing (MPI)
- Threads (OpenMP)
- Memory
- IO

6.6.3 *Future Work and Developments*

Further investigation of the IBM HPC Toolkit has to be done, especially in the areas of POMP Profiler (OpenMP profiling), MIO (I/O profiling), SiGMA (Memory Simulation) and pSigma (Binary Instrumentation Facility). IBM is developing a new integrated version of the HPC Toolkit for the PERCS/HPCS Initiative. This is the HPCS Toolkit which will include a Bottleneck Detection Engine.

6.7 IPM: Integrated Performance Monitoring

6.7.1 *Introduction*

Integrated Performance Monitoring (IPM) is an approach to performance analysis that is focused on ease of use, scalable lightweight profiling and portability. It is less of a performance “tool” and more of a profiling infrastructure. It serves the needs of users and managers of HPC resources and is available under an open source license. See also [30].

- Developers: David Skinner team (NERSC), Open Source (LGPL)
- Supported platform(s): IPM currently runs on IBM SPs and BlueGene, Cray XT (CLE), SGI Altix, NEC SX8, various Linux clusters and the Earth Simulator.
- Platform used for the assessment : CRAY XT, IBM LinuxPWR5, see below
- Description of the tool: A scalable lightweight portable open source (LGPL) profiling tool
- Scalability: already tested on thousands of cores

6.7.2 *IPM Details*

Design goals

- Easy to use
- Parallel aware
- High level performance profiles
- Fixed memory footprint
- Minimal CPU overhead
- Portable

Overview
IPM is a portable profiling infrastructure developed at NERSC which outputs a report on the execution of parallel jobs. IPM reports MPI function timings, memory usage, and hardware counters data (where available). IPM provides a performance summary of the computation and communication in a parallel program. The amount of details reported is selectable at runtime via environment variables or through an MPI_Pcontrol interface. IPM has extremely low overhead, is scalable and easy to use requiring no source code modification.

The monitors that IPM currently integrates are:

- MPI: communication topology and statistics for each MPI call and buffer size.
- HPM: PAPI (many) or PMAPI (AIX) performance events.
- Memory: wall clock, user and system timings.
- Switch: Communication volume and packet loss

Ease of use
Insofar as performance profiling is a cumbersome process, especially at scale, users and managers of HPC resources rely on performance data from experiments that may not reflect actual workloads or worse performance monitoring will simply not be done. Ease of use is paramount in providing quality profiles from in-situ workloads. Advanced user interfaces for HPC developers and researchers are available but not required.

Scalable lightweight profiling
IPM is lightweight introducing very little overhead to running codes. At application startup and termination IPM makes good use of parallel HPC resources to aggregate, process, and store application profiles. While the application runs IPM uses a fast hashing algorithm to build the profile with minimal impact on the application. IPM runs regularly on systems with tens of thousands of tasks.

Profiling infrastructure

IPM is an approach to scalable HPC application profiling which serves both users and center managers. IPM brings together several types of information important to developers and users of parallel HPC codes. The information is gathered in a way the tries to minimize the impact on the running code, maintaining a small fixed memory footprint and using minimal amounts of CPU. When the profile is generated the data from individual tasks is aggregated in a scalable way.

The 'integrated' in IPM is multi-faceted. It refers to binding the above information together through a common interface and also the integration of the records from all the parallel tasks into a single report. On some platforms IPM can be integrated into the execution environment of a parallel computer. In this way IPM profiling is available either automatically or with very little effort.

The final level of integration is the collection of individual performance profiles into a database which synthesizes the performance reports via a web interface. This web interface can be used by all those concerned with parallel code performance, namely users, HPC consultants, and HPC center managers. Since profiles are stored centrally in a SQ L database they provide a performance track record to developers and a means of workload characterization to HPC managers. Both groups are well served by optimising application/architecture matches via in-situ performance monitoring.

Portability
IPM takes an approach to performance analysis that is focused on lightweight scalable profiling that is easy to use. IPM profiles use an XML format which allows comparison of profiles across runs and between platforms. IPM currently runs on IBM SPs, Cray XT, NEC SX, various Linux clusters and the Earth Simulator. IPM implementation is portable and is available under an Open Source software license (LGPL).

Current status of IPM

Type	Site	OS	Comments
IBM P6	SARA	Linux Suse 10.2	Papi not installed on P6 yet; Needs to modify IPM in order to use IBM HPM counters
IBM BG/P	Juelich	Linux Suse 10.1	Papi-c 3.9.0 available, needs more testing
NEC SX8	HLRS	SUPERUX 15.1	Availability of papi for NEC ?
Cray XT5	CSC	CNL 2.1.27 HD	Fully supported
Cray XT4	CSCS	CNL 2.1.26	Fully supported
IBM P5	CSCS	Linux Suse 10.2	Papi not installed on P5 yet; Needs to modify IPM in order to use IBM HPM counters

Table 11: Availability of IPM.

6.8 Scalasca

6.8.1 *Introduction*

Scalasca has been designed for use on large-scale systems including IBM BlueGene/P, but is also well-suited for small- and medium-scale HPC platforms. For a complete list of the supported platforms the reader is referred to website of Scalasca [31]. Scalasca V1.0 has been released at June 18, 2008 by Forschungszentrum Jülich. Scalasca is open-source, and requires just a BSD license.

6.8.2 *Platforms used for assessment*

- **IBM BlueGene/P**
  At the University of Groningen a three-racks IBM BlueGene/P system has been installed. For the evaluation of Scalasca on the BlueGene two racks (8192 cores with a peak performance of 28 Tflop/s) have been used.
- **Scalasca** V1.0 released at June 18 2008 by Forschungszentrum Jülich (open-source - BSD license).
- **Linpack**
  The open-source package Linpack is a numerical solver for a dense system of linear equations. It is accepted world-wide as a benchmark and as such part of this evaluation. The problem size can be chosen such that performance (flop/s) is best, e.g. *n = 337919* for half a rack (2048 cores).

6.8.3 *Scalasca Details*

Scalasca is a toolset that can be used to analyse the performance behavior of parallel applications and to identify opportunities for optimisation. Scalasca supports an incremental performance-analysis procedure that integrates runtime summaries with in-depth studies of concurrent behavior via event tracing, adopting a strategy of successively refined measurement configurations. A distinctive feature is the ability to identify wait states that occur, for example, as a result of unevenly distributed workloads. Especially when trying to scale communication-intensive applications to large processor counts, such wait states can present severe challenges to achieving good performance.

Design goals

- Scalasca has been designed for use on large-scale systems.
- Easy identification of wait states.
- Support of OpenMP, MPI and hybrid.
- Supported languages: Fortran, C and C++.
- Easy generation summary reports with performance metrics for function call paths.
- Traces record individual run time events.

Usage
The basic use of Scalasca is as follows. First the source code is instrumented with additional wrapper calls around each function call, including MPI and/or OpenMP calls. Next, the

instrumented source file is compiled with the standard compilers available on the system. The tool to create instrumented source files is called *skin.* Apart from automatic instrumentation by skin it is also possible to do manual instrumentation by means of an API. Instrumented executables are run in exactly the same way as 'normal' executables, however during execution they generate trace files and files with statistical metrics. Some of the metrics are:

- number of visits of a function,
- MPI statistics like synchronisations, communications and bytes transferred,
- elapsed time, and,
- optionally, hardware metrics (platform dependent).

The tool to analyse these files is called *scan.* Scan produces so-called cube-files which can be examined with a tool called *cube3*.

Cube3 is a GUI which allows visual examination of the collected metrics. There are three coupled tree browsers. Each node in the tree displays the severity of a bottleneck in color (for easy identification) as well as in value (for precise comparison). Each node can be expanded into subnodes to achieve a more detailed view. On each node one can right-click to get detailed information, such as the line number of the location in the source code. Figure 10 shows an impression of the GUI. In this example the receive behavior of the function HPL_reduce is examined.

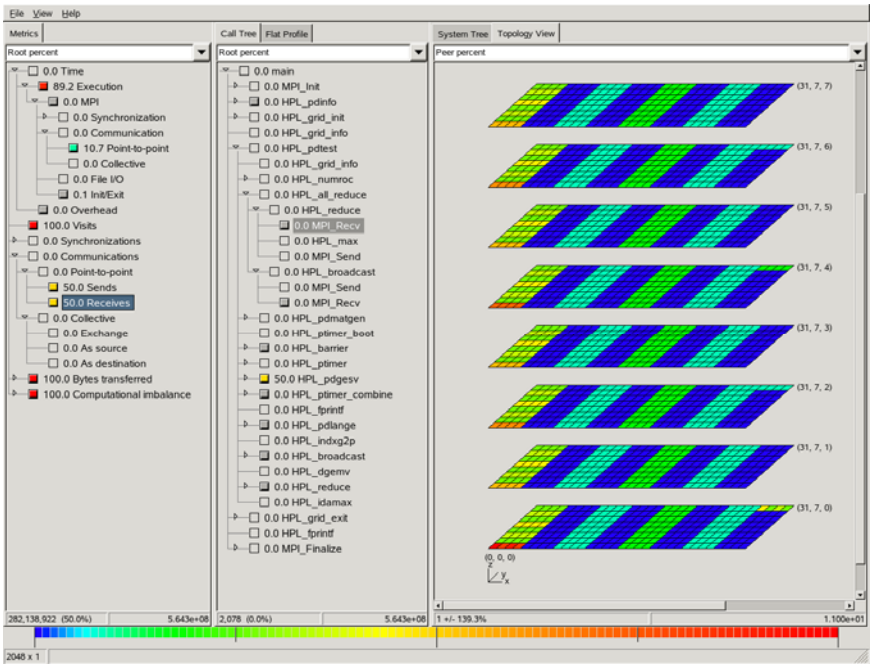

Figure 10: Screenshot of cube3 running Linpack on a 2048 core partition of the IBM BG/P.

Evaluation

This evaluation is based on the assessment of Scalasca on an IBM BlueGene/P system, located at the University of Groningen in the Netherlands, running Linpack HPL. Two racks (8192 cores with a peak performance of 28 Tflop/s) have been used.

- The program cube3 greatly provides immediate insight in hot spots of the code.

- Trace analysis is based on parallel replay: this enables scalability to tens of thousands of cores, however this is only suited for relatively short execution times. For long execution times the size of the trace files explodes and the analysis takes a long time
- Tracing offers critical insight into the temporal behaviour of a program.
- Scan automatically detects patterns of inefficient behaviour such as *late senders*. This is less error prone than manual inspection since it is guaranteed to cover the entire trace of the execution.
- From an assessment with linpack it appears that the overhead introduced by Scalasca is negligible.
- Scalability up to tens of thousands of cores appears to be possible however one may wonder whether it is acceptable to use as many cores for the analysis as for the actual execution of the program.

6.8.4 *Future Work*

The overall assessment is that Scalasca is a useful instrument to analyse the behaviour of massively parallel programs on large scale architectures.

Areas of improvement are:

- Update of documentation. Currently only a quick reference appears to be available.
- Installation procedure is poorly documented: the prerequisites were only partly addressed.
- For the visual inspection of trace files additional software (like Vampir) needs to be installed. As opposed to Scalasca his program, however, is not freely available.

6.9 Vampir VNG

6.9.1 *Introduction*

Vampir 5.0 [32] is a front end for displaying trace files. The trace files are written using the Open Trace Format (OTF) and are obtained when an application is instrumented using the libraries and wrappers provided by the VampirTrace library. While Vampir VNG is a commercial application, the VampirTrace library as well as the OTF format is available under BSD license. Vampir is marketed by “Gesellschaft für Wissens- und Technologietransfer der TU Dresden GmbH”.

Vampir is supported on Linux (IA32, x86_64, IA64, PPC/32, PPC/64), Sun Solaris (SPARC/32, SPARC/64, x86_64), IBM AIX (PPC), SGI IRIX (MIPS), Mac OS X. Soon MS-Windows.

6.9.2 *Vampir Details*

The instrumentation supports MPI and OpenMP, alone or in hybrid mode. It also allows manual, automatic and binary instrumentation, in case the source code is not available. The

instrumentation can trace not only MPI events, but also hardware counters (using the PAPI interface), memory allocation and I/O events.

Vampir can be used as stand-alone application or as server-client application. Small trace files of some MB are recommended for stand-alone mode. However, larger trace files (several GB) need to be visualized using the server-client implementation. The server application distributes and processes the information on the host machine by using MPI while the client connects to it and is used for visualization.

Usage

Although Vampir offers several options, the standard instrumentation produces most information required. Once the code is compiled and linked against the VampirTrace library the application is executed as usual. One can, through environmental variables, modify the size of the output, the level of instrumentation, the hardware counters or memory usage among other options. This way tracing with VampirTrace is a very versatile operation.

The Parallel Linpack benchmark (HPL) was used to test Vampir capabilities. We run a 64 MPI tasks case with full instrumentation of functions and MPI-calls, memory tracing and several hardware counters. This produced 30GB of compressed information. A timeline view example is shown in figure 11.

Vampir was designed for scalability and it was tested here as server-client application, as such for this test case was necessary to use a 128 MPI-task job with VampirServer to process this amount of information. Its interface offers several options and is very intuitive. The user can obtain a general view of how the application evolved. It can also trace single processes giving detailed information about specific function calls, MPI calls or hardware counters. An example is given in figure 12. In this example, a time-line view (zoom 1.92ms, call tree depth of level 8 (main has level 1), function name identification, with information of mem. allocation, L2 misses and load instructions) for process 7 is shown.

The resolution can capture different events up to ms, identify functions symbols or define the depth of the call tree accurately. The user has the choice of compacting all this information in suitable tables or report for further post processing.

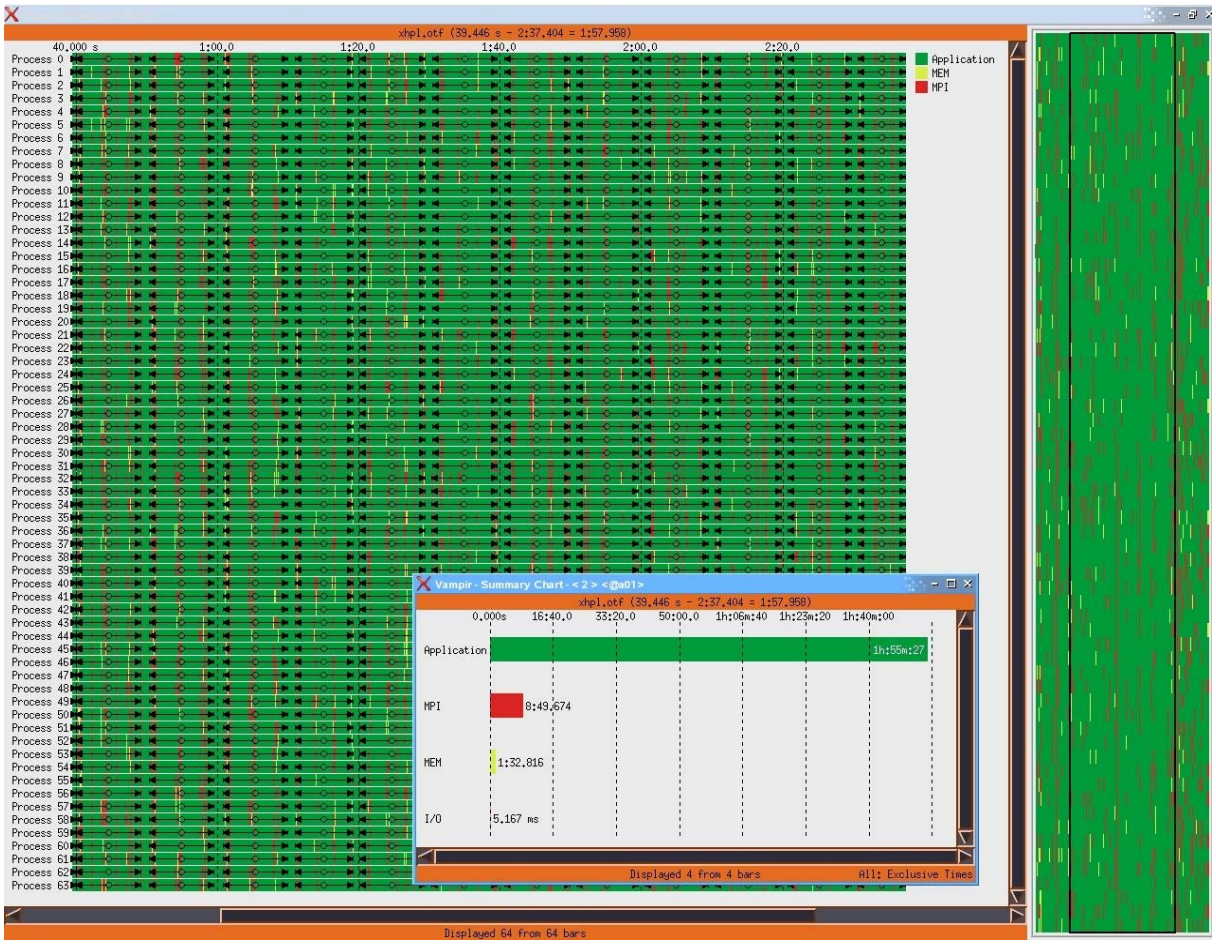

Figure 11: Timeline view of 64 MPI-tasks in HPL benchmark, VampirServer uses 128 MPI-Tasks.

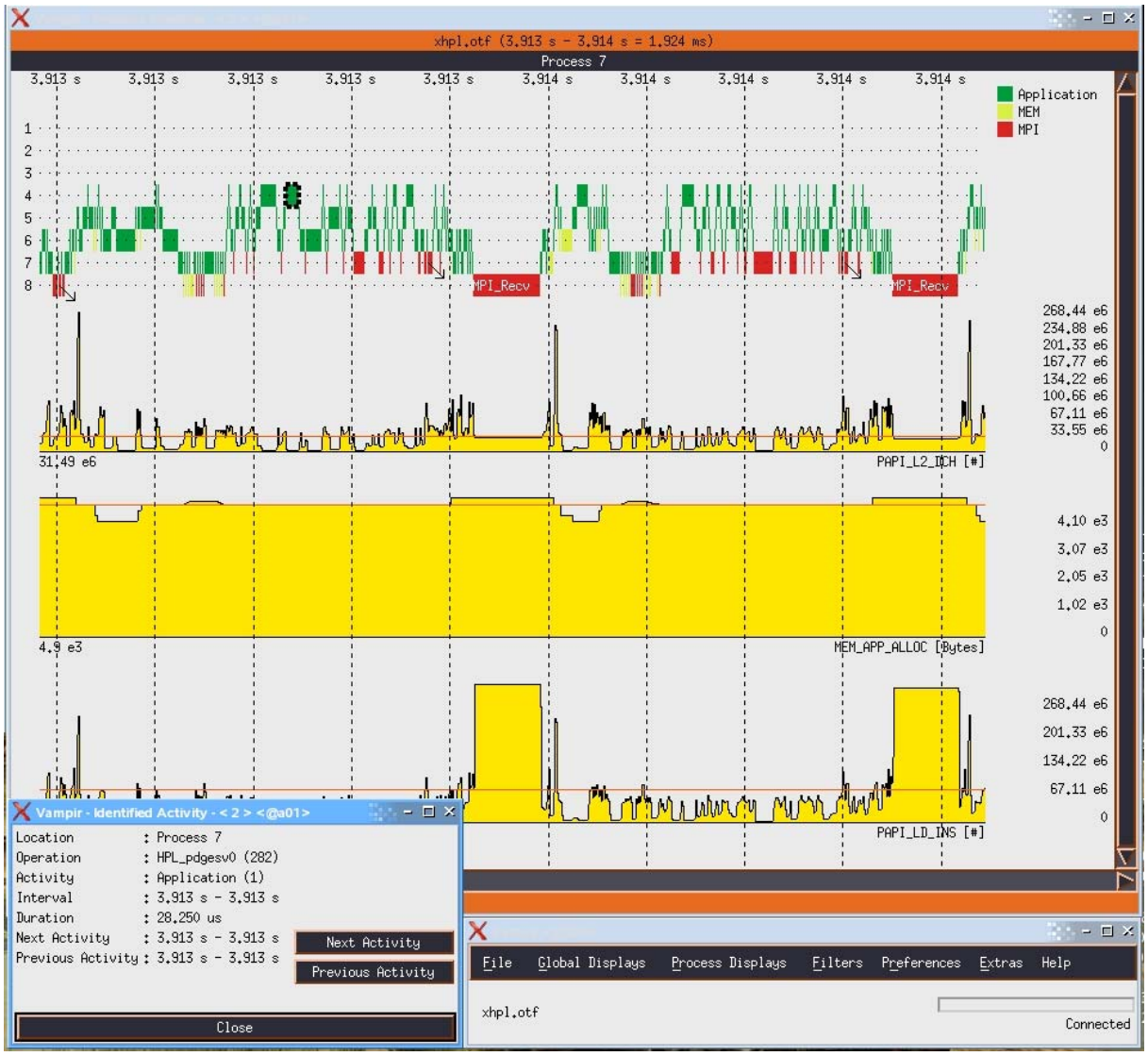

Figure 12: Example of individual process timeline view.

Future work

The tracing process is in parallel, but the unification of results into suitable files for Vampir is still a serial operation. In future work, this characteristic may have to be improved upon. Also, some features present in the stand-alone version are to be implemented on the client interface, like source code navigation. An MS-Window version may be released.

6.10 VPA: Visual Performance Analyzer

6.10.1 *Introduction*

Visual Performance Analyzer (VPA) is an Eclipse-based performance visualization toolkit. It is being developed by IBM, and is supported on Linux- x86, AIX-PPC, Cell and MS-Windows (32 bit). See also [33] for more information and users'guides.

6.10.2 *VPA Details*

Visual Performance Analyzer (VPA) is an Eclipse-based performance visualization toolkit. It consists of six major components: Profile Analyzer, Code Analyzer, Pipeline Analyzer, Counter Analyzer, Trace Analyzer, and Control Flow Analyzer. VPA is a collection of performance data analysis tools that can be used to identify performance bottlenecks. VPA does not supply performance data collection tools. Instead, it relies on platform specific tools, such as AIX Tprof, to collect the performance data.

- **Profile Analyzer** provides a powerful set of graphical and text-based views that allow users to narrow down performance problems to a particular process, thread, module, symbol, offset, instruction, or source line. Profile Analyzer supports time-based system profiles (Tprofs) collected from a number of IBM® platforms and the Linux® profile tool oprofile 0.9.3.
- **Code Analyzer** examines executable files and displays detailed information about functions, basic blocks, and assembly instructions. It is built on top of FDPR-Pro (Feedback Directed Program Restructuring) technology and allows adding of FDPR-Pro and Tprof profile information. (The Linux version of FDPR-Pro is available here at alphaWorks.) Code Analyzer is able to show statistics; navigate disassembled instructions; and display performance comments, instruction grouping information, and map instructions back to source code.
- **Pipeline Analyzer** is a port of the IBM Performance Simulator for Linux on POWER™, another alphaWorks technology. Pipeline joins the VPA toolkit to provide VPA users with the means of examining how code is executed on various IBM POWER processors. Pipeline Analyzer displays the pipeline execution of instruction traces generated by a POWER series processor. It does so by providing a scroll view and a resource view of the instruction execution.
- **Counter Analyzer** accepts hardware performance data from collection tools such as CPC or HPMCOUNT. The data is provided as XML and is parsed by this plug-in in order to allow visualizing and analysis through CPI breakdown models. The data can be saved in the embedded database for later viewing, or it can be exported to a CSV file for inclusion in a spreadsheet.
- **Trace Analyzer** visualizes Cell Broadband Engine™ traces containing information such as DMA communication, locking and unlocking activities, mailbox messages, etc. Trace Analyzer shows this data organized by core along a common timeline. Extra details are available for each kind of event: for example, lock identifier for lock operations, accessed address for DMA transfers, etc.
- **Control Flow Analyzer** is a tool that analyses call trace data collected by tools such as Jprof, which is part of Performance Inspector. The call trace data contains information about each method call, such as how much time is spent in every invocation and who calls

whom. Control Flow Analyzer provides two major ways of visualizing the call trace data: a graph of the execution flow and and a set of tables displaying the calling tree.

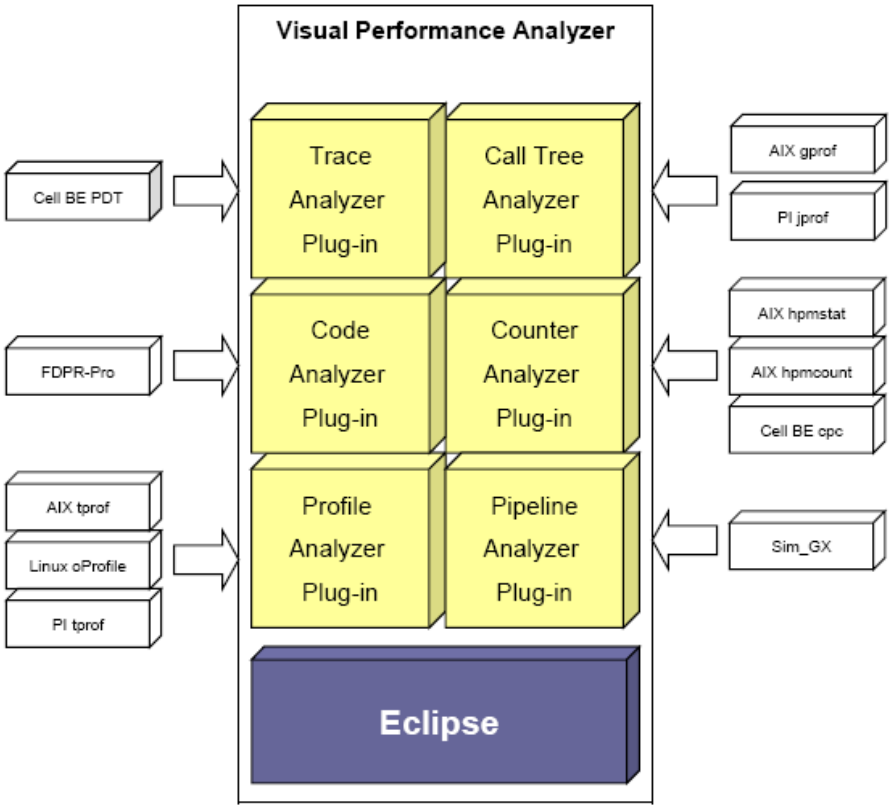

Figure 13: System architecture of Visual Performance Analyzer.

How does it work?

Profile Analyzer parses system profiles into an internal profiling data model that supports the profile hierarchy, offset locations, tick counts, CPU counter data, source line information, and disassembly. The plug-in then displays this data model, using various Eclipse views. The system profiles are those produced by Performance Inspector, AIX Tprof, and Linux oprofile. However, Visual Performance Analyzer can be extended to support almost any platform by converting a system profile to an XML schema that it understands.

Why the tool is interesting?

VPA can be very usefull when a programmer is trying to understand the complexity and performance of the particular code on the Cell/B.E. architecture. There are not many tools which enable controlling the PPU and SPU objects together with the DMA transfers between Main Memory and SPU Local Stores. VPA can be very useful for achieving powerful performance on Cell processors while it handles all the above functionalities and many more. Moreover VPA can be used remotly with a thin Java client working on a client machine and collecting the profile data from remote server.

Scalability

For practical reasons, our first interest will be focused on the QS22 18 cores performance. Next, scalability issues accross Cell blades will be considered, as they become available in the prototype system.

6.11 Considerations for Future Work

The performance tools, as considered in the previous subsections, are one way or another based on traces. Trace based systems provide a way to perform very detailed analyses. Proper views can convey a lot of information about the behavior of the application. Future work will have the intent to focus more and more on the usability of these tools for analysis of scalability. Depending on the results, feedback into the actual developers of the tools may be appropriate. In that respect, as a summary, the following practicalities may need to be considered for future work:

- The time to generate a sufficiently small trace with as much detail as possible should be minimized. This refers especially to the analyst time, but also if possible to the batch processing time.
- A minimal set of configuration files that would let a user identify the major performance issues.
- Automatic analysis functionalities. We would greatly appreciate some type of expert system that would automatically obtain metrics and models describing at a very high level the behaviour of the application.
- It would be ideal if such automatic analysis could be performed online without even requiring to generate a trace (unless specifically required by the analyst).
- File handling transparency: the original traces are generated at the parallel machine and they require a level of storage not available on many laptops. The analysis is nevertheless more conveniently done on a local laptop. It would thus be desirable to run the GUI locally and have it automatically access or remotely manipulate the large files at the parallel machine. It should be possible to operate on them as if they were local, remotely perform the filtering and cutting functionalities and automatically transferring the summarized files to the laptop for a more convenient analysis.
- The prototype architectures for PRACE in WP7 are known, and ready to be installed. Therefore, it will make sense to map the available tools on the available prototype architectures, just as done with the benchmark codes in chapter 3.

7 Annex

7.1 Benchmark Report Template

In order to prepare a uniform way of reporting benchmark progress, we have used a so-called benchmark report template, which covers details of the benchmark code, including porting details on the assigned hardware architectures. This enables some statistical analysis, as done in section 3.3, but also serves as a starting point for future work in tasks 5.4, 6.4 and 6.5.

Benchmark reporting template for usage in PRACE WP6, task 3, for porting progress with benchmark codes on hardware architectures.

GENERAL	
Name of Code, Abbreviation	
Application area(s)	e.g. materials science, astronomy, …
Key numerical method(s)	e.g. fft's, sparse solvers, dense matrices, ….
Origin (developers, institute)	
Current developers	
Contactperson	
License policy	
Copyright	
Usage rules (within PRACE, outside PRACE, …)	e.g. free within PRACE, not outside PRACE, …
PRACE INFORMATION	
BCO: name, email, institute	
Contributors (PRACE partners)	
Targeted hardware platforms as in BCO list	Choice of: MPP-BG, MPP-Cray, SMP-TN-x86, SMP-FN-Pwr6, SMP-FN+Cell, SMP-TN-Vector
CODE STATISTICS	
Programming language(s)	
Amount of source lines	
Libraries	e.g. LAPACK, NAG, FFTW, vendor libraries, HDF, ..
Parallellization method	e.g. MPI, OpenMP, pthreads, hybrid, SHMEM, single-sided MPI, …
Development platform(s)	
IO characteristics	e.g. none, read at start, write at end, each iteration, size, MPI IO, HDF, …
PORTING REPORT	For platform #1 (repeat for platform #2, #3, …)
Porting platform	e.g. MPP-BG, MPP-Cray, ……
Details porting platform	Hardware details, software details (OS version, compiler versions, libraries, …)
Overall porting result	Successful/not successful/partly successful
General comments	… free format …

Porting report on programming language constructs in general	… free format …
Porting report on libraries used	… free format …
Porting report on parallelisation method	… free format …
Porting report on IO	… free format …
PERFORMANCE RESULTS	For platform #1 (repeat for platform #2, #3, …)
Execution platform	e.g. MPP-BG, MPP-Cray, ……
Details execution platform	If different from porting platform
Performance details	Name of input set, #cores, speed-up results, initial performance profile, …
RECOMMENDATIONS	For Petascaling and optimisation on platform #1 (repeat for platform #2, #3, …)
Expected potential for Petascaling	Large/medium/small
Expected effort to reach Petascaling potential	Amount of pm's, what should be done to the code, …
Expected potential for Optimisation	Large/medium/small
Expected effort to reach Optimisation potential	Amount of pm's, what should be done to the code, …

7.2 Benchmark Porting Details

This section contains the actual filled benchmark report templates for each of the benchmark codes, as defined in section 3.2. Obviously, this is work in progress, and therefore should be recognised as a snapshot of the current situation.

7.2.1 *QCD*

Lukas Arnold
FZJ

GENERAL	
Name of Code, Abbreviation	multi-kernel lattice QCD benchmark, QCD
Application area(s)	particle physics
Key numerical method(s)	multiple
Origin (developers, institute)	multiple, see README in PABS
Current developers	none
Contactperson	multiple, see README in PABS
License policy	to be clarified
Copyright	to be clarified
Usage rules (within PRACE, outside PRACE, …)	free
PRACE INFORMATION	
BCO: name, email, institute	Lukas Arnold

	l.arnold@fz-juelich.de FZJ
Contributors (PRACE partners)	EPCC, CSC
Targeted hardware platforms as in BCO list	MPP-BG/P, MPP-Cray, FatNode-Pwr6
CODE STATISTICS	
Programming language(s)	Fortran 90 and C
Amount of source lines	
Libraries	none
Parallellization method	MPI
Development platform(s)	various
IO characteristics	
PORTING REPORT	For huygens(SARA)
Porting platform	FatNode-Pwr6
Details porting platform	**Hardware:** – 104 SMP nodes with 32 SMT processors each (total 3328) – Processortype: Power6 4.7 GHz – Overall peak performance: 60 Teraflops – Main memory: 83 x 128 Gbytes + 18 x 256 Gbytes (aggregate 15.2 TB) – InfiniBand (MPI communication) – Disk capacity: 700 TBytes **Software:** – Operating system: Linux (SuSE) – Operating mode: interactiv and batch – Compiler versions: IBM AIX compiler (xlf 11.1; xlc 9.0)
Overall porting result	successful

General comments	The QCD benchmark is embedded in the PRACE benchmarking suite, which sets all compilation and execution parameter. The used (default) setting are:
	fortran:
	`mpxlf90_r -q64 -qtune=pwr6 -qarch=pwr6`
	C:
	`mpcc_r -q64 -qtune=pwr6 -qarch=pwr6`
	Further performance flags will be used as the kernel is ready for benchmarking.

Porting report on programming language constructs in general	Up to now (3 kernels), there have been no problems.
Porting report on libraries used	none used
Porting report on parallelization method	Using AIX MPI compiler wrapper.

Porting report on IO	no data output is produced
PERFORMANCE RESULTS	For huygens
Execution platform	FatNode-Pwr6
Details execution platform	same with the porting platform
Performance details	none available, yet
PORTING REPORT	For jugene(FZJ)
Porting platform	FatNode-Pwr6
Details porting platform	**Hardware:** – 16384 compute nodes with 4-way SMT processors (total 65536 processors) – Processortype: PowerPC 450 850 MHz – Overall peak performance: 223 Teraflops – Linpack: 167 Teraflops – Main memory: 2 Gbytes per node (aggregate 32 TB) – Three-dimensonal torus (compute nodes) – Global tree / Collective network (compute nodes, I/O nodes) – 10 Gigabit Ethernet (I/O) – Disk capacity for system data: 4.5 TBytes – Disk capacity for user data: 1.0 PBytes – Migration storage for user data: 1.5 PBytes **Software:** – Operating system: CNL – Operating mode: interactive and batch – Compiler versions: IBM AIX compiler (xlf 9.1/10.1/11.1/12.1; xlc 7.0/8.0/9.0/10.1)
Overall porting result	Successful

General comments	The QCD benchmark is embedded in the PRACE benchmarking suite, which sets all compilation and execution parameter. The used (default) setting are:
	fortran:
	`mpxlf90_r -q64 -qtune=450 -qarch=450`
	C:
	`mpcc_r -q64 -qtune=450 -qarch=450`
	Further performance flags will be used as the kernel is ready for

	benchmarking.
Porting report on programming language constructs in general	Up to now (3 kernels), there have been no problems.
Porting report on libraries used	none used
Porting report on parallelization method	Using AIX MPI compiler wrapper.
Porting report on IO	no data output is produced
PERFORMANCE RESULTS	For jugene
Execution platform	FatNode-Pwr6
Details execution platform	same with the porting platform
Performance details	none available, yet
PORTING REPORT	For louhi (CSC)
Porting platform	MPP-Cray
Details porting platform	**Hardware:** – CPU: 2.3 GHz AMD Opteron 64 bit quad-core processors – Number of nodes: 1012 computing nodes – Memory: 1 GB or 2 GB memory per core. – Interconnect: Seastar 2+ – Set up physically in 11 cabinets. – Architechture: MPP – I/O subsystem: Lustre, 70 TB **Software:** – Operating system: CNL – Operating mode: interactiv and batch – Compiler versions: PGI compiler version 7.2.4
Overall porting result	in process
General comments	
Porting report on programming language constructs in general	
Porting report on libraries used	
Porting report on parallelization method	
Porting report on IO	
PERFORMANCE RESULTS	For louhi
Execution platform	MPP-Cray
Details execution platform	same with the porting platform
Performance details	none available, yet
RECOMMENDATIONS	For Petascaling and optimization on all target platforms
Expected potential for Petascaling	High (all platforms)
Expected effort to reach Petascaling potential	All kernels should be able to scale to high number of processes.

	0PM
Expected potential for Optimization	Low
Expected effort to reach Optimization potential	All kernels are already optimized. There exist highly optimized versions, but they can not be used in PABS, due to the usage policies. The effort to get this high optimization is very high. 0PM

7.2.2 *VASP*

Miquel Català
BSC-CNS

GENERAL	
Name of Code, Abbreviation	Vienna Ab-initio Package Simulation (VASP)
Application area(s)	ab-initio quantum-mechanical molecular dynamics
Key numerical method(s)	
Origin (developers, institute)	Mike Payne, MIT
Current developers	Dr. Doris Vogtenhuber
Contact person	vasp.materialphysik@univie.ac.at
License policy	not public domain
Copyright	
Usage rules (within PRACE, outside PRACE, …)	Free within PRACE, not outside PRACE
PRACE INFORMATION	
BCO: name, email, institute	Miquel Català i Coït <miquel.catala@bsc.es> Barcelona Supercomputing Center
Contributors (PRACE partners)	NCF, GUP, HLRS
Targeted hardware platforms as in BCO list	MPP-BG, FatNode-Pwr6, Cell, Vector
CODE STATISTICS	
Programming language(s)	FORTRAN 90
Amount of source lines	121776 lines
Libraries	vasp.lib (included with application) + scalapack + lapack + blacs + pthreads + MPI
Parallellization method	MPI
Development platform(s)	Pentium II, III, IV and Athlon based PC's under LINUX and DEC Alpha (TRUE 64 and Linux)
IO characteristics	No input parameters. Needed files on current directory.
PORTING REPORT	
Porting platform	IBM Blue Gene/P
Details porting platform	16 Racks with 32 nodecards x 32 compute nodes (total 16384) • Compute node: 4-way SMP processor • Processortype: 32-bit PowerPC 450 core 850 MHz • Processors: 65536 • Overall peak performance: 223 Teraflops • Linpack: 167 Teraflops • Main memory: 2 Gbytes per node (aggregate 32 TB) • I/O Nodes: 152 • Networks: • Three-dimensonal torus (compute nodes)

	• Global tree / Collective network (compute nodes, I/O nodes) • 10 Gigabit ethernet / Functional network (I/O Nodes) • Power Consumption: max.40 kW per rack 2 Service nodes IBM p55A: • Total number of processors: 8 • Processortype: Power5 1.6 GHz • Total amount of memory: 32 GB • Operating System: SuSE Linux Enterprise (SLES 10) 2 Login nodes IBM p55A: – Total number of processors: 8 – Processortype: Power5 1.6 GHz – Total amount of memory: 32 GB – Operating System: SuSE Linux Enterprise (SLES 10) – Internet address: jugene.zam.kfa-juelich.de
Overall porting result	Successfully
General comments	
Porting report on programming language constructs in general	
Porting report on libraries used	
Porting report on parallelisation method	VASP is well parallelised. No specific development has to be done.
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	IBM Blue Gene/P (jugene.fz-juelich.de)
Details execution platform	This platforms allow executions in queue.
Performance details	The testcase achieves the best performance when 16 MPI tasks are launched. With more tasks performance drops.
RECOMMENDATIONS	
Expected potential for Petascaling	Good potential
Expected effort to reach Petascaling potential	Depends on BLAS and FFT performance
Expected potential for Optimisation	Good potential
Expected effort to reach Optimisation potential	Depends on BLAS and FFT performance

PORTING REPORT	
Porting platform	Power6
Details porting platform	IBM pSeries 575, a clustered SMP (Symmetric Multiprocessing) system. • 104 nodes • 16 dual core processors (IBM Power6, 4.7 GHz) per node • 128 GByte or 256 GByte of memory per node • 700 TByte of disk space • total peak performance is 60 Teraflop/sec • In total, the system has: • 1664 dual core processors = 3328 cores • 15.25 TByte of memory • 700 TByte of disk space

	An IBM Power6 processor has the following characteristics: – Dual core running on 4.7 GHz – L1 cache: 128 KByte of L1 cache per core (64 KByte data cache + 64 KByte instruction cache) – L2 cache: 4 MByte per core (semi shared: the cache is assigned a specific core, but the other has a fast access to it) – L3 cache: 32 MByte per processor The nodes are interconnected with an Infiniband network providing an MPI bandwidth of 160 Gbit/sec between neighboring nodes
Overall porting result	Successfully
General comments	
Porting report on programming language constructs in general	
Porting report on libraries used	
Porting report on parallelisation method	VASP is well parallelitzed. No specific development has to be done.
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	Power6 (huygens.sara.nl)
Details execution platform	Same above
Performance details	The testcase achieves the best performance when 32 MPI tasks are launched. With more tasks performance drops.
RECOMMENDATIONS	
Expected potential for Petascaling	Good potential
Expected effort to reach Petascaling potential	Depends on BLAS and FFT performance
Expected potential for Optimisation	Good potential
Expected effort to reach Optimisation potential	Depends on BLAS and FFT performance

7.2.3 *NAMD*

Dr. Joachim Hein
EPCC, The University of Edinburgh

GENERAL	
Name of Code, Abbreviation	NAMD
Code release	2.6
Application area(s)	(Bio)chemistry
Key numerical method(s)	Molecular dynamics
Origin (developers, institute)	K. Schulten, L. Kale, et. al., Beckman Institute, UIUC, US
Current developers	*as above*
Contactperson	J. Phillips (UIUC), NAMD team email: namd@ks.uiuc.edu
License policy	Own license, see: http://www.ks.uiuc.edu/Research/namd/license.html
Copyright	The Board of Trustees of the University of Illinois
Usage rules (within PRACE, outside PRACE, …)	At present: Download source from UIUC website
PRACE INFORMATION	
BCO: name, email, institute	Joachim Hein, j.hein@epcc.ed.ac.uk, EPCC, The University of

	Edinburgh
Contributors (PRACE partners)	Xu Guo (EPCC), Jon Hill (EPCC), Martin Polak (GUP), CSC, GRNET
Targeted hardware platforms as in BCO list	MPP-BG, MPP-Cray, FatNode-Pwr6 and maybe Cell
CODE STATISTICS	
Programming language(s)	C++ (using Charm++)
Amount of source lines	62400 in .C-files and 17300 in .h-files plus Charm++ source
Libraries	Charm++, FFTW, TCL
Parallellization method	Charm++, which in most cases is build ontop of MPI, but could also be build ontop of e.g. Myrinet, Infiniband, Shmem, etc
Development platform(s)	
IO characteristics	appears to be master/worker model (nothing fancy)
PORTING REPORT	For platform #1 (repeat for platform #2, #3, …)
Porting platform	MPP-Cray
Details porting platform	HECToR (Cray XT4), dual core 2.8 GHz Opteron nodes, CLE (Cray Linux Environment, aka CNL)
Overall porting result	Successful (production executable)
General comments	
Porting report on programming language constructs in general	Build using gcc The CRAY XT using CNL is newer than code release, hence unsupported in NAMD2.6 Architecture files for charm++ and NAMD developed during porting
Porting report on libraries used	For Cray XT, later version of Charm++ than 5.9, which is bundled with the NAMD source distribution is needed. We used the nightly build version 03 July 2007. To build NAMD with gcc, FFTW2 build with gcc (only provided for the PGI compiler on the service) is needed TCL library needs building under gcc (not provided as part of service)
Porting report on parallelisation method	Build charm++ using MPI
Porting report on IO	Test benchmark didn't stress I/O
PORTING REPORT	For platform #2 (repeat for platform #2, #3, …)
Porting platform	FatNode-Pwr5
Details porting platform	HPCx, UK IBM Pwr5 with AIX
Overall porting result	Successful (production executable)
General comments	
Porting report on programming language constructs in general	Build using IBM's xlc 9.0, 32-bit addressing with large file support (>2GB) Changing the architecture specific flags: -qarch=pwr4 -qtune=pwr4 (from com and pwr3 respectively) yields better performance
Porting report on libraries used	Build using a later version of charm++ than the one bundled with the NAMD 2.6 source. Using nightly build 28 Nov 2005 (since release version 5.9 would not work on AIX 5.3). FFTW2 available on HPCx service Used TCL library which comes as part of the NAMD 2.6 distribution
Porting report on parallelisation method	Build charm++ using MPI
Porting report on IO	The benchmarks used up till now do not stress I/O
PORTING REPORT	For platform #3 (repeat for platform #2, #3, …)
Porting platform	FatNode-Pwr6
Details porting platform	Huygens @ SARA, NL Pwr6 with Linux OS (not AIX)
Overall porting result	Several successful executables build, exploring further compiler optimizations desirable

General comments	Fat node IBM Power architecture with Linux not (yet?) supported by NAMD developers (AIX is well supported). Received support from SARA who made their configurations available
Porting report on programming language constructs in general	Optained executables using IBM's xlc 9 and xlc 10 compilers. User 64bit addressing, hence large files (>2GB) should be naturally supported.
Porting report on libraries used	The charm++ version supplied with the NAMD 2.6 source works on the architecture. Though since the architecture is not supported in the version of charm++ architecture configuration files are needed. The support team of the service has provided such files which build a charm++ which passes the tests. Used the FFTW2 and TCL libraries which are provided as part of the service
Porting report on parallelisation method	Build charm++ using MPI
Porting report on IO	The benchmarks used up till now do not stress I/O
PORTING REPORT	For platform #4
Porting platform	MPP-BG
Details porting platform	JuGene @ FZJ in Germany IBM Blue Gene P system
Overall porting result	Successful NAMD executable
General comments	Code had already been built on site before, we used build procedures and executables supplied by FZJ.
Porting report on programming language constructs in general	Built using IBM bgxlC_r V9.0 compiler together with the architecture files provided by FZJ and optimization options "-O3 -qhot -arch=450d -qtune=450" which should already deliver a decent performance and uses SIMD features of the BG's compute nodes.
Porting report on libraries used	The charm++ version supplied with the NAMD 2.6 source is not supported on this architecture, built using charm-6.0 sources from its official download site using a build script from FJZ with -qarch=450d (simd). Using fftw-2.1.5 which is part of the service, but can also be also built using different optimisation and different precision switches using the FJZ build scripts. Using tcl-8.4.19 not available on site, which can be built using supplied build-scripts: needs the "bgxlC_r" compiler and patching of the configure scripts
Porting report on parallelisation method	Build charm++ using MPI
Porting report on IO	The benchmarks used up till now do not stress I/O
PORTING REPORT	For platform #5
Porting platform	MPP-CRAY
Details porting platform	Louhi @ CSC in Finland Cray XT4 and XT5, quad core opteron nodes
Overall porting result	Successful
General comments	Need access to larger partitions of the machine to properly assess e.g. XT4 vs XT5 differences on scalability
Porting report on programming language constructs in general	Build using gcc The CRAY XT using CNL is newer than code release, hence unsupported in NAMD2.6 Architecture files for charm++ and NAMD developed during porting
Porting report on libraries used	For Cray XT, later version of Charm++ than 5.9, which is bundled with the NAMD source distribution is needed. We used the nightly build version 03 July 2007. To build NAMD with gcc, FFTW2 build with gcc (only provided for the PGI compiler on the service) is needed TCL library needs building under gcc (not provided as part of service)
Porting report on parallelisation method	Build charm++ using MPI

Porting report on IO	Test benchmark didn't stress I/O
PERFORMANCE RESULTS	For platform #1
Execution platform	MPP-Cray
Details execution platform	HECToR
Performance details	• Initial performance results part of D6.2.2 • Managed to run benchmarks with 1 and 2 million atoms but (so far) failed to get a 9 million atom benchmark going. 2 million atom benchmark shows good scalability on up to 8192 cores. This dataset is not expected to be large enough for the code @ Petascale • We have a 9 million atom benchmark system. So far no success in getting this running.
PERFORMANCE RESULTS	For platform #2
Execution platform	FatNode-Pwr5
Details execution platform	HPCx
Performance details	• Code benefits from SMT (simultaneous multi threading) • Initial performance results part of D6.2.2
PERFORMANCE RESULTS	For platform #3
Execution platform	FatNode-Pwr6
Details execution platform	Huygens @ SARA
Performance details	• Initial tests show the code can benefit greatly from SMT (simultaneous multi threading), potentially greater benefit than Pwr5 system. Though SMT runs are extremely noise (wide fluctuations in runtime between repeat runs). When using large number of nodes the benefit is completely lost (in contrast to the experience we have with the Pwr5 system) • Managed to run benchmarks with 1 and 2 million atoms but (so far) failed to get a 9 million atom benchmark going. 2 million atom benchmark shows good scalability on up to 1024 cores. For 2048 cores the performance drops dramatically. This data set is not expected to be large enough for the code @ Petascale • We have a 9 million atom benchmark system. So far no success in getting this running.
PERFORMANCE RESULTS	For platform #4
Execution platform	MPP-BG
Details execution platform	JuGene @ FZJ
Performance details	• Successful running of the 1 and 2 million atom benchmarks • Very good scalability up to 8192 cores • Due to memory limitations (2GB per quad core processor installed) the 1 million atom benchmark can use at most 2 compute task on a quad core processor (-mode=dual), the 2 million atom benchmark can only place 1 task per quad core processor (-mode=smp)
PERFORMANCE RESULTS	For platform #5
Execution platform	MPP-Cray
Details execution platform	Louhi @ CSC
Performance details	• Managed to get the 1 and 2 Million atom benchmarks running. • Getting the 2 Million atom benchmark running requires fine tuning of the buffer space assigned to the MPI library. The small 1 GB/core of memory is clearly restrictive here. • Initial performance assessment shows the machine to be comparable in performance to HECToR, if the difference in clock rates is considered

RECOMMENDATIONS	For Petascaling and optimisation on platform #1 (HECToR)
Expected potential for Petascaling	**Medium** reached, further improvement might be possible
Expected effort to reach Petascaling potential	*Not a PRACE prototype but related to the Louhi system* • Using a 2 Million atom benchmark we can use 4096 cores with better than 50% efficiency (related to 128 cores) • For a Peta-scale system using this (or similar e.g. Louhi) technology a larger benchmark is needed • So far no success in using the larger 9 Million atom benchmark available to us o Might be a matter of not having found the right environment settings o Might be a shortcoming in the architecture of the application, e.g. due to Master/Worker components o Might be caused by memory consumption o Needs further investigation • NAMD 2.6 is several years old. The problems might have been addressed in a more recent (development version), we will try to liaise with the developers to clarify • If such work is to be carried out under PRACE effort levels can't be determined at this point. Anticipated effort: 3 person month, shared on all prototypes • Investigate feasibility of converting NAMD into a hybrid (MPI + OpenMP) which could be more efficient if the memory is a main obstacle to run peta-scale Benchmark. • This should benefit all multi-core systems Anticipated effort: 7 person month, shared on all prototypes
Expected potential for Optimisation	Low
Expected effort to reach Optimisation potential	
RECOMMENDATIONS	For Petascaling and optimisation on platform #2 (HPCx)
Expected potential for Petascaling	*Not a PRACE prototype*
Expected effort to reach Petascaling potential	*Not a PRACE prototype*
Expected potential for Optimisation	*Not a PRACE prototype*
Expected effort to reach Optimisation potential	*Not a PRACE prototype*
RECOMMENDATIONS	For Petascaling and optimisation on platform #3 (Huygens)
Expected potential for Petascaling	Medium or better
Expected effort to reach Petascaling potential	• Using a 2 Million atom benchmark we can use 1024 cores with 88% efficiency (related to 128 cores) • The same benchmark (2 M atom) shows extremely poor performance when used on 2048 cores. Since this performs reasonable on the Cray XT4 and BlueGene/P up to 8192 cores, this appears a machine specific issue. The reasons need understanding and fixing for the architecture underlying Huygens to be a viable candidate for a Peta-scale system. • For a Peta-scale system using this technology a larger benchmark is needed. So far no success in using the larger 9 Million atom benchmark available to us o Might be a matter of not having found the right environment settings o Might be a shortcoming in the architecture of the application, e.g. due to Master/Worker components o Might be caused by memory consumption o Needs further investigation

	• NAMD 2.6 is several years old. The problems might have been addressed in a more recent (development version), we will try to liaise with the developers to clarify • If such work is to be carried out under PRACE effort levels can't be determined at this point. Anticipated effort: 3 person month, shared on all prototypes • Investigate feasibility of converting NAMD into a hybrid (MPI + OpenMP) which could be more efficient if the memory is a main obstacle to run peta-scale benchmark. Anticipated effort: 7 person month, shared on all prototypes
Expected potential for Optimisation	Low
Expected effort to reach Optimisation potential	• Initial performance assessment shows the application to benefit substantially from SMT (simultaneous multi threading). However: o These runs suffer from bad noise o Benefits are lost at large task count • Tuning of poe (IBM's MPI library) or the machine might overcome the issue.
RECOMMENDATIONS	For Petascaling and optimisation on platform #4 (JuGene)
Expected potential for Petascaling	**Medium** reached, further improvement might be possible
Expected effort to reach Petascaling potential	• For a Peta-scale system using this technology a larger benchmark is needed. So far no success in using the larger 9 Million atom benchmark available to us o Might be a matter of not having found the right environment settings o Might be a shortcoming in the architecture of the application, e.g. due to Master/Worker components o Might be caused by memory consumption o Needs further investigation Anticipated effort: 3 person month, shared on all prototypes • Due to memory demands for the large multi-million atom benchmarks, the quad core processors have to be under-populated (fewer than 4 compute tasks per processor). This leaves computing power unused. Converting into a hybrid (MPI + OpenMP) could provide a possible mitigation strategy Anticipated effort: 7 person month, shared on all prototypes
Expected potential for Optimisation	Low
Expected effort to reach Optimisation potential	
RECOMMENDATIONS	For Petascaling and optimisation on platform #5 (Louhi)
Expected potential for Petascaling	Present system to small for assessment. Based on HECToR experience**,** **Medium** should be easily achievable if a larger system was available, further improvement might be possible
Expected effort to reach Petascaling potential	• So far no success in using the larger 9 Million atom benchmark available to us o Might be a matter of not having found the right environment settings o Might be a shortcoming in the architecture of the application, e.g. due to Master/Worker components o Might be caused by memory consumption o Needs further investigation • NAMD 2.6 is several years old. The problems might have been addressed in a more recent (development version), we will try to liaise with the developers to clarify • If such work is to be carried out under PRACE effort levels can't be determined at this point. • *The comments made on HECToR should apply in a similar fashion, though experience with smaller benchmarks indicate*

	that scalability might be slightly worse, most likely due to higher core to network access point ratio. Needs further investigation. Anticipated effort: 3 person month, shared on all prototypes • Investigate feasibility of converting NAMD into a hybrid (MPI + OpenMP) which could be more efficient if the memory is a main obstacle to run peta-scale benchmark Anticipated effort: 7 person month, shared on all prototypes
Expected potential for Optimisation	Low
Expected effort to reach Optimisation potential	

7.2.4 *CPMD*

Albert Farrés
BSC-CNS

GENERAL	
Name of Code, Abbreviation	CPMD
Application area(s)	ab-initio molecular dynamics
Key numerical method(s)	fft's, dense matrices
Origin (developers, institute)	R. Car, International School for Advanced Studies, Trieste, Italy. M. Parrinello, Dipartimento di Fisica Teorica, Università di Trieste, Trieste, Italy, and International School for Advanced Studies, Trieste, Italy.
Current developers	CPMD consortium, coordinated by Prof. Michele Parrinello (Chair of Computational Science, ETH Zurich) and Dr. Wanda Andreoni (Program Manager of Deep Computing Applications at IBM Zurich Research Laboratory)
Contact person	Alessandro Curioni <cur@zurich.ibm.com>
License policy	http://www.cpmd.org/cpmd_licence.html
Copyright	IBM Corp. and Max Planck Institute, Stuttgart
Usage rules (within PRACE, outside PRACE, …)	Free within PRACE, not outside PRACE
PRACE INFORMATION	
BCO: name, email, institute	Albert Farres <albert.farres@bsc.es> BSC-CNS, (Spain)
Contributors (PRACE partners)	CSC, CINECA, SIGMA, HLRS
Targeted hardware platforms as in BCO list	MPP-BG, FatNode-Pwr6, Cell, Vector
CODE STATISTICS	
Programming language(s)	FORTRAN 77
Amount of source lines	~ 174047
Libraries	BLAS, LAPACK
Parallellization method	MPI
Development platform(s)	See source code. More than 100 different configurations for several platforms
IO characteristics	unknown
PORTING REPORT	JUGENE
Porting platform	IBM Blue Gene/P
Details porting platform	16 Racks with 32 nodecards x 32 compute nodes (total 16384) • Compute node: 4-way SMP processor • Processortype: 32-bit PowerPC 450 core 850 MHz • Processors: 65536

	• Overall peak performance: 223 Teraflops • Linpack: 167 Teraflops • Main memory: 2 Gbytes per node (aggregate 32 TB) • I/O Nodes: 152 • Networks: o Three-dimensonal torus (compute nodes) o Global tree / Collective network (compute nodes, I/O nodes) o 10 Gigabit ethernet / Functional network (I/O Nodes) • Power Consumption: max.40 kW per rack 2 Service nodes IBM p55A: • Total number of processors: 8 • Processortype: Power5 1.6 GHz • Total amount of memory: 32 GB • Operating System: SuSE Linux Enterprise (SLES 10) 2 Login nodes IBM p55A: – Total number of processors: 8 – Processortype: Power5 1.6 GHz – Total amount of memory: 32 GB – Operating System: SuSE Linux Enterprise (SLES 10) – Internet address: jugene.zam.kfa-juelich.de
Overall porting result	Successfully
General comments	
Porting report on programming language constructs in general	
Porting report on libraries used	All the libraries needed by CPMD are avalaible on JUGENE system.
Porting report on parallelisation method	CPMD is well parallelitzed. No specific development has to be done.
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	IBM Blue Gene/P
Details execution platform	Same above
Performance details	With 128 cores it goes 1.15 times faster than it does in Marenostrum. With 256 it goes 1.1 times faster.
RECOMMENDATIONS	
Expected potential for Petascaling	Optimal
Expected effort to reach Petascaling potential	
Expected potential for Optimisation	
Expected effort to reach Optimisation potential	
PORTING REPORT	SARA
Porting platform	Power6
Details porting platform	IBM pSeries 575, a clustered SMP (Symmetric Multiprocessing) system. – 104 nodes – 16 dual core processors (IBM Power6, 4.7 GHz) per node – 128 GByte or 256 GByte of memory per node – 700 TByte of disk space

	– total peak performance is 60 Teraflop/sec In total, the system has: – 1664 dual core processors = 3328 cores – 15.25 TByte of memory – 700 TByte of disk space An IBM Power6 processor has the following characteristics: • Dual core running on 4.7 GHz • L1 cache: 128 KByte of L1 cache per core (64 KByte data cache + 64 KByte instruction cache) • L2 cache: 4 MByte per core (semi shared: the cache is assigned a specific core, but the other has a fast access to it) • L3 cache: 32 MByte per processor The nodes are interconnected with an Infiniband network providing an MPI bandwidth of 160 Gbit/sec between neighboring nodes
Overall porting result	Successfully
General comments	
Porting report on programming language constructs in general	
Porting report on libraries used	All the libraries needed by CPMD are avalaible on SARA system
Porting report on parallelisation method	CPMD is well parallelitzed. No specific development has to be done.
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	Power6
Details execution platform	Same above
Performance details	With 128 cores it goes 2.2 times faster than it does in Marenostrum. With 256 it goes 2.5 times faster.
RECOMMENDATIONS	
Expected potential for Petascaling	Optimal
Expected effort to reach Petascaling potential	
Expected potential for Optimisation	
Expected effort to reach Optimisation potential	

7.2.5 *Code_Saturne*

Andrew Sunderland, Charles Moulinec

STFC Daresbury Laboratory

GENERAL	

Name of Code, Abbreviation	Code_Saturne
Application area(s)	CFD, heat transfer, turbulence
Key numerical method(s)	Finite Volume & Sparse Linear Algebra
Origin (developers, institute)	EDF-R&D
Current developers	EDF-R&D
Contactperson	marc.sakiz@edf.fr, c.moulinec@stfc.ac.uk, a.g.sunderland@stfc.ac.uk
License policy	GPL
Copyright	EDF's copyright
Usage rules (within PRACE, outside PRACE, …)	Free
PRACE INFORMATION	
BCO: name, email, institute	Andrew Sunderland, andrew.sunderland@stfc.ac.uk, STFC Daresbury Laboratory, UK
Contributors (PRACE partners)	BSC Barcelona, HLRS Stuttgart, SARA
Targeted hardware platforms as in BCO list	BGP, Cray XT4/5, Pwr5, FatNode-Pwr6, Cell, Vector
CODE STATISTICS	
Programming language(s)	49% Fortran 77, 41% C99, 10% Python
Amount of source lines	500,000 lines

Libraries	BLAS, if activated
Parallellization method	MPI
Development platform(s)	Clusters, BG/L, BG/P.
IO characteristics	read at start, write at end, every x iteration if required
PORTING REPORT	
Porting platform	**MPP-BG/P (at STFC)**

Details porting platform	Hardware details:
	• Model: IBM Blue Gene / P
	• Proc Type: PowerPC 450 850 MHz
	  o double precision, dual pipe floating point acceleration on each core (3.4 GFlops)
	• Clock rate: 850 MHz
	• Total Cores: 4096
	• Cores Per Chip: 4
	• Chips per Compute Card (Node): 1
	• Memory per core: 512MB.
	• Total Memory: 2048 Gbytes
	• Caches:
	  o Private 32 KB per core L1 cache
	  o Private 14 stream prefetching per core L2 cache
	  o Shared 8MB L3 cache
	• Interconnect: Proprietary 3D Torus
	• 32 I/O nodes ~4 Tbytes disk
	
	Software Details
	• OS Version: Linux 2.6.16.46-0.12-ppc64
	• Fortran Compilers:
	  o IBM XLF v11.1
	  o GNU 4.1.2
	• C Compilers:
	  o IBM XLC v9.0
	  o GNU GCC v4.1.2
	• Libraries: Essl, Blas

Overall porting result	Successful
General comments	Porting to BG/P was carried out by cross-compilation. Compilation flags used are -O3 -qarch=450d -qtune=450d No special settings are required in the batch scripts. The 3 modes, SMP, CO, VN can be used.
Porting report on programming language constructs in general	Fully compliant with XLF & XLC compilers
Porting report on libraries used	No external libraries used
Porting report on parallelisation method	Use bgxlf & bgxlc to link all mpi routines & config files.
Porting report on IO	No special requirements
Porting Platform	**IBM p575 Server (HPCx at STFC)**
Details Porting Platform	Hardware: • Model: eServer 575 cluster • Proc Type: Power5 • Clock rate: 1.5 GHz • Total Cores: 2560 • Cores Per Chip: 2 • Cores Per Node: 16 • Memory per core: 2GB • Total Memory: 5120 Gbytes • Cache: Each core has a 32 Kbyte data cache and a 64 Kbyte instruction cache. The level 1 data cache has 128-byte lines, is 2-way set associative and write-through. The level 2 cache is on-chip, shared between the cores. It is a 1.9 Mbyte combined data and instruction cache, with 128 byte lines and is 10-way set associative and write-back. The level 3 cache is 36 Mbytes, off-chip and is shared between the 2 cores. It has 256 byte lines, and is 12-way set associative and write-back. • Interconnect: IBM High Performance Switch (HPS). Each eServer node has two network adapters and there are two links per adapter, making a total of four links between each of the frames and the switch network. • I/O: 72 Tbytes of disk running GPFS. Connected to computes nodes via HPS Software details: • OS version: AIX 5.3 • Compiler versions: IBM XL Fortran compiler 10.01.0000.0007
Overall Porting Result	Successful
General Comments	The PWR5 is an established platform and porting was straightforward.

	compilation flags used for initial port are: f90: *-q64 -O1* c: *-q64 -funsigned-char -qhot -qarch=pwr5* for performance optimisation suggested flags are: f90: -O3 -qarch=pwr5 -qtune=pwr5 c: *-q64 -funsigned-char -O3 -qhot -qarch=pwr5 -qtune=pwr5* No special settings are required in the batch scripts.
Porting report on programming language constructs in general	Code is fully compliant with XLF and XLC compilers.
Porting report on libraries used	None
Porting report on parallelisation method	Using mpixlf95_r and mpixlc_r compiler variants links mpi libraries and config files automatically
Porting report on IO	No special requirements
Porting Platform	**Cray XT4 (HECToR)**
Details Porting Platform	Model: Cray XT4 • Proc Type: AMD Opteron Dual Core • Clock rate: 2.8 GHz • Total Cores: 11328 • Cores Per Chip: 2 • Cores Per Node: 2 • Memory per core: 3GB. • Total Memory: 33984 Gbytes • Cache: Separate level 1 caches for data and instructions of 64 kB each. The L1 data cache is 2-way set associative. There is a combined data and instruction L2 cache of 1 MB for each core, which is 16-way associative. The L1 data and the L2 cache use 64 byte cache lines. The L2 cache acts as a victim cache for the L1 cache. Data evicted from the L1 cache gets established on the L2 cache. • Interconnect: Cray SeaStar2 3D torus • I/O: 12 I/O nodes connected to 576 TB of RAID disks running Lustre Software details: • OS version: UNICOS/lc version 2.0.53 • Compiler versions: PGI compilers, version 7.1.4
General Comments	Porting was not problematic for the XT4 Compilation using the GNU compiler: module swap PrgEnv-pgi PrgEnv-gnu cc -O or -O2 depending on the source file ftn -O1, -O2, -O3
Porting report on programming language constructs in general	Code is fully compliant with GNU compiler. Currently testing with PGI and Pathscale
Porting report on libraries used	None
Porting report on parallelisation method	Use of ftn command (pathscale) links in mpi libraries and

	config files automatically
Porting report on IO	No special requirements
Porting Platform	**NEC SX-8 Cluster (HLRS)**
Details Porting Platform	Model: NEC SX-8 Distributed-memory multi-vector processor • Proc Type: dedicated vector CPUs • Clock rate: 2 GHz • Total Cores: 72 * 8 CPUs • Cores Per Chip: N/A • Cores Per Node: N/A • Total Memory: 9.2 TB. • Cache: None • Interconnect: IXZ 16 GB/s per node • I/O: 160 TB shared disk, 72 * 140 GB local Software details: • Batch system: NQSII • OS version: TX7: SUSE SLES9, SX8: SUPER-UX 15.1 • Compiler versions:
General Comments	Long compilation times was the only problem Code_Saturne has been awarded a Gold Capability Incentive Award for Scalability on HPCx: http://www.hpcx.ac.uk/services/policies/capability.html
Porting report on programming language constructs in general	Code_Saturne contains several vectorizable loops
Porting report on libraries used	None
Porting report on parallelization method	MPI
Porting report on IO	No special requirements
PERFORMANCE RESULTS	
Execution platform	**IBM Blue Gene/L (Note – pre-runner architecture to targeted IBM BG/P listed above)**
Details execution platform	IBM PowerPC 440 800 MHz with IBM interconnect
Performance details	100M Cell Mixer_Grid (Prace Benchmark) Inclusive of I/O (200 iterations) Speed-up is relative to 512 core performance

Cores	Speed-up	Speed-up

	<table><tr><td></td><td>Mode CO</td><td>Mode VN</td></tr><tr><td>512</td><td>1</td><td></td></tr><tr><td>1024</td><td>2.01</td><td>1.68</td></tr><tr><td>2048</td><td>3.39</td><td>2.99</td></tr><tr><td>4096</td><td>N/A</td><td>5.79</td></tr></table>
Execution platform	**IBM Pwr5 Cluster (HPCx)**
Details execution platform	As IBM Pwr5 listed above
Performance Details	78M Cell Benchmark. N.B. This is a much simpler computational problem than the official Prace benchmark dataset. The run involves 50 iterations excusive of I/O. <table><tr><td>Cores</td><td>Time (s)</td></tr><tr><td>512</td><td>1132.7</td></tr><tr><td>1024</td><td>614.5</td></tr></table>
Execution platform	**Cray XT4 (HECToR)**
Details Execution platform	As listed above
Performance Details	100M Cell Mixer Grid (Prace Benchmark) 5 iterations, with and without I/O <table><tr><td>Cores</td><td>Time (s) Inclusive of I/O</td><td>Time (s) Exclusive of I/O</td></tr><tr><td>256</td><td>2235.3</td><td>2014.5</td></tr><tr><td>512</td><td>1176.8</td><td>979.0</td></tr></table>

	1024 658.2 476.1 2048 559.1 252.1 4096 520.0 123.4 8192 660.5 71.4
RECOMMENDATIONS	All Platforms
Expected potential for Petascaling	Medium: expect to be able to reach 10k cores with still speed-up over 5k cores
Expected effort to reach Petascaling potential	– I/O needs optimising for petascale architectures – Preprocessing stage needs improved scaling – Anticipated manpower required: 3 pms.
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	– Incorporation of high-performance numerical library routines – Loop optimisation e.g. on vector – Cache optimisation of finite volume scheme on scalar architectures – Anticipated manpower required: 3 pms.

7.2.6 *GADGET*

Orlando Rivera
LRZ - Leibniz Rechenzentrum

GENERAL	
Name of Code, Abbreviation	GADGET
Application area(s)	Cosmology, Cosmological structure formation
Key numerical method(s)	PDE, Space filling curves
Origin (developers, institute)	Dr. V. Springel, Max-Planck-Institute for Astrophysics
Current developers	Dr. V. Springel, Max-Planck-Institute for Astrophysics
Contact person	Dr. V. Springel <volker@MPA-Garching.MPG.DE>
License policy	General Public License GPL
Copyright	Volker Springel
Usage rules (within PRACE, outside PRACE, …)	Free inside and outside of PRACE
PRACE INFORMATION	
BCO: name, email, institute	Orlando Rivera, rivera@lrz.de , LRZ
Contributors (PRACE partners)	

Targeted hardware platforms as in BCO list	MPP-BG, ThinNode-x86, FatNode-Pwr6
CODE STATISTICS	
Programming language(s)	Ansi C89
Amount of source lines	30932
Libraries	FFTW 2.1.5, GSL 1.11, HDF5 (optional)
Parallellization method	Pure MPI 1.0
Development platform(s)	Altix 4700, LINUX Workstation
IO characteristics	By specifying write once every several time steps or at the end. Binary data written,endian dependent
PORTING REPORT	LRZ/HLRB2
Porting platform	SGI Altix 4700
Details porting platform	• Hardware: Model: SGI Altix 4700 CPUs: Intel Itanium2 Montecito Dual Core @ 1.6GHz N-cores: 9728 PEAK Performance: 62.3 Tflops Memory: 8.5GB shared memory among 2 or 4 processors (low or high density) Total Memory : 39 TByte Cache: L1 Data Cache: integer data only , 16kByte, cacheline: 64bytes, 4-way associative. L2 Data Cache: 256kBytes, cacheline 128 Bytes 8-way associativity, Float and integer . L3 Cache Data and Instruction: 6MB, cacheline 128Bytes 12-way , associative Interconnection: NUMAlink 4 IO: 39 Tbyte @ 20 GB/sec. • Software: OS: SLED 10 SP1 (GNU/Linux 2.6.16.54) intel C compiler 10.1 SGI native MPI , MPT gsdl 1.11, fftw 2.1.5 with mpi support
Overall porting result	Successful
General comments	• Because of the large input data set needed , an external mpi program with 32 mpi-tasks was used to generate and distributed. • Compilation straightforward, • Profiling with VampirTrace , intel trace analyser, and mpiP. VampirTrace and Intel trace analyser may produce very large data (> 200GB) • Hdw counter used PAPI and pfmon. Some minor difficulties , but ARCH dependent
Porting report on programming language constructs in general	• Ansi C89 supported by gcc and intel icc, • -O3, -falias , -ipo were useds
Porting report on libraries used	• Most libraries straightforward. FFTW 2.1.5 is an old version because its old interface is needed
Porting report on parallelisation method	• Pure MPI ver 1.2 used. Specific linkage needed when profiling tools used
Porting report on IO	• Large data Set read/write. Optional with HDF5. Endianness need to be specified

PORTING REPORT	JUGENE
Porting platform	Blue Gene/P
Details porting platform	• Hardware: Model: IBM Blue Gene P CPUs:32-bit PowerPC 450 core @ 850 MHz N-cores: 65536 Peak Performance: 223 Tflops Memory: 2 Gbytes per node Total Memory: 32 TByte Interconnect: Three-dimensonal torus • Software: GNU/Linux 2.6.16.54-0.2.5 ibm c compiler ver 9.0 ibm MPI (mpich-based) gsdl 1.11, fftw 2.1.5 with mpi support
Overall porting result	Successful
General comments	– Mostly same as Altix – Compilation straightforward, with xlc_r or mpcc – Profiling with mpiP. – PAPI couldn't be build, PAPI bug
Porting report on programming language constructs in general	– Ansi C89 supported by xlc_r, – -05 -qstrict were used
Porting report on libraries used	– Most libraries straightforward, need to build from scratch . Shared libraries were not build
Porting report on parallelisation method	– Pure MPI ver 1.2 used. Linkage against static libraries
Porting report on IO	– Large data Set read/write. Optional with HDF5. Endianness need to be specified
PORTING REPORT	SARA Huygens
Porting platform	IBM pSeries 575
Details porting platform	• Hardware: Model: IBM pSeries 575 CPUs: 104 nodes, 16 dual core processors (IBM Power6, 4.7 GHz) N-cores: 3328 Peak Performance: 60 Tflops Memory: 4 Gbytes per node L1 cache: 128 KByte of L1 cache per core (64 KByte data cache + 64 KByte instruction cache) L2 cache: 4 MByte per core (semi shared: the cache is assigned a specific core, but the other has a fast access to it) L3 cache: 32 MByte per processor Interconnect: Infiniband network, max MPI bandwidth 160 Gbit/sec • Software: GNU/Linux 2.6.16.60 ibm c compiler ver 9.0 ibm MPI shared lib gsdl 1.11, fftw 2.1.5 with mpi support,
Overall porting result	Successful
General comments	– Mostly same as JUGENE – Compilation straightforward, with xlc_r or mpcc
Porting report on programming language constructs in general	– Ansi C89 supported by xlc_r, – -05 -qstrict were used

Porting report on libraries used	– Most libraries straightforward, need to build from scratch . Shared libraries were not build
Porting report on parallelisation method	– Pure MPI ver 1.2 used. Linkage against shared libraries
Porting report on IO	– Large data Set read/write. Optional with HDF5. Endianness need to be specified
PERFORMANCE RESULTS	
Execution platform	Altix 4700
Details execution platform	Altix 4700
Performance details	See Attachement "altix_6.2.2.doc" additional timings runs:1024 cpus: 297 sec
PERFORMANCE RESULTS	
Execution platform	Blue Gene P
Details execution platform	Blue Gene P
Performance details	Same input as Altix, use it as baseline 512 cpus: 1593 sec. 1024 cpus: 773 sec. 2048 cpus: 406 sec
PERFORMANCE RESULTS	
Execution platform	IBM pSeries 575 SARA
Details execution platform	IBM pSeries 575
Performance details	Same input as Altix, use it as baseline 512 cpus: 496 sec. 1024 cpus: 276 sec.
RECOMMENDATIONS	For Petascaling and optimisation on LRZ altix 4700
Expected potential for Petascaling	medium
Expected effort to reach Petascaling potential	Hybrid MPI-OPENMP could increase the performance on particular sections of the code. Larger Segments of data should be prefered to about communication overhead Intrusive tracing reduced performance in a factor of 2 1 pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Some sections do not perform FP operations, change algorithms.
RECOMMENDATIONS	For Petascaling and optimisation on JUGENE BG/P
Expected potential for Petascaling	High
Expected effort to reach Petascaling potential	With a larger data set a ver large number of processors can be requested Some Sections produce a barrier, send_all and all_reduce , whose effects in this platform are more notorius, as long as all memory can be used per core, its potential is high Data Tracing is needed to investigate MPI overhead on this platform 2 pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Some sections do not perform FP operations, change algorithms.
RECOMMENDATIONS	For Petascaling and optimisation on SARA
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	Very fast execution times, However for a large number of cores more data need to be analysed The large amount of memory reduces the MPI overhead , it accepts even larger data set, the application can reach petascaling if enough ncpus are allowed, for this test we reach the max allowed by the system (1024 cpu)

	1 pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Some sections do not perform FP operations, change algorithms.

7.2.7 *TORB*

Xavier Saez
BSC-CNS

GENERAL	
Name of Code, Abbreviation	TORB / EUTERPE
Application area(s)	Fusion plasma
Key numerical method(s)	fft's, sparse solvers
Origin (developers, institute)	T.M. Tran, K. Appert, M. Fivaz, G. Jost, J. Vaclavik and L. Villard . Centre de Recherches en Physique des Plasmas, Lausanne , Switzerland
Current developers	R. Kleiber, R. Hatzky, and V. Kornilov . Max-Planck Institut fuer Plasmaphysik , Germany
Contact person	
License policy	Not Public Domain
Copyright	
Usage rules (within PRACE, outside PRACE, …)	Free within PRACE, not outside PRACE
PRACE INFORMATION	
BCO: name, email, institute	Xavier Sáez Pous <xavier.saez@bsc.es> and Guillaume Houzeaux <guillaume.houzeaux@bsc.es>, Barcelona Supercomputing Center
Contributors (PRACE partners)	None
Targeted hardware platforms as in BCO list	MPP-BG, FatNode-Pwr6, Cell
CODE STATISTICS	
Programming language(s)	FORTRAN 90
Amount of source lines	22.000
Libraries	PETSC and FFTW
Parallellization method	MPI
Development platform(s)	Intel, IBM, AIX, Fujitsu
IO characteristics	read at start, write periodically
PORTING REPORT	JUGENE
Porting platform	IBM Blue Gene/P

Details porting platform	16 Racks with 32 nodecards x 32 compute nodes (total 16384)
	• Compute node: 4-way SMP processor
	• Processortype: 32-bit PowerPC 450 core 850 MHz
	• Processors: 65536
	• Overall peak performance: 223 Teraflops
	• Linpack: 167 Teraflops
	• Main memory: 2 Gbytes per node (aggregate 32 TB)
	• I/O Nodes: 152
	• Networks:
	  • Three-dimensonal torus (compute nodes)
	  • Global tree / Collective network (compute nodes, I/O

	nodes) • 10 Gigabit ethernet / Functional network (I/O Nodes) • Power Consumption: max.40 kW per rack 2 Service nodes IBM p55A: • Total number of processors: 8 • Processortype: Power5 1.6 GHz • Total amount of memory: 32 GB • Operating System: SuSE Linux Enterprise (SLES 10) 2 Login nodes IBM p55A: • Total number of processors: 8 • Processortype: Power5 1.6 GHz • Total amount of memory: 32 GB • Operating System: SuSE Linux Enterprise (SLES 10) Internet address: jugene.zam.kfa-juelich.de
Overall porting result	Successfully
General comments	Compilation flags: -O3 -qarch=450d -qautodbl=dbl4 -qmaxmem=-1
Porting report on programming language constructs in general	
Porting report on libraries used	All the libraries needed by TORB are avalaible on JUGENE system.
Porting report on parallelisation method	TORB is well parallelised. No specific development has to be done.
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	IBM Blue Gene/P
Details execution platform	Same above
Performance details	The execution of 512 cores takes about half of the execution of 256 cores, so the program scales correctly.
RECOMMENDATIONS	
Expected potential for Petascaling	Expected scaling to tens of thousand of cores
Expected effort to reach Petascaling potential	Medium effort. The current code only allows executions until 999 cores.
Expected potential for Optimisation	Good potential
Expected effort to reach Optimisation potential	Great effort. Improve the communication scheduling. Simdization (the SIMD vectorization) of the loops.
PORTING REPORT	SARA
Porting platform	Power6
Details porting platform	IBM pSeries 575, a clustered SMP (Symmetric Multiprocessing) system. – 104 nodes – 16 dual core processors (IBM Power6, 4.7 GHz) per node – 128 GByte or 256 GByte of memory per node – 700 TByte of disk space – total peak performance is 60 Teraflop/sec In total, the system has: • 1664 dual core processors = 3328 cores • 15.25 TByte of memory • 700 TByte of disk space An IBM Power6 processor has the following characteristics:

	– Dual core running on 4.7 GHz – cd .L1 cache: 128 KByte of L1 cache per core (64 KByte data cache + 64 KByte instruction cache) – L2 cache: 4 MByte per core (semi shared: the cache is assigned a specific core, but the other has a fast access to it) – L3 cache: 32 MByte per processor The nodes are interconnected with an Infiniband network providing an MPI bandwidth of 160 Gbit/sec between neighboring nodes
Overall porting result	Successfully
General comments	Compilation flags: -O3 -q64 -qarch=auto -qautodbl=dbl4 -qmaxmem=-1
Porting report on programming language constructs in general	
Porting report on libraries used	The FFTW library is avalaible on SARA system, but the PETSc library is not installed.
Porting report on parallelisation method	CPMD is well parallelized. No specific development has to be done.
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	Power6
Details execution platform	Same above
Performance details	The executions with 256 and 512 cores go 2 times faster than the executions of Marenostrum.
RECOMMENDATIONS	
Expected potential for Petascaling	Expected scaling to tens of thousand of cores
Expected effort to reach Petascaling potential	Medium effort. The current code only allows executions until 999 cores.
Expected potential for Optimisation	Good Potential
Expected effort to reach Optimisation potential	Great effort. Improve the communication scheduling. Simdization (the SIMD vectorization) of the loops.

7.2.8 *ECHAM5*

Mark Cheeseman
CSCS

GENERAL	
Name of Code, Abbreviation	ECHAM5-HAM
Application area(s)	Earth & Atmospheric Sciences
Key numerical method(s)	Spectral code, FFTs
Origin (developers, institute)	Luis Kornblueh, Erik Roeckner *Max Planck Institute for Meteorology*
Current developers	Luis Kornblueh, Uwe Scholuzweida *Max Planck Institute for Meteorology* Ulrike Lohmann *ETH-Zurich*
Contact person	Luis Kornblueh <luis.kornblueh@zmaw.de>
License policy	Code is freely accessible after site/researcher agrees to MPI's Software License Agreement. Licensing is managed by Ms. Sonja Kempe at MPI <kempe@dkrz.de>.
Copyright	Max Planck Institute for Meteorology
Usage rules (within PRACE, outside	Free within PRACE. No disclosure/No propagation of the source

PRACE, …)	code.
PRACE INFORMATION	
BCO: name, email, institute	Mark Cheeseman <mpch@cscs.ch> *CSCS*
Contributors (PRACE partners)	Juha Lento <Juha.Lento@csc.fi> *CSC* Harald Klimach <klimach@hrls.de> *HRLS*
Targeted hardware platforms as in BCO list	MPP-Cray, FatNode-Pwr6, Vector-SX9
CODE STATISTICS	
Programming language(s)	F90
Amount of source lines	~50000
Libraries	BLAS, LAPACK, NetCDF
Parallellization method	MPI and OpenMP
Development platform(s)	NEC SX6 Cray XT3
IO characteristics	Output frequency specified by user via a namelist. Usually, restart files generated every model month, diagnostic output generated every 6 model hours.
PORTING REPORT	For XT platforms at CSCS
Porting platform	Cray XT3
Details porting platform	Hardware: • Model: Cray XT3 • Proc Type: AMD Opteron Dual Core • Clock rate: 2.6 GHz • Total Cores: 3328 • Cores Per Chip: 2 • Cores Per Node: 2 • Memory per core: 1GB. • Total Memory: 3328 Gbytes • Cache: Separate level 1 caches for data and instructions of 64 kB each. The L1 data cache is 2-way set associative. There is a combined data and instruction L2 cache of 1 MB for each core, which is 16-way associative. The L1 data and the L2 cache use 64 byte cache lines. The L2 cache acts as a victim cache for the L1 cache. Data evicted from the L1 cache gets established on the L2 cache. • Interconnect: Cray SeaStar 3D torus • I/O: 12 I/O nodes connected to 31 TB of RAID disks running Lustre Software details: • OS version: UNICOS/lc version 1.5.47 • Compiler versions: PGI compilers, version 7.2.4
Overall porting result	Successful
General comments	ECHAM5-HAM was already running on all Cray XT platforms at CSCS. No porting was required. Changing the number of cores used per test is easy as only a namelist needs to be modified –i.e. no re-compiling needed. Makefiles were already available for the XT platform and both the PGI and PathScale compilers. The principal compiler optimisations used were: `PGI: -O3 -fast -tp amd64`

	`PATHSCALE: -O3 –m3dnow –align64 –march=opteron` NOTE: # of cores used is dependent on the spectral resolution used. T106 allows a maximum of ~640 cores.
Porting report on programming language constructs in general	FORTRAN 90 code
Porting report on libraries used	Libraries used: • NetCDF Version 3.6.2 • ACML Version 4.0.1a
Porting report on parallelisation method	MPI-only parallelisation was used. OpenMP not currently supported on CSCS' XT platforms.
Porting report on IO	IO characteristics of benchmark: • Restart files generated at end of job • Diagnostic output datafiles generated hourly • Benchmark runs for 4 model days • Output frequency specified via a namelist IO is 2nd most time-consuming process. ECHAM5 developers extremely interested in IO optimisation.
PORTING REPORT	For CSCS
Porting platform	IBM Power5
Details porting platform	Hardware: • Model: eServer 575 cluster • Proc Type: Power5 • Clock rate: 1.5 GHz • Total Cores: 768 • Cores Per Chip: 2 • Cores Per Node: 16 • Memory per core: 2GB • Total Memory: 1784 Gbytes • Cache: Each core has a 32 Kbyte data cache and a 64 Kbyte instruction cache. The level 1 data cache has 128-byte lines, is 2-way set associative and write-through. The level 2 cache is on-chip, shared between the cores. It is a 1.9 Mbyte combined data and instruction cache, with 128 byte lines and is 10-way set associative and write-back. The level 3 cache is 36 Mbytes, off-chip and is shared between the 2 cores. It has 256 byte lines, and is 12-way set associative and write-back. • Interconnect: Infiniband. • I/O: 30 Tbytes of disk running GPFS. Connected to computes nodes via Infiniband Software details: • OS version: SLES 9 • Compiler versions: IBM XL Fortran compiler 9
Overall porting result	Successful
General comments	Porting ECHAM5-HAM was straight-forward as IBM PowerX configuration files already existed. The principal compiler optimisations used were: `-q64 –O2 -qarch=pwr5 -qtune=pwr5 -qessl` NOTES: 1> -O2 was the highest level of optimisation that could be safely used. 2> 1.7GB of physical memory alloted per MPI task. This allowed all 16 CPUs per node to be used.

	3> # of required CPUs can be selected by the user via a namelist. It is also dependent on the spectral resolution used. T106 allows a maximum of ~640 CPUs to be used.
Porting report on programming language constructs in general	FORTRAN 90 code
Porting report on libraries used	Libraries used: • MPICH (PrgEnv default version) • ESSL (OS default version) • LaPACK 3.1.l (built by user) • NetCDF Version 4.0.0
Porting report on parallelisation method	MPI-only and hybrid (MPI-OpenMP) parallelisation runs performed.
Porting report on IO	IO characteristics of benchmark: • Restart files generated at end of job • Diagnostic output datafiles generated hourly • Benchmark runs for 4 model days • Output frequency specified via a namelist IO is 2nd most time-consuming process. ECHAM5 developers extremely interested in IO optimisation.
PERFORMANCE RESULTS	For CSCS
Execution platform	Cray XT3
Details execution platform	Same with the porting platform
Performance details	Please find the detailed results in the attached results report *echam5_xt3_cscs.xls*.
PERFORMANCE RESULTS	For CSCS
Execution platform	IBM Power5
Details execution platform	Same with the porting platform
Performance details	Please find the detailed results in the attached results report *echam5_pwr5_cscs.xls*.
RECOMMENDATIONS	For Petascaling and optimisation on Cray XT4 (CSC)
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	With the T106L31 input datasets, ECHAM5-HAM scales well to 320 cores. A larger dataset (T159L95) is under construction and should allow greater scalability without any optimisation. OpenMP directives need to be added to the HAM submodel. Once completed, the hybrid MPI-OpenMP version should scale quite well to even higher core counts. *Amount of pm's: 2pm* Data output mechanism in ECHAM5-HAM creates significant load imbalance. This imbalance should hopefully be lessened with the inclusion of multiple IO nodes and MPI-2 directives. Rewriting the IO modules in ECHAM5-HAM will be extremely beneficial in increasing its scalability. *Amount of pm's: 6pm*
Expected potential for Optimisation	High
Expected effort to reach Optimisation potential	XT-optimised version of ECHAM5 exists and has been received from Cray. Optimisations are currently being evaluated so that the most beneficial ones can be added to the benchmark. *Amount of pm's: 0.5pms*
RECOMMENDATIONS	For Petascaling and optimisation at CSCS(IBM Power5)
Expected potential for Petascaling	Low-Medium
Expected effort to reach Petascaling	Global communication calls (such as MPI_Allreduce and

potential	MPI_Bcast) create a significant performance obstacle. Reducing these calls will improve scalability. Currently, the T106L31 configuration of ECHAM5-HAM scales well to 128 processors. *Amount of pm's: 0.5pm* An acute shortage of proper profiling tools on the Linux-on-Power platform made a performance investigation difficult. ECHAM5-HAM has internal counters that can sometimes be unreliable. However, the most recent OS version has allowed a new HPC Performance Toolkit to be available. Testing has started evaluating the usefulness of this utility. *Amount of pm's: 4pm* IO and OpenMP optimisation activities performed for the Cray XT platform are expected to be beneficial for the Linux-On-Power5 platform as well.
Expected potential for Optimisation	High
Expected effort to reach Optimisation potential	An IBM-optimised version of ECHAM5 exists and has been received. Optimisations are currently being evaluated so that the most beneficial ones can be added to the benchmark. *Amount of pm's: 0.5pms*
Further Porting Notes	**IBM Blue Gene**: Previous attempts to scale ECHAM5 on this architecture by DKRZ (Hamburg) and IBM failed. It is the opinion of the code developers (and IBM) that porting/optimisation efforts be ceased. **x86 thin-node**: No real barriers exist in porting ECHAM5-HAM to this platform as it already runs on a variety of linux-based clusters. CSC was able to port and run the benchmark on their HP cluster. It is expected that optimisations performed for the Cray XT and IBM Linux-On-Power architectures will be beneficial for the x86 thin-node architecture as well. **Optimisation Notes**: Preliminary work deliverables were drafted and assigned during the ECHAM5 Workshop on September 15. The first deliverables are due on October 17. These deliverables include the following: ▪ A benchmark configuration (reduced walltime requirement, higher resolution) - CSCS ▪ Input datasets for the new benchmark configuration – DKRZ,ETHZ ▪ Preliminary inclusion of OpenMP into the HAM module – HRLS ▪ Analysis of vendor-optimised ECHAM5 benchmarks used in the recent DKRZ procurement – CSCS ▪ Inclusion of additional instrumentation calls/functions – CSCS ▪ Preliminary analysis of the IO infrastructure – CSC **Other Notes**: A new version of ECHAM was released in October 2008. CSCS, DKRZ and ETHZ are currently evaluating this version to determine whether it can be used in the benchmark.

7.2.9 *NEMO*

Dr. John Donners

SARA

GENERAL	
Name of Code, Abbreviation	Nucleus for European Modelling of the Ocean, NEMO
Application area(s)	oceanography, climate science
Key numerical method(s)	
Origin (developers, institute)	Gurvan Madec et al, IPSL
Current developers	NEMO team, IPSL
Contactperson	Rachid Benshila
License policy	CeCiLL
Copyright	
Usage rules (within PRACE, outside PRACE, …)	GYRE configuration is free to use, DRAKKAR is only free to use within PRACE.
PRACE INFORMATION	
BCO: name, email, institute	John Donners, SARA
Contributors (PRACE partners)	
Targeted hardware platforms as in BCO list	MPP-Cray, Pwr6, MPP-BG, (vector)
CODE STATISTICS	
Programming language(s)	Fortran 90
Amount of source lines	82.000
Libraries	NetCDF
Parallellization method	MPI
Development platform(s)	
IO characteristics	read at start, write at end, each iteration (depends on configuration)
PORTING REPORT	For platform #1 (repeat for platform #2, #3, …)
Porting platform	Pwr6
Details porting platform	The huygens system at SARA consists of 104 nodes, each with 16 IBM Power6 (4.7GHz) dual-core processors and 128 or 256 gigabyte of internal memory. – L2 cache: 4 MByte per core – L3 cache: 32 Mbyte per core – Infiniband network: MPI bandwidth of 160 gigabit/second between neighbouring nodes. – SUSE Linux Enterprise Server 10 – IBM XL C/C++ 9.0, XL Fortran 11.1 Fortran – NetCDF 3.6.2 – Parallel Operation Environment, version 4.3.2.2-s002a
Overall porting result	Successful
General comments	
Porting report on programming language constructs in general	
Porting report on libraries used	
Porting report on parallelisation method	
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	Pwr6
Details execution platform	same as porting platform
Performance details	find the performance details and results on https://trac.csc.fi/pracewp6-nemo/wiki
RECOMMENDATIONS	For Petascaling and optimisation on platform #1 (repeat for platform #2, #3, …)
Expected potential for Petascaling	scaling is different for both configurations: the GYRE configuration

	scales well and has petascaling potential; the DRAKKAR configuration seems much less scalable at the moment.
Expected effort to reach Petascaling potential	– Extremely thin or wide regions for every task may be advantageous, because the cpu may be able to reuse cache data from a previous column or row. Amount of pm's:0.5 • The NEMO model uses exactly the same region size for every task, and land regions are also divided over tasks. If less tasks are given than the multiple of the nr. of tasks in both directions, the model will first remove tasks with only land points. There is a minimum nr. of tasks for every decomposition, where every task has some ocean points to calculate. The practically usable nr. of tasks therefore increases in steps which are not powers of 2. Possibly implement variable column widths or row heights to account for resulting imbalance? Amount of pm's: 1 • different solvers. Most of the communication is in the solver for the free surface. Different solvers can have different scaling characteristics. Amount of pm's: 0.5 • Try to interleave communication and calculation with Isend/Irecv before calculation and Waits after. Amount of pm's:0.5
Expected potential for Optimisation	medium
Expected effort to reach Optimisation potential	• Use different compiler flags and see what is the impact on performance. Compiler flags are platform and compiler dependent, so these will be different and hard-to-compare experiments on every platform. Some options possibly improve cache blocking, see e.g. Cray XT4 benchmark results. Amount of pm's: 0.2 • 'Cache blocking': Rewrite loops to improve cache reuse. Most important issue is how to keep code readable/maintainable. cpu time is equally spread over many routines. Single routine optimisations can therefore not be very effective. Advice on how to write such code is better than just changing the code. Amount of pm's: 1 DRAKKAR: Disk reads the same amount per node, independent of #tasks? Disk writes per node increase with #tasks? Large variety in FP stores and L2 misses?
PORTING REPORT	For platform #2
Porting platform	Cray XT4
Details porting platform	– AMD Opteron quad-core (2.3 GHz) processors – Cray SeaStar2 3D-torus High Speed Network – Cache (per core): – Two 64 KB L1 caches (instruction and data) – 1 MB L2 cache – 1 GB or 2 GB of memory per core. Peak transfer rate of 5.3 GB/s – Lustre parallel file system
Overall porting result	Successful
General comments	added --fastsse
Porting report on programming language constructs in general	No large efforts needed. The default small memory model for the PGI compilers resulted initially in an error when linking the model. Widening the domain decomposition, i.e. increasing the number of processes, reduces the memory per process and will fix the problem.
Porting report on libraries used	NETCDF-3.6.2
Porting report on parallelisation method	
Porting report on IO	

PERFORMANCE RESULTS	
Execution platform	Cray XT4
Details execution platform	
Performance details	find the performance details and results on https://trac.csc.fi/pracewp6-nemo/wiki NOTE that the benchmark differs on some minor points:300 timesteps, I/O needed to be done to separate files per task.
RECOMMENDATIONS	For Petascaling and optimisation on platform #2
Expected potential for Petascaling	Communication uses 50% of the wallclocktime at 512 tasks.
Expected effort to reach Petascaling potential	It is expected that optimisation on Pwr6, XT4 and BG is very similar.
Expected potential for Optimisation	medium
Expected effort to reach Optimisation potential	

PORTING REPORT	For platform #3
Porting platform	IBM BlueGene
Details porting platform	– P450 PowerPC chip, quad-core, 850 Mhz – Each core has own dedicated L1 cache – 2 GB RAM/node => 512 MB/core (fully occupied) – Nodes can be fully-, half- or singly-occupied – Bespoke high-performance interconnect with separate network for global operations – xlf2003 compiler, mpich MPI library, load-leveler scheduler
Overall porting result	Successful
General comments	i) Some care needed to get pre-processing working correctly. F2003 has signed zero which must be turned off for NEMO. Edit util/AA_make.gdef to add compiler definitions for BG/P: #-Q- bgp #- Global definitions for IBM BlueGene P (MPP) #-Q- bgp M_K = gmake #-Q- bgp MPIDIR=/bgsys/drivers/V1R2M0_200_2008-080513P/ppc/comm #-Q- bgp P_C = /lib/cpp #-Q- bgp P_O = -P -traditional-cpp -C $(P_P) -I$(MPIDIR)/include #-Q- bgp F_C = mpixlf2003 -c #-Q- bgp F_P = -qxlf90=nosignedzero -qrealsize=8 -qstrict #-Q- bgp w_w = $(F_P) -O3 -qarch=450d -qtune=450 -qsuffix=f=f90 #-Q- bgp F_O = -O3 -qarch=450d -qtune=450 -qfree=f90 $(F_D) $(F_P) -I$(MODDIR) -I$(MODDIR)/oce -qmoddir=$(MODDIR)/oce -I$(NCDF_INC) #-Q- bgp F_L = mpixlf2003 #-Q- bgp L_O = $(F_P) $(NCDF_LIB) #-Q- bgp A_C = ar -r #-Q- bgp A_G = ar -x #-Q- bgp C_C = mpixlc -c #-Q- bgp C_O = #-Q- bgp C_L = mpixlc #-Q- bgp prefix = "-WF,-D"

	#-Q- bgp #- #-Q- bgp NCDF_INC = $(HOME)/netcdf-3.6.2/include #-Q- bgp NCDF_LIB = -L$(HOME)/netcdf-3.6.2/lib -lnetcdf #-Q- bgp #- In particular, 'prefix' is set to the correct incantation for setting compiler-defined constants. EdiGet to link stage: ../../../lib/oce/libopa.a(in_out_manager.o): In function `__in_out_manager_NMOD_ctl_stop': in_out_manager.F90:(.text+0x804): undefined reference to `flush' From http://publib.boulder.ibm.com/infocenter/comphelp/v8v101/index.jsp discovered that 'flush' in F2003 is a _statement_ and therefore is _not_ CALL'd. I decided to change the code but have since realised that I could have added "FLUSH=FLUSH_" to the 'keys' in the build system. t ins_make file to add bgp as valid machine type.
Porting report on programming language constructs in general	
Porting report on libraries used	
Porting report on parallelisation method	
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	IBM BlueGene
Details execution platform	“scalasca” tool used for profiling on Jugene
Performanc	find the performance details and results on https://trac.csc.fi/pracewp6-nemo/wiki

e details	NOTE that the benchmark differs on some minor points:300 timesteps, I/O needed to be done to separate files per task.
RECOMMENDATIONS	For Petascaling and optimisation on platform #2
Expected potential for Petascaling	– Some filesystems (Lustre) cannot cope with a program opening ~100s of files in one go. => – Parallel-IO - NetCDF4?
Expected effort to reach Petascaling potential	See Power6
Expected potential for Optimisation	medium
Expected effort to reach Optimisation potential	See Power6

7.2.10 *CP2K*

Pekka Manninen
CSC Finland

GENERAL	
Name of Code, Abbreviation	CP2K
Application area(s)	Computational chemistry
Key numerical method(s)	FFT, dense matrix algebra
Origin (developers, institute)	University of Zürich
Current developers	Axel Kohlmeyer, Christopher J. Mundy, Fawzi Mohamed, Florian Schiffmann, Gloria Tabacchi, Harald Forbert, William Kuo, Jürg Hutter, Matthias Krack, Marcella Iannuzzi, Matthew McGrath, Manuel Guidon, Thomas D. Kuehne, Teodoro Laino, Joost VandeVondele, Valery Weber
Contactperson	Jürg Hutter
License policy	GPL
Copyright	
Usage rules (within PRACE, outside PRACE, …)	Free Software
PRACE INFORMATION	
BCO: name, email, institute	Pekka Manninen, pekka.manninen@csc.fi, CSC Finland
Contributors (PRACE partners)	Vegard Eige / Sigma Norway
Targeted hardware platforms as in BCO list	MPP-BG, MPP-Cray, FN-Pwr6
CODE STATISTICS	
Programming language(s)	Fortran 95
Amount of source lines	553,043

Libraries	LAPACK, SCALAPACK, FFT (FFTW, ACML, ESSL supported)
Parallellization method	MPI
Development platform(s)	
IO characteristics	Checkpoints and output

PORTING REPORT	
Porting platform	IBM Blue Gene / P (MPP-BG prototype)
Details porting platform	Power-PC 450 cores; Proprietary intereconnect; memory 512 MB per core
Overall porting result	Successful
General comments	No large efforts needed; but optimisation must be kept at a conservative level
Porting report on programming	Well written, standards-conforming Fortran 95
Porting report on libraries used	FFT interface to IBM ESSL library buggy, FFTW 3.1.5 works fine
Porting report on parallelisation method	Unable to take full advantage out of the BG interconnect
Porting report on IO	Not an issue
PERFORMANCE RESULTS	
Execution platform	Same as porting platform
Details execution platform	Same as porting platform
Performance details	Quickstep DFT dynamics of 512 water molecules 3 fs simulation, wall-time in secs Using the in-build FFT library (FFTSG) with and without the improved halo-exchange routines of the development version #Cores Original w/ improved routines 256 1755.93 1486.43 512 1499.06 1245.63 1024 1268.44 950.56 2048 1573.73 1080.18
RECOMMENDATIONS	Load imbalance and the amount of communication is the blockade for scaling. Not very good code for Blue Gene/P due to modest scalability and intense CPU and memory demands.
Expected potential for Petascaling	Not likely - low
Expected effort to reach Petascaling potential	?, load balance to be improved in general
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	1 man-months, some inefficiencies in cache utilization; the utilization of the second FP element of BG/P cores should be investigated

PORTING REPORT	
Porting platform	Cray XT5 (MPP-Cray prototype)
Details porting platform	AMD 2.3 GHz Barcelona quad-core CPUs; Proprietary intereconnect; memory 1-2 GB per core
Overall porting result	Successful
General comments	No large efforts needed
Porting report on programming	Well written, standards-conforming Fortran 95; numerics do not break down even with heavy optimisation
Porting report on libraries used	FFT interface to ACML library buggy, FFTW 3.1.5 works fine
Porting report on parallelisation method	
Porting report on IO	Fast I/O on Cray
PERFORMANCE RESULTS	
Execution platform	Same as porting platform
Details execution platform	Same as porting platform

Performance details	Quickstep DFT dynamics of 512 water molecules 3 fs simulation, wall-time in secs Using the in-build FFT library (FFTSG) with and without the improved halo-exchange routines of the development version #Cores Original w/ improved routines 64 1271.86 1223.14 128 845.50 818.14 256 585.96 544.95 512 527.94 500.07
RECOMMENDATIONS	Load imbalance and the amount of communication is the blockade for scaling
Expected potential for Petascaling	Not likely - low
Expected effort to reach Petascaling potential	?, load balance to be improved in general
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	2, for optimal cache utilization and vectorization

PORTING REPORT	
Porting platform	IBM Power 6 cluster (FN-Pwr6 prototype)
Details porting platform	4.7 GHz Power-PC CPUs, Infiniband, 4-8 GB/core
Overall porting result	Successful
General comments	No large efforts needed
Porting report on programming	Well written, standards-conforming Fortran 95
Porting report on libraries used	ESSL 4.3.1, FFTW 3.1.2 works fine
Porting report on parallelisation method	
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	Same as porting platform
Details execution platform	Same as porting platform
Performance details	Quickstep DFT dynamics of 512 water molecules 3 fs simulation, wall-time in secs #Cores w/ improved routines 64 1164 128 722 256 556 512 512
RECOMMENDATIONS	Load imbalance and the amount of communication is the blockade for scaling
Expected potential for Petascaling	
Expected effort to reach Petascaling potential	
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Cache utilization

7.2.11 *GROMACS*

Sebastian von Alfthan
CSC Finland

GENERAL	

Name of Code, Abbreviation	Gromacs 4.0
Application area(s)	Life sciences, Computational chemistry
Key numerical method(s)	Particle methods, Spectral methods(FFT)
Origin (developers, institute)	Herman Berendsens group, department of Biophysical Chemistry of Groningen University.
Current developers	Head Authors & project leaders: • Erik Lindahl (Stockholm Center for Biomembrane Research, SE) • David van der Spoel (Biomedical Centre, Uppsala, SE) • Berk Hess (Max Planck Institute for Polymer Research, Mainz, DE)
Contactperson	Erik Lindahl
License policy	GPL
Copyright	Authors have copyright to their code
Usage rules (within PRACE, outside PRACE, …)	Free inside & outside PRACE. Any changes should be contributed back to the original code

PRACE INFORMATION	
BCO: name, email, institute	Sebastian von Alfthan, alfthan@csc.fi, CSC
Contributors (PRACE partners)	PDC,CSCS,NCF
Targeted hardware platforms as in BCO list	MPP-Cray, FatNode-Pwr6,MPP-BG,

CODE STATISTICS	
Programming language(s)	C, assembler
Amount of source lines	~1.4M lines of code. ~340k lines of c, ~ 85k lines in header files, ~1M lines of assembler.
Libraries	FFTW,BLAS,LAPACK
Parallellization method	MPI
Development platform(s)	
IO characteristics	With most systems not very demanding. read at start, write at end, write each n iterations. Automatic snapshoting ability with restarting.

PORTING REPORT	For platform #1
Porting platform	MPP-CRAY
Details porting platform	Current louhi: XT5 with 800MHZ DDR2 memory. OS2.1, GCC 4.2.0 (patched for QC support) FFTW 3.1.1, ACML 4.1.0
Overall porting result	Successful
General comments	Did not encounter any specific problems.
Porting report on programming language constructs in general	Uses assembler routines for short-ranged forces.
Porting report on libraries used	Used FFTW 3.1.1 and ACML 4.1.0 that were installed as modules.
Porting report on parallelisation method	No issues.
Porting report on IO	No issues

PORTING REPORT	For platform #2
Porting platform	PWR6
Details porting platform	Current Huygens: Kernel: 2.6.16.60-0.2501-ppc64 IBM XL Fortran for Linux, V12.1 Version: 12.01.0000.0000 IBM XL C/C++ for Linux, V10.1 Version: 10.01.0000.0000

	FFTW 3.1.2
Overall porting result	Successful
General comments	
Porting report on programming language constructs in general	Uses assembler routines for short-ranged forces.
Porting report on libraries used	Used precompiled libraries available on BG/P.
Porting report on parallelisation method	No issues
Porting report on IO	No issues

PORTING REPORT	For platform #3
Porting platform	MPP-BG/P
Details porting platform	Current Jugene: FFTW 3.1.2 IBM XL C/C++
Overall porting result	Successful
General comments	Ported successfully based on instructions from FZJ.
Porting report on programming language constructs in general	Gromacs has assembler routines developed for the BG/P for calculating short-ranged forces.
Porting report on libraries used	Used precompiled libraries available on BG/P.
Porting report on parallelisation method	No issues
Porting report on IO	No issues
PERFORMANCE RESULTS	For platform #1
Execution platform	MPP_CRAY
Details execution platform	Louhi – see porting platform
Performance details	**Algorithms** Long-ranged forces are calculated using the smooth particle-mesh Ewald (SPME) scheme. This scheme divides the calculation of long-ranged forces in two parts, a short-ranged calculation in real-space and a Fourier-space part where the charges are assigned to a grid. The Fourier part requires two 3D-FFT calculations, to Fourier space and back. As 3D-FFT:s require all-to-all communication, Gromacs assigns a certain set of MPI processes to only calculate the Fourier-space part of PME (PME processes), while the others calculate normal short-ranged interactions (real-space processes). This enables the code to extract better performance from the interconnect on a machine with multiple cores per node. In Gromacs the 3D-FFT is currently only parallelised in one dimension, this unfortunately sets a great deal of restrictions on the number of processes one can effectively use with PME. Generally one should aim at having 25-33% of the processes assigned to long ranged forces. It also limits the scalability as the communication requirements are higher due to the 1D implementation. Alternatively one can also use coarser approximations such as the reaction field (RF) approximation where the long-ranged Coulomb forces are approximated via short-ranged interactions. This algorithm removes most all-to-all communication and should scale much further. **Testcase** The testcase comprised two vesicles in water with 1752 POPC lipids and 334489 water molecules giving in total 1094681 atoms. The system was provided by Erik Lindahl, a Gromacs developers. Long-ranged forces

were calcuated with either PME or RF. For PME the grid-dimension of the test-case in the parallelised direction was 176. The number of iterations was five times greater for the RF simulation, due to its better performance.

PME-results

PME/real-space load-balance
Getting the load balance correct between processes calculating PME and processes calculating short-ranged forces is of crucial importance to get maximum performance:

N	N(PME)/N	PME/RS load	time(s)
132	0.33	1.15	772
176	0.25	1.45	645
176	0.5	1.01	610
264	0.33	1.56	456
352	0.25	1.89	418
352	0.5	1.70	502
528	0.33	2.52	382
704	0.25	3.04	339

On the XT5 maximum performance is not always achieved when the load balance is the best possible one. For the case of 352 cores the best performance is achieved when the load balance suggests that there are far too few PME nodes. The reason is that with fewer PME processes per node, the performance of all-to-all operations is significantly increased. The Cray MPI library doesn't appear to do any message aggregation and thus the all-to-all algorithm works best if only one process per node is involved in it.

Node performance
The node-performance is excellent due to the built in X86-64 assembler routines for short ranged forces. An 88-core simulation achieves a performance of 2.4 Gflops/core according to Gromacs built-in counters.

Scalability

The scalability for a PME process fraction of 0.25 or 0.5 is:

cores	time(s)	speedup	GFlops
176	610		347.2
352	418	1.46	457.1
704	339	1.23	563.9

The scalability for a PME process fraction of 0.33 is:

cores	time(s)	speedup	GFlops
132	772		267.4
264	456	1.69	430.5
528	382	1.18	513.9

The scalability is limited by the calculation of long ranged forces; the all-to-all communication this requires does not scale. This is evident in the fact that for 352 cores optimal performance was obtained for a case with a greater degree of load imbalance.

RF-results

Scalability
The scalability for the RF test-case is:

cores	time(s)	speedup	GFlops
512	447		1515.6
1024	264	1.69	2567.1
2048	166	1.59	4085.2

	With the RF approximation Gromacs scales extremely well. It also achieves an impressive flops count of almost 3 Gflops/core for the 512 core simulation. Note that the number of iterations is five times larger than for the PME runs, thus the execution times cannot be directly compared. **I/O** For this testcase and many other MD simulations, I/O is not very demanding. Time spent writing out results is close to 0%.

RECOMMENDATIONS	For Petascaling and optimisation on platform #1
Expected potential for Petascaling	For PME-simulations the scalability can be improved to one or several thousands of cores For RF-simulations the scalability will probably extend to ten thousand cores, with large enough systems. • With 2D or 3D-PME the major scalability bottleneck for PME can be alleviated. • Optimised communication patterns (all-to-all) with aggregated messages could help on the Cray-XT5 high-bandwidth medium-latency network. • Gromacs doesn't overlap communication with computation. This could potentially improve scalability as communication costs could be hidden. This could also help with the scalability of RF calculations • Gromacs 4 has a brand new parallelisation and thus we expect that there will also be additional things that can be tuned.
Expected effort to reach Petascaling potential	8 pm (petascaling on all platforms)
Expected potential for Optimisation	Small. • Most processor intensive parts of the code are written in well optimised assembler and accounts for 70% of all floating point operations.
Expected effort to reach Optimisation potential	1 pm (optimisation on all platforms) We will spend some effort to verify that the code is well tuned and that there are no obvious problems.

PERFORMANCE RESULTS	For platform #2
Execution platform	Fatnode-PWR6
Details execution platform	Huygens – see porting platform
Performance details	**PME-results** **PME/real-space load-balance** Getting the load balance correct is of crucial importance to get maximum performance: **N N(PME)/N PME/RS load time(s)** 132 0.33 0.937 866 176 0.25 1.361 774 264 0.33 1.015 465 352 0.25 1.426 431 528 0.33 1.185 281 On the PWR6 the best performance is achieved when one third of the processes is assigned to PME. This is also what the Gromacs preprocessing tool grompp suggests. **Node performance** The node-performance is good, but not quite as good as on the Cray XT5; the wall time of smaller runs with 132, 176 and 264 processes is larger. The reason for this is unclear; we suspect that the hand-tuned assembler routines have not been specifically tuned for the new Power 6 processors. **Scalability** The scalability for the test-case is much better than on the Cray XT5. The larger runs with 428 processes executes faster than on the XT5 platform. The scalability for a PME process fraction of 0.33 is: **cores time(s) speedup GFlops** 132 866 226.6 264 465 1.86 422.1 528 281 1.65 698.6 **RF-results** **Scalability** The scalability for the RF test-case is: **cores time(s) speedup GFlops** 256 906 747.5 512 503 1.80 1346.8 1024 281 1.79 2411.3 When one uses the RF approximation Gromacs scales extremely well. The scalability is better than on the XT5 platform, but the performance remains lower even for 1024 cores. It achieves a flops count of 2.6 Gflops/core with 512 cores.

RECOMMENDATIONS	For Petascaling and optimisation on platform #2
Expected potential for Petascaling	For PME-simulations the scalability can be improved to one or several thousands of cores For RF-simulations the scalability will probably extend to ten thousand cores, with large enough systems.

	• With 2D- or 3D-PME larger number of PME nodes can be used, enabling larger simulations • Gromacs doesn't overlap communication with computation. Doing this could potentially improve scalability as communication costs could be hidden. • Gromacs 4 has a brand new parallelisation and thus we expect that there will also be additional things that can be tuned.
Expected effort to reach Petascaling potential	8 pm (petascaling on all platforms)
Expected potential for Optimisation	Small; the most processor intensive parts of the code are written in well optimised assembler.
Expected effort to reach Optimisation potential	1 pm (optimisation on all platforms) We will spend some effort to verify that the code is well tuned and that there are no obvious problems.
PERFORMANCE RESULTS	For platform #3
Execution platform	BG/P
Details execution platform	Jugene – see porting platform
Performance details	**PME-results** As there is a runtime limit for small queues with 512 cores the number of iterations was reduced by half. . **PME/real-space load-balance** **N N(PME)/N PME/RS load time(s)** 512 0.18 0.835 852 1024 0.18 1.002 462 On the BG/P one can use a much smaller PME fraction while still having good load balance. This is fortunate, as it allows larger simulations to be performed. **Node performance** There are assembler routines for the BG/P architecture that have been enabled at compile time. These are able to extract good performance from the processors. The low clock frequency of the processors is evident as much larger simulations have to be run to achieve comparable performance to the one on the Cray XT5 machine. **Scalability** The scalability the test-case is as follows: **cores time(s) speedup GFlops** 512 852 110.3 1024 462 1.84 203.5 Unfortunately one cannot calculate with a larger number of processors as the parallel algorithm (PME) does not scale any further. Thus performance is lower than on either the PWR6 or the XT5.

RECOMMENDATIONS	For Petascaling and optimisation on platform #3
Expected potential for Petascaling	For PME-simulations the scalability can be improved to several thousands of cores For RF-simulations the scalability will probably extend to several tens of thousands of cores, with large enough systems. • With 2D or 3D-PME larger number of PME nodes can be used, enabling larger simulations • Investigating how the torus network of BG/P can be directly mapped to the domain decomposition used by Gromacs • Gromacs 4 has a brand new parallelisation and thus we expect that there will also be additional things that can be tuned.
Expected effort to reach Petascaling potential	8 pm (petascaling on all platforms)
Expected potential for Optimisation	Small; the most processor intensive parts of the code are written in well optimised assembler.
Expected effort to reach Optimisation potential	1pm (optimisation on all platforms) We will spend some effort to verify that the code is well tuned and that there are no obvious problems.

7.2.12 *N3D*

Harald Klimach
HLRS

GENERAL	
Name of Code, Abbreviation	N3D
Application area(s)	CFD
Key numerical method(s)	FFT, sparse linear solver on structured grids
Origin (developers, institute)	Ulrich Rist, IAG, University of Stuttgart
Current developers	IAG, University of Stuttgart
Contact person	Tillman Friederich <friederich@iag.uni-stuttgart.de>
License policy	This code has access restrictions: permission for use must be obtained from Tillman Friederich <friederich@iag.uni-stuttgart.de>.
Copyright	IAG, University of Stuttgart
Usage rules (within PRACE, outside PRACE, …)	Free within PRACE. No disclosure/No propagation of the source code.
PRACE INFORMATION	
BCO: name, email, institute	Harald Klimach <klimach@hlrs.de> HLRS
Contributors (PRACE partners)	
Targeted hardware platforms as in BCO list	MPP-Cray, ThinNode-x86, FatNode-Pwr6, Vector
CODE STATISTICS	
Programming language(s)	F90

Amount of source lines	16088
Libraries	EAS3, Netlib (FFT)
Parallellization method	MPI, NEC-MicroTasks
Development platform(s)	NEC-SX8 (a1.hww.de), Linux Workstations
IO characteristics	Read at start and write at end and on restart points (typically around every 200 iterations, can be specified in the parameter file)
PORTING REPORT	For a1.hww.de NEC-SX8
Porting platform	NEC-SX8
Details porting platform	Hww NEC-SX8 is described in Deliverable 6.1
Overall porting result	Successful
General comments	As this platform is the main development platform for this application, there are no issues. The only thing, that needs special care and may cause conflicts with libraries or instrumentation is the usage of the global -ew compiler flag for the sxf90 compiler, which enforces 4 byte integers and reals.
Porting report on programming language constructs in general	N3D code is written in Fortran 90 and can be compiled using the provided Makefiles.
Porting report on libraries used	Libraries need to be available in the double precision compiled version compatible to the application itself (compiled with compiler option -ew), but the required libraries are available for the SX8.
Porting report on parallelisation method	Special shared memory parallelisation for this platform available: NEC-MicroTasking. Just compiling with the systems Fortran 90 compiler is sufficient to gain full parallelisation possibilities.
Porting report on IO	IO uses the EAS3 library, which needs to be compiled with a compatible dataformat (-ew option). No further actions needed to be taken on IO.
PORTING REPORT	For bwGrid
Porting platform	x86-TN
Details porting platform	Harpertown-Cluster with 8 Cores and 16 GB RAM per node. Nodes connected over Infiniband interconnect.
Overall porting result	In progress.
General comments	Propably major adaptions necessary.
Porting report on programming language constructs in general	N3D code is written in F90 and is fairly portable, however the problemsize is hard coded during compile time into the executable, and compilers give up on this due to memory issues with static data as it seems. Also one of the files is not compilable by the Intel compiler, but is by the PGI.
Porting report on libraries used	Libraries need to be available in the double precision compiled version compatible to the application itself (compiled with compiler options -i8 -r8), but the required libraries are available for the platform.
Porting report on parallelisation method	Special shared memory parallelisation is used in N3D: NEC-MicroTasking. This is not available on the x86 platform and needs to be replaced (with OpenMP). There is some OpenMP parallelisation in the code but it seems to be unrelated to the parallelisation used on the SX. It is unclear if the current MPI-Implementation, which is relying on Microtasking in tandem, is suitable at all with that OpenMP parallelisation.
Porting report on IO	IO uses the EAS3 library, which needs to be compiled with a compatible dataformat (-i8 -r8 option). No further actions needed to be taken on IO.
PERFORMANCE RESULTS	For hww NEC-SX8
Execution platform	Hww NEC-SX8
Details execution platform	See Deliverable 6.1
Performance details	Please find the detailed results in the attached results report N3D_SX8.*xls*.

RECOMMENDATIONS	For Petascaling and optimisation on NEC-SX8 (hww-NEC-SX8)
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	Porting N3D is very straightforward and the code is developed on that platform. It scales up to the complete available machine, so there is a possibility that it can reach Petascaling potential. It should be noticed the MPI communication is using mainly AllToAll routines. Reducing this part would be helpful to improve the scaling. Amount of pm's: 1pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Most of the computational time spent in the selected testcase is spent in the FFT calls to the Netlib implementation. Changing theses to more platform specific implementations should help to increase the sustained performance on the platform. Other parts of the application already gain a high sustained performance. Amount of pm's: 1pm
RECOMMENDATIONS	For Petascaling and optimisation on x86-TN
Expected potential for Petascaling	low
Expected effort to reach Petascaling potential	High, the used algorithms do not allow distribution on more than a limited number of processes (depending on the dataset).
Expected potential for Optimisation	Medium, the code is not yet optimised for this platform, and there are features of the architecture that might be better exploited.
Expected effort to reach Optimisation potential	High: The algorithms are not designed to fit the needs of thin node x86 platforms, it can be expected to be a lot of work and require some redesigns to gain optimal performance for this platform. Amount of pm's: 4pm

7.2.13 *AVBP*

Bertrand Cirou (also with Francois Rue's effort)
CINES

GENERAL	
Name of Code, Abbreviation	AVBP
Application area(s)	Turbulent Combustion + CFD
Key numerical method(s)	Large Eddy Simulation
Origin (developers, institute)	
Current developers	Gabriel Staffelbach, CERFACS (Toulouse, FRANCE)
Contact person	gabriel.staffelbach@cerfacs.fr
License policy	This code has access restrictions: permission for use must be obtained from gabriel.staffelbach@cerfacs.fr
Copyright	CERFACS
Usage rules (within PRACE, outside PRACE, …)	Free within PRACE. No disclosure/No propagation of the source code.
PRACE INFORMATION	
BCO: name, email, institute	Bertrand Cirou, cirou@cines.fr, CINES
Contributors (PRACE partners)	
Targeted hardware platforms as in BCO list	MPP-BG/P, Thin-Node x86, FatNode-Pwr6
CODE STATISTICS	

Programming language(s)	F90
Amount of source lines	239578
Libraries	Hdf5, szip, Metis
Parallellization method	MPI
Development platform(s)	SGI ORIGIN 3800, Power 5, generic x86_64, Bull itanium, Cray XT3/XT4
IO characteristics	Write once every several time steps. The output frequency needs to be specified in the source code by assign a particular parameter.
PORTING REPORT	For jade.cines.fr
Porting platform	SGI ICE EX 8200
Details porting platform	Hardware: • Model: Cray XT4 • Proc Type: Xeon Harpertown E5472 • Clock rate: 3GHz • Total Cores: 12288 • Cores Per Chip: 4 • Cores Per Node: 8 • Memory per core: 4GB. • Total Memory: 49152 Gbytes • Cache : primary 32-kB instruction cache and • 32-kB write-back data cache in each core and 12 MB (2 x 6MB) Level 2 cache with Intel Advanced Smart Cache architecture. The processors' Data Prefetch Logic • speculatively fetches data to the L2 cache before an L1 cache requests occurs, resulting in reduced effective bus latency and improved performance. The 1600 MHz Front Side Bus (FSB) is a quadpumped bus running off a 400 MHz system clock making 12.80 GBytes per second data transfer rates possible • Interconnect: two planes of infiniband 4x DDR with ConnectX Mellanox (hypercube network) • I/O: 20 I/O nodes connected to 640 TB of RAID6 with LSI controllers, disks running Lustre Software details: • OS version: SLES 10 patch 1+SGI ProPack 5SP5 • Compiler versions: Intel 10.1.017
Overall porting result	Successful
General comments	It was very straightforward to port AVBP to jade.cines.fr Makefiles are provided for all architectures
Porting report on programming language constructs in general	AVBP code is written in FORTRAN 90 and can be compiled using the intel ifort compiler directly.
Porting report on libraries used	
Porting report on parallelisation method	Using Intel ifort compiler and no other special requirements for the MPI code porting.
Porting report on IO	

PORTING REPORT	For vargas.idris.fr
Porting platform	IBM Power6
Details porting platform	Hardware: – Model: IBM eServer SMP p575 IH POWER6 cluster – Proc Type: Power6 – Clock rate: 4.7 GHz – Total Cores: 3584 – Cores Per Chip: 2 – Cores Per Node: 32 – Memory per core: 8GB

	– Total Memory: 16500 Gbytes – Cache: Each core has a 64 Kbyte data cache and a 64 Kbyte instruction cache. The level 2 cache is on-chip, shared between the cores. It is a 4 Mbyte combined data and instruction cache. The level 3 cache is 32 Mbytes. – Interconnect: Infiniband 4x DDR – I/O: 400 Tbytes of disk running GPFS. Software details: – OS version: AIX 5.3 – Compiler versions: IBM XL Fortran compiler 11.1.0.3
Overall porting result	Successful
General comments	Porting AVBP to vargas.idris.fr is straightforward. Some minor issues on compiler flags usage had to be solved.
Porting report on programming language constructs in general	AVBP code is written in FORTRAN 90 and can be compiled using the IBM XL MPI Fortran compiler.
Porting report on libraries used	AVBP needed to be linked with a specific HDF5 library on vargas.idris.fr to avoid a GPL License problem with libz.
Porting report on parallelisation method	Using IBM XL Fortran MPI compiler.
Porting report on IO	

PERFORMANCE RESULTS	For vargas.cines.fr
Execution platform	IBM eServer SMP p575 IH POWER6 cluster
Details execution platform	Same with the porting platform
Performance details	No results. Only one node was available at this time.

PERFORMANCE RESULTS	For jade.cines.fr
Execution platform	SGI ICE 8200 EX
Details execution platform	Same with the porting platform
Performance details	Optimal scalability on 4096 cores 10 % below optimal on 8192 cores
RECOMMENDATIONS	For Petascaling and optimisation on x86_64 + infiniband
Expected potential for Petascaling	Good
Expected effort to reach Petascaling potential	Numerical stability on allreduce Amount of pm's: 2pm
Expected potential for Optimisation	Low, AVBP code is already written with performances in mind
Expected effort to reach Optimisation potential	Amount of pm's: 6pm

7.2.14 *HELIUM*

Xu Guo (also with the effort from Jon Hill and Andrew Sunderland
EPCC

GENERAL	
Name of Code, Abbreviation	HELIUM
Application area(s)	Atomic Physics
Key numerical method(s)	Sparse linear algebra
Origin (developers, institute)	Jonathan Parker, Ken Taylor, Queen's University Belfast
Current developers	Queen's University Belfast
Contact person	Ken Taylor <k.taylor@qub.ac.uk>
License policy	This code has access restrictions: permission for use must be obtained from Ken Taylor <k.taylor@qub.ac.uk>.
Copyright	Queen's University Belfast

Usage rules (within PRACE, outside PRACE, …)	Free within PRACE. No disclosure/No propagation of the source code.
PRACE INFORMATION	
BCO: name, email, institute	Xu Guo <xguo@epcc.ed.ac.uk> EPCC, The University of Edinburgh
Contributors (PRACE partners)	Jon Hill <jon@epcc.ed.ac.uk> EPCC, The University of Edinburgh Andrew Sunderland <a.g.sunderland@dl.ac.uk> STFC Daresbury
Targeted hardware platforms as in BCO list	MPP-BG, MPP-Cray, FatNode-Pwr6
CODE STATISTICS	
Programming language(s)	FORTRAN 90
Amount of source lines	14569 (in one file)
Libraries	
Parallellization method	MPI
Development platform(s)	HPCx (IBM Power 5), SGI ORIGIN 3000, INTEL 64 bit PENTIUM machines , DELL (maybe on ITANIUMS) , Cray (AMD opterons) , Honeywell Bull , NEWTON, LINUX WORK STATIONS
IO characteristics	Write once every several time steps. The output frequency needs to be specified in the source code by assign a particular parameter.
PORTING REPORT	**For Louhi @ CSC**
Porting platform	Cray XT4 / XT5 (MPP-Cray)
Details porting platform	Hardware: • Model: Cray XT4/XT5 • Proc Type: AMD Opteron Quad Core • Clock rate: 2.3 GHz • Cores Per Chip: 4 • Cores Per Node: 4 cores for XT4 (i.e. 1 processor per node), 8 cores for XT5 (i.e. 2 processors per node) • Memory per core: mostly 1 GB/core, except one XT4 cabinet having 2GB/core and one XT5 cabinet having 2GB/core. • Interconnect: Cray SeaStar2+ 3D torus • I/O: Lustre Software details: • OS: UNICOS/lc • Compiler versions: PGI compilers, version 7.2.4; PathScale compilers, version 3.1;
Overall porting result	Successful
General comments	1. Problem size and cores number The cores number and memory size required for the code execution are related to the parameters value in the source code, so not all the cores number or problem size can be selected for the benchmark tests. On Cray XT4 part, the problem size 770 and 1540 were benchmarked with (66, 105), 253, (406), 630, 990, 1540 cores. On Cray XT5 part, the problem size 770 and 1540 were benchmarked with (66, 105), 253, (406) cores. 2. Compiling No Makefile is provided along with the source code. On the Cray

	XT4/XT5 system, the HELIUM code can be compiled using either PGI FROTRAN90 compiler or PathScale FORTRAN90 compiler, as below: With the module PrgEnv-pgi loaded: ftn -fast -Mipa=fast,inline helium.f90 -o helium With the module PrgEnv-pathscale loaded*:* ftn –O3 –OPT:Ofast helium.f90 -o helium The HELIUM source code can be compiled directly by the PGI compiler, but it should be noticed that using the PGI compiler may encounter a reallocation limit compiling problem if some parameters in the source code has a large value. If using the PathScale compiler, the current version code then needs some minor modifications because of the FORTRAN 90 syntax checking. Note that one module can not be used in both the routine and its subroutines which are defined inside the routine body. 3. Memory limit HELIUM code consume memory a lot, so even with a successfully build, the execution may failed due to the memory size limitation.
Porting report on programming language constructs in general	HELIUM code is written in FORTRAN 90 and can be compiled using the Fortran 90 compiler.
Porting report on libraries used	
Porting report on parallelisation method	Using PGI f90 compiler or PathScale f90 compiler for the MPI code compiling.
Porting report on IO	The IO frequency was specified in the test cases' source code. It will define how many times the output will be written out during the execution. In the benchmarks on the Cray XT4/XT5 prototype, the total time steps was defined as 80 and the output was written out once every 20 time steps.
PORTING REPORT	**For HECToR**
Porting platform	Cray XT4 (dual core MPP-Cray) (This platform is not a prototype, but related to the prototype Louhi Cray XT4/XT5.)
Details porting platform	Hardware: • Model: Cray XT4 (dual core) • Proc Type: AMD Opteron Dual Core • Clock rate: 2.8 GHz • Total Cores: 11328 • Cores Per Chip: 2 • Cores Per Node: 2 • Memory per core: 3GB/core. • Total Memory: 33984 Gbytes • Cache: Separate level 1 caches for data and instructions of 64 kB each. The L1 data cache is 2-way set associative. There is a combined data and instruction L2 cache of 1 MB for each core, which is 16-way associative. The L1 data and the L2 cache use 64 byte cache lines. The L2 cache acts as a victim cache for the L1 cache. Data evicted from the L1 cache gets established on the L2 cache. • Interconnect: Cray SeaStar2 3D torus • I/O: 12 I/O nodes connected to 576 TB of RAID disks running Lustre Software details: • OS: UNICOS/lc

	• Compiler versions: PGI compilers, version 7.1.4; PathScale compilers, version 3.0.
Overall porting result	Successful
General comments	See the comments above for Louhi Cray XT4/XT5. Problem size 1540 scaling with 66, 105, 253, 406, 630, 990, 1540 cores are benchmarked on Hector Cray XT4 (dual core). Problem size 682 and 1364 scaling with 253 and 496 cores are also benchmarked for D6.2.2.
Porting report on programming language constructs in general	HELIUM code is written in FORTRAN 90 and can be compiled using the Fortran 90 compiler.
Porting report on libraries used	
Porting report on parallelisation method	Using PGI f90 compiler or PathScale f90 compiler for the MPI code compiling.
Porting report on IO	The IO frenquency was specified in the test cases' source code: for the benchmark tests with 40 time steps, the output was written out once every 10 time steps; for the benchmark tests with 80 time steps, the output was written out once every 20 time steps.
PORTING REPORT	**For Huygens @ SARA**
Porting platform	IBM Power6 (FatNode-Pwr6)
Details porting platform	Hardware: • Model: IBM eServer pSeries 575 • Proc Type: Power 6 • Clock rate: 4.7 GHz • Total Cores: 3328 • Cores Per Chip: 2 • Cores Per Node: 32 • Total Memory: 15.25 TByte • Cache: L1 (per core) – 128KB; L2 (per core, semi shared: the cache is assigned a specific core, but the other has a fast access to it) – 4MB; L3 (per processor) – 32MB; • Interconnect: The nodes are interconnected with an Infiniband network providing an MPI bandwidth of 160 Gbit/sec between neighboring nodes. Software details: • OS: SuSE Linux SLES10 SP2 • Compiler versions: IBM XL Fortran compiler version 12.1
Overall porting result	Successful
General comments	1. Problem size and cores numbers Problem size 1540 and cores numbers 253, 406, 630, 990, 1540 are selected for the benchmarking for the prototype Huygens Pwr6. 2. Compiling The code was compiled by the command line as below: mpfort -qfree=f90 –O3 -qessl helium.f90 -o helium To compile the Fortran 90 code on Huygens Pwr6, the flag -qfree=f90 must be used for the mpfort. The Flag –qessl is not necessary for the compiling, but could improve the execution performance as the ESSL will replace the Lapack for the matrix calculation. 3. Running Not all the cores number can be selected for the benchmarking, so there is a big possibility of not fully allocating tasks on nodes. Therefore in the job script, the total tasks number and total node number should be required separately, for example:

	#@ total_tasks = 253 #@ node = 8 Otherwise the total tasks number will be set as the node number multiply by 32 by default.
Porting report on programming language constructs in general	HELIUM code is written in FORTRAN 90 and can be compiled using the IBM XL MPI Fortran compiler.
Porting report on libraries used	HELIUM can be linked with the ESSL scientific routine library on Huygens which will make a performance improvement.
Porting report on parallelisation method	Using IBM XL Fortran MPI compiler.
Porting report on IO	The IO frenquency was specified in the test cases' source code: for the benchmark tests on Huygens the total time steps was set as 80 and the output was written out once every 20 time steps.
PORTING REPORT	**For HPCx**
Porting platform	IBM Power5 (Fatnode-Pwr5) (This platform is not a prototype, but related to the prototype Huygens Pwr6)
Details porting platform	Hardware: • Model: IBM eServer 575 cluster • Proc Type: Power5 • Clock rate: 1.5 GHz • Total Cores: 2560 • Cores Per Chip: 2 • Cores Per Node: 16 • Memory per core: 2GB • Total Memory: 5120 Gbytes • Cache: Each core has a 32 Kbyte data cache and a 64 Kbyte instruction cache. The level 1 data cache has 128-byte lines, is 2-way set associative and write-through. The level 2 cache is on-chip, shared between the cores. It is a 1.9 Mbyte combined data and instruction cache, with 128 byte lines and is 10-way set associative and write-back. The level 3 cache is 36 Mbytes, off-chip and is shared between the 2 cores. It has 256 byte lines, and is 12-way set associative and write-back. • Interconnect: IBM High Performance Switch (HPS). Each eServer node has two network adapters and there are two links per adapter, making a total of four links between each of the frames and the switch network. • I/O: 72 Tbytes of disk running GPFS. Connected to computes nodes via HPS Software details: • OS: AIX 5.3 • Compiler versions: IBM XL Fortran compiler 10.01.0000.0007
Overall porting result	Successful
General comments	1. Problem size and cores number Problem size 682 and 1364 scaling with 253 and 496 cores were benchmarked for D6.2.2. 2. Compiling The compiling command is suggested inside the HELIUM source code as below: mpxlf90_r -qlanglvl=extended -qfree=f90 -q64 -qrealsize=8 -O4 -qarch=pwr5 -qtune=pwr5 -qessl -qsuffix=f=f90 helium.f90 -o helium Note that never use –O5.

	3. Running On HPCx, when running the helium, the max stack limit is 400MB. When the problem size is very large, the default memory size per core (i.e. 2GB) may not be enough and therefore cause the job running failed. Specify less number of tasks per node by adding the following line in the job script: #@ tasks_per_node = <n> where <n> is the tasks number assigned to each node that is less than 16, e.g. 10. In that case, the total cores number used is the same, but the memory size per core is increased.
Porting report on programming language constructs in general	HELIUM code is written in FORTRAN 90 and can be compiled using the IBM XL MPI Fortran compiler.
Porting report on libraries used	HELIUM can be linked with the ESSL scientific routine library on HPCx.
Porting report on parallelisation method	Using IBM XL Fortran MPI compiler.
Porting report on IO	The IO frenquency was specified in the test cases' source code: for the benchmark tests with 40 time steps, the output was writen out once every 10 time steps; for the benchmark tests with 80 time steps, the output was writen out once every 20 time steps.
PORTING REPORT	**For BlueGene/P @ STFC**
Porting platform	IBM BlueGene/P (MPP-BG/P) (This platform is not a prototype, but related to the prototype JeGene BlueGene/P.)

Details porting platform	Hardware details:
	
	• Model: IBM Blue Gene / P
	• Proc Type: PowerPC 450 850 MHz
	  o double precision, dual pipe floating point acceleration on each core (3.4 GFlops)
	• Clock rate: 850 MHz
	• Total Cores: 4096
	• Cores Per Chip: 4
	• Chips per Compute Card (Node): 1
	• Memory per core: 512MB.
	• Total Memory: 2048 Gbytes
	• Caches:
	  o Private 32 KB per core L1 cache
	  o Private 14 stream prefetching per core L2 cache
	  o Shared 8MB L3 cache
	• Interconnect: Proprietary 3D Torus
	• 32 I/O nodes ~4 Tbytes disk
	
	Software Details
	
	• OS Version: Linux 2.6.16.46-0.12-ppc64
	• Fortran Compilers:
	  o IBM XLF v11.1 (mpixlf90)
	  o GNU 4.1.2
	• C Compilers:
	  o IBM XLC v9.0
	  o GNU GCC v4.1.2
	Libraries: Essl, Blas

Overall porting result	Successful
General comments	Porting to BG/P was carried out by cross-compilation.

	Basic Compilation flags used are -qlanglvl=extended -qfree=f90 -qrealsize=8 -qsuffix=f=f90 –qessl For optimised performance add the options -O3 -qarch=450d -qtune=450 Notes 1. Compiling with -qessl gives improved performance over linking explicitly to the Essl library 2. Loadleveler batch jobs require the setting #@ stack_limit = 200MB Helium benchmarks with large local memory overheads may fail in VN mode, due to lack of available memory per core.
Porting report on programming language constructs in general	The HELIUM code is written in FORTRAN 90 and can be compiled using the IBM XL MPI Fortran compiler (with -qlanglvl=extended).
Porting report on libraries used	Engineering and Scientific Subroutine Library (ESSL) (see note in General comments above)
Porting report on parallelisation method	The code uses MPI. The mpixlf90 compiler configuration script automatically links in the appropriate mpi libraries and header files.
Porting report on IO	The IO/time step frequency can be specified in the test cases' source code. For all benchmark tests, the frequency was set to write output every 20 time steps for a total of 80 time steps.
PORTING REPORT	**For Hector X2**
Porting platform	Cray X2 (Vector) (This platform is not a prototype, but related to the prototype Cray XT4 and vector NEC SX9.)
Details porting platform	Hardware: • Model: Cray X2 (vector) • Proc Type: X2 vector processors • Clock rate: 1.6GHz • Proc per node: 4 • Total cores: 112 • Vector pipes per processing unit: 8 • Memory per processing unit: 7.5 GB/processing unit • Interconnect: Black Widow interconnection network Software details: • Compiler version: Cray compilers 6.0.0.5
Overall porting result	Successfully
General comments	1. Problem size and cores number The problem size 308, 770 and 1540 were benchmarked with (28), 66, 105 cores. 2. Compiling Only the Cray compiler can be used for the compiling on X2: To use modules for X2: module purge module use /opt/ctl/modulefiles module add PrgEnv-x2 module add pbs With the module PrgEnv-x2 loaded: ftn –O 3 helium.f90 -o helium The current version code then needs some minor modifications because of the FORTRAN 90 syntax checking. Note that one

	module can not be used in both the routine and its subroutines which are defined inside the routine body. 3. Running To run the job using X2, the job script must contains: #PBS -q vector #PBS -A z01-X2 And add -a x2 in the aprun command.
Porting report on programming language constructs in general	HELIUM code is written in FORTRAN 90 and can be compiled using the Fortran 90 compiler.
Porting report on libraries used	
Porting report on parallelisation method	Using Cray f90 compiler for the MPI code compiling.
Porting report on IO	The IO frequency was specified in the test cases' source code. On X2, the total time step was set as 80 and the output was written out once every 20 time steps.
PERFORMANCE RESULTS	**For Louhi @ CSC**
Execution platform	Cray XT4/XT5
Details execution platform	Same with the porting platform
Performance details	On the Cray XT4 part, HELIUM scales well up to 1540 cores. 1GB/core memory is enough for most of the running with problem size 770 and 1540. For the running requires more memory size per core, use the cores with 2GB/core memory instead but will have a very poor performance. Some executions were not available due to that. The performance on XT4 and XT5 are similar. Using the PGI compiler and PathScale compiler (with the general optimisation flags used, see above of porting) have roughly similar performance. Please find the detailed results data in the attached results report *helium_cray.xls*.
PERFORMANCE RESULTS	**For HECToR XT4**
Execution platform	Cray XT4 (dual core)
Details execution platform	Same with the porting platform
Performance details	Scale well up to 1540 cores. Performance results are similar to the results on Louhi Cray XT4. For the running requires more memory size per core, place 1 task per core rather than fully allocation on the node, but performance was very poor. Using PathScale compiler can have a better performance compared with using PGI compiler (with the general optimisation flags used, see above of porting). The final results reported are those using PathScale compiler. Please find the detailed results data in the attached results report *helium_hector.xls* and *helium_cray.xls*.
PERFORMANCE RESULTS	**For Huygens @ SARA**
Execution platform	IBM Power6
Details execution platform	Same with the porting platform
Performance details	Code scale well up to 1540 cores and can have a roughly 85% efficiency with 1540 cores (related to 630 cores). Link with the ESSL lib will improve the code performance. Please find the detailed results data in the attached results report

	helium_pwr6.xls.
PERFORMANCE RESULTS	**For HPCx**
Execution platform	IBM Power5
Details execution platform	Same with the porting platform
Performance details	Please refer to D6.2.2 and find the detailed results data in the attached results report *helium_hpcx.xls*.
PERFORMANCE RESULTS	**For BG/P @ STFC**
Execution platform	IBM BG/P
Details execution platform	Same with the porting platform
Performance details	Please find the detailed results in the attached results report *helium_bgp.xls*.
PERFORMANCE RESULTS	**For Hector X2**
Execution platform	Cray X2 (vector)
Details execution platform	Same with the porting platform
Performance details	Code scale up to 105. Performance is OK but not as good as expected. The expected results should be around 5 times faster than the Hector XT4 but actually is only around 3 times. Please find the detailed results data in the attached results report *helium_cray.xls*.
RECOMMENDATIONS	**For Petascaling and optimisation on Cray XT4/XT5 (Louhi)**
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	Porting HELIUM is straightforward and the code scale well to 1540 cores on Louhi, so there is a possibility that it can reach Petascaling potential. However, the memory size per core could be a big bottleneck and therefore need to select suitable parameters values for a required problem size and cores number. Currently Louhi seems not allow user to half populate the node or quarter populate the node. If this is allowed by the system, it may be helpful for the scaling. Amount of pm's: 4pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	There are a number of subroutines having long loops that can be further optimised. Further compiler optimisation, i.e. using special flags or linking with special libs, may improve the performance. Some performance profiling may help to find out the further bottleneck or potential problem for scaling. Amount of pm's: 4~6pm
RECOMMENDATIONS	**For Petascaling and optimisation on Cray XT4 (HECToR)**
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	The code scale well to 1540 cores on HECToR, so there is a possibility that it can reach Petascaling potential. See other more comments for Cray XT4/XT5 (Louhi).

	It should be noticed from the MPI profiling results (from the D6.2.2) that some synchronisation time is quite expensive. Reducing this part will be helpful for the scaling improvement. Amount of pm's: 4pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	There are a number of subroutines having long loops. For example, the most expensive user routine Incr_Result_w_1_Over_R12_terms of module Local_Ham_Matrix is mainly a loop. These routines can be further optimised. Further compiler optimisation, i.e. using special flags or linking with special libs, may improve the performance. Amount of pm's: 6pm
RECOMMENDATIONS	**For Petascaling and optimisation on IBM Power6 (Huygens @ SARA)**
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	HELIUM is easy porting and scales well up to 1540 cores. The parallel performance scaling speed tailed off with the increasing cores number but is acceptable when cores number up to 1540, so there is a possibility that it can reach Petascaling potential. Amount of pm's: 3pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Using proper compiler flags to build the HELIUM code for optimisations or linking with selected libraries could be quite useful for improving the scaling and performance, but need more investigation for the compilers used for each platform. Do some performance profiling, e.g. MPI Trace and user routine execution time profiling, to help find out the bottleneck for the scaling performance. Then some focused effort will be required for those bottlenecks. Amount of pm's: 4~6pm.
RECOMMENDATIONS	**For Petascaling and optimisation on IBM Power5 (HPCx)**
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	MPI_Barrier appears to be a main cause of slow performance and occupies as much time as some of the heavily used subroutines. It may be possible to reduce these calls. The code currently scales well to 496 processors and with a reduction of MPI_Barrier calls, this code improve further. It should be noticed that when using 496 cores for the problem size 1364 test case benchmarking, the pure running (without any profiling) of HELIUM succeed, but the MPI profiling results had no output. Not clear about the reason, but can not avoid a potential problem of running HELIUM using large number cores for large problem size. This should be investigated more. Amount of pm's: 4pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	The majority of time is spent in 4-6 routines, so a focused effort is possible. Amount of pm's: 6pm.
RECOMMENDATIONS	**For Petascaling and optimisation on IBM BG/P (BlueGene @**

	STFC)
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	Performance scales well up to 990 cores so the code has Petascaling potential. Profiling Tools are not currently available on STFC's BG/P. Further insight will be gained from runs on Jugene BG/P, were detailed performance analysis tools can be applied. Amount of pm's: 3pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Helium outputs large amounts of temporary data and hardware parallel I/O performance on BG/P is relatively limited. There may be software changes that can be made to improve this. Profiling Tools are not currently available on STFC's BG/P. Further insight will be gained from runs on the Jugene BG/P, where detailed performance analysis tools can be applied. Amount of pm's: 3pm
RECOMMENDATIONS	**For Petascaling and optimisation on Cray X2 (Hector)**
Expected potential for Petascaling	Unknown
Expected effort to reach Petascaling potential	Only 112 cores are available on the current Hector X2, so it still unknown whether X2 has the petascaling potential. Benchmarking on more cores is necessary. Amount of pm's: unknown
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	The current performance is not quite ideal. Can using profiling tool, e.g. CrayPat, to find out the bottleneck. Amount of pm's: 4~6pm

7.2.15 *TRIPOLI4*

Jean-Christophe Trama
CEA

GENERAL	
Name of Code, Abbreviation	TRIPOLI®
Application area(s)	Nuclear energy : core physics, radiation protection, criticality
Key numerical method(s)	Monte Carlo
Origin (developers, institute)	CEA Saclay SERMA R&D unit
Current developers	CEA Saclay SERMA R&D unit
Contact person	Jean-Christophe Trama, jean-christophe.trama@cea.fr
License policy	code available from the NEA databank (www.nea.fr) and RSICC (www-rsicc.ornl.gov) TRIPOLI-4 version 3 is available for all use TRIPOLI-4 version 4 is available for R&D and teaching only
Copyright	TRIPOLI® is a registered trade mark of CEA
Usage rules (within PRACE, outside PRACE, …)	cf NEA and RSICC licences, the code may also be used under specific licences granted by CEA
PRACE INFORMATION	
BCO: name, email, institute	Jean-Christophe Trama, jean-christophe.trama@cea.fr, CEA Saclay

Contributors (PRACE partners)	GENCI
Targeted hardware platforms as in BCO list	Thin nodes, BLG
CODE STATISTICS	
Programming language(s)	C++
Amount of source lines	400000
Libraries	no external lib.
Parallellization method	native (TCP/IP sockets)
Development platform(s)	Linux, Unix (DEC/OSF1, SUN/SOLARIS, IBM/AIX)
IO characteristics	write once every several time steps. The output frequency needs to be specified in the input deck.
PORTING REPORT	For CEA CCRT Platine
Porting platform	BULL Novascale 3045 (Thin node)
Details porting platform	Hardware details: • 932 computational nodes – 26 I/O nodes • Each node; ItaniumII, Montecito (double core – 1.6Ghz) – 24 Go RAM • Interconnect: Infiniband • Storage: 420 TB- Lustre Software details • BULL Advanced Server 4 • Intel 10.1 – g++ v4
Overall porting result	Successful
General comments	A production run of TRIPOLI has been successfully launched on CEA CCRT Platine. A whole nuclear reactor core has been calculated on 1000 processors with a very good scaling factor (up to 80 %) with a reasonable amount of effort.
Porting report on programming language constructs in general	g++ compiler is okay, first test show good results for the intel native compiler.
Porting report on libraries used	no external lib.
Porting report on parallelisation method	embarrassingly parallel ! (Monte Carlo method, sets of independent particules)
Porting report on IO	standard
PERFORMANCE RESULTS	CEA CCRT Platine
Execution platform	same as porting platform
Details execution platform	
Performance details	The input set is describing a whole nuclear core, 1000 cores, linear speed up up to 1000 core, 80 % efficiency.
RECOMMENDATIONS	
Expected potential for Petascaling	high
Expected effort to reach Petascaling potential	around 6 pm to modify the information exchange architecture to run more than 1000 proc. Specific effort may be needed on IO for very large number of results.
Expected potential for Optimisation	The code is already very optimised for parallel operation (direct parallelisation of sets of independent particles, intrinsic to the Monte Carlo method)
Expected effort to reach Optimisation potential	low.

7.2.16 *PEPC*

Lukas Arnold
FZJ

GENERAL	
Name of Code, Abbreviation	Pretty Efficient Parallel Coulomb solver, PEPC
Application area(s)	Plasma physics
Key numerical method(s)	Tree-code for rapid computation of long-range Coulomb forces in N-body particle systems
Origin (developers, institute)	Paul Gibbon, FZJ
Current developers	Paul Gibbon
Contactperson	Paul Gibbon
License policy	freely available
Copyright	Forschungszentrum Jülich GmbH
Usage rules (within PRACE, outside PRACE, …)	to be clarified
PRACE INFORMATION	
BCO: name, email, institute	Lukas Arnold l.arnold@fz-juelich.de FZJ
Contributors (PRACE partners)	none
Targeted hardware platforms as in BCO list	MPP-BG/P, MPP-Cray, FatNode-Pwr6
CODE STATISTICS	
Programming language(s)	Fortran 90
Amount of source lines	24500
Libraries	none
Parallellization method	MPI
Development platform(s)	PWR6-JuMP
IO characteristics	read at start, the output frequency can be choosen in the parameter file (run.h); no output is performed in the benchmarks included in this report
PORTING REPORT	For JuMP(FZJ)
Porting platform	FatNode-Pwr6
Details porting platform	**Hardware:** – 14 SMP nodes with 32 SMT processors each (total 448) – Processortype: Power6 4.7 GHz – Overall peak performance: 8.4 Teraflops – Linpack: 5.4 Teraflops – Main memory: 14 x 128 Gbytes (aggregate 1.8 TB) – InfiniBand (MPI communication) – 10 Gigabit Ethernet (I/O) – 1 Gigabit Ethernet (cluster management) – Disk capacity for system data: 4.5 TBytes

	– Disk capacity for user data: 1.0 PBytes – Migration storage for user data: 1.5 PBytes **Software:** – Operating system: AIX 5.3 – Operating mode: interactiv and batch – Compiler versions: IBM AIX compiler (xlf 9.1/10.1/11.1/12.1; xlc 7.0/8.0/9.0/10.1)
Overall porting result	Successful
General comments	Porting PEPC to JuMP is basic, because JuMP is a development platform. JuBE is used for the benchmarking. The platform dependent settings need to be set in makefile.defs, which result in the following compilation command on JuMP: `mpxlf90_r -q64 -qtune=pwr6 -qarch=pwr6 -O3` This settings are set by JuBE. The main makefile calls the makefiles in the directories lpepcsrc and pepc-b. The compiled objects and a small wrapper (JuBE-PEPC.c) are linked together, no additional libraries are needed. The problem size, i.e. the number of particles, can be set in the configuration file run.h, which needs to be in the executing directory. This nuber is set by JuBE. Nothing in the PEPC code/config needs to be set to choose the nuber of MPI processes. PEPC distributes the computation on all available processes automatically.
Porting report on programming language constructs in general	PEPC compiles directly with the IBM AIX fortran compiler.
Porting report on libraries used	none used
Porting report on parallelization method	Using AIX MPI compiler wrapper.
Porting report on IO	The output is switched off (idump parameter in run.h).
PORTING REPORT	huygens(SARA)
Porting platform	FatNode-Pwr6
Details porting platform	**Hardware:** – 104 SMP nodes with 32 SMT processors each (total 3328) – Processortype: Power6 4.7 GHz – Overall peak performance: 60 Teraflops – Main memory: 83 x 128 Gbytes + 18 x 256 Gbytes

	(aggregate 15.2 TB) – InfiniBand (MPI communication) – Disk capacity: 700 TBytes **Software:** – Operating system: Linux (SuSE) – Operating mode: interactiv and batch – Compiler versions: IBM AIX compiler (xlf 11.1; xlc 9.0)
Overall porting result	Successful
General comments	Porting PEPC to huygens is basic, because it is very similar to JuMP. The platform dependent settings need to be set in makefile.defs, which result in the following compilation command on JuMP: `mpfort -qfree=f90 -q64 -qtune=pwr6 -qarch=pwr6 -O3` This settings are set by JuBE. The main makefile calls the makefiles in the directories lpepcsrc and pepc-b. The compiled objects and a small wrapper (JuBE-PEPC.c) are linked together, no additional libraries are needed. The problem size, i.e. the number of particles, can be set in the configuration file run.h, which needs to be in the executing directory. This nuber is set by JuBE. Nothing in the PEPC code/config needs to be set to choose the nuber of MPI processes. PEPC distributes the computation on all available processes automatically.
Porting report on programming language constructs in general	PEPC compiles directly with the IBM AIX fortran compiler.
Porting report on libraries used	none used
Porting report on parallelization method	Using AIX MPI compiler wrapper.
Porting report on IO	The output is switched off (idump parameter in run.h).
PORTING REPORT	For JUGENE
Porting platform	MPP-BG/P
Details porting platform	**Hardware:** – 16384 compute nodes with 4-way SMT processors (total 65536 processors) – Processortype: PowerPC 450 850 MHz

	– Overall peak performance: 223 Teraflops – Linpack: 167 Teraflops – Main memory: 2 Gbytes per node (aggregate 32 TB) – Three-dimensonal torus (compute nodes) – Global tree / Collective network (compute nodes, I/O nodes) – 10 Gigabit Ethernet (I/O) – Disk capacity for system data: 4.5 TBytes – Disk capacity for user data: 1.0 PBytes – Migration storage for user data: 1.5 PBytes **Software:** – Operating system: CNL – Operating mode: interactive and batch – Compiler versions: IBM AIX compiler (xlf 9.1/10.1/11.1/12.1; xlc 7.0/8.0/9.0/10.1)
Overall porting result	successful
General comments	All settings are set by PABS.
Porting report on programming language constructs in general	
Porting report on libraries used	
Porting report on parallelization method	
Porting report on IO	
PORTING REPORT	For louhi
Porting platform	MPP-Cray
Details porting platform	**Hardware:** – CPU: 2.3 GHz AMD Opteron 64 bit quad-core processors – Number of nodes: 1012 computing nodes – Memory: 1 GB or 2 GB memory per core. – Interconnect: Seastar 2+ – Set up physically in 11 cabinets. – Architechture: MPP – I/O subsystem: Lustre, 70 TB **Software:**

	– Operating system: CNL – Operating mode: interactiv and batch – Compiler versions: PGI compiler version 7.2.4
Overall porting result	successful
General comments	Porting PEPC to louhi is streight forward. The platform dependent settings need to be set in makefile.defs, which result in the following compilation command on louhi: `ftn -O3` The architecture options are set by default; this settings are set by JuBE. The main makefile calls the makefiles in the directories lpepcsrc and pepc-b. The compiled objects and a small wrapper (JuBE-PEPC.c) are linked together, no additional libraries are needed. The problem size, i.e. the number of particles, can be set in the configuration file run.h, which needs to be in the executing directory. This nuber is set by JuBE. Nothing in the PEPC code/config needs to be set to choose the nuber of MPI processes. PEPC distributes the computation on all available processes automatically.
Porting report on programming language constructs in general	PEPC compiles directly with the PGI fortran compiler.
Porting report on libraries used	none used
Porting report on parallelization method	Using PGI MPI compiler wrapper.
Porting report on IO	The output is switched off (idump parameter in run.h).
PERFORMANCE RESULTS	For JuMP
Execution platform	FatNode-Pwr6
Details execution platform	Same with the porting platform
Performance details	
PERFORMANCE RESULTS	For huygens
Execution platform	FatNode-Pwr6
Details execution platform	Same with the porting platform
Performance details	
PERFORMANCE RESULTS	For platform JUGENE
Execution platform	MPP-BG/P
Details execution platform	Same with the porting platform
Performance details	

PERFORMANCE RESULTS	For platform louhi
Execution platform	MPP-Cray
Details execution platform	Same with the porting platform
Performance details	
RECOMMENDATIONS	For Petascaling and optimization on all platforms
Expected potential for Petascaling	High (all platforms)
Expected effort to reach Petascaling potential	the scalability of PEPC is in the moment limited to up to 4k - 8k processes due to the used memory structure, i.e. the memory requirements grow with the number of processes. The developer are working on this urgent problem, whereas the developement time scales are large. 0PM
Expected potential for Optimization	Medium
Expected effort to reach Optimization potential	0PM

7.2.17 *GPAW*

Jussi Enkovaara
CSC Finland

GENERAL	
Name of Code, Abbreviation	GPAW
Application area(s)	Nanoscience, materials science
Key numerical method(s)	finite differences, sparse solvers
Origin (developers, institute)	J. J. Mortensen, Technical Univ. Denmark
Current developers	Several developers in CSC and in universities in Finland, Denmark, Sweden and Germany.
Contactperson	J. J. Mortensen, Technical Univ. Denmark
License policy	GPL
Copyright	GPL
Usage rules (within PRACE, outside PRACE, …)	
PRACE INFORMATION	
BCO: name, email, institute	Jussi Enkovaara, jussi.enkovaara@csc.fi, CSC
Contributors (PRACE partners)	CINECA
Targeted hardware platforms as in BCO list	MPP-BG, MPP-Cray, FatNode-Pwr6
CODE STATISTICS	
Programming language(s)	Python, C
Amount of source lines	42000 + 10000
Libraries	LAPACK, BLAS
Parallellization method	MPI
Development platform(s)	
IO characteristics	typically write at end…
PORTING REPORT	For platform #1
Porting platform	MPP-Cray
Details porting platform	Cray XT4 2.1 GHz Quad-core opteron, PGI 7.2.2, ACML 4.1.0, xt-MPT 3.0.1
Overall porting result	Successful
General comments	

Porting report on programming language constructs in general	Static build of python required
Porting report on libraries used	
Porting report on parallelisation method	
Porting report on IO	
PERFORMANCE RESULTS	For platform #1
Execution platform	MPP-Cray
Details execution platform	
Performance details	See https://trac.csc.fi/pracewp6-gpaw/wiki/Deliverable_6.2.2%3A
RECOMMENDATIONS	For Petascaling and optimisation on platform #1
Expected potential for Petascaling	medium/high
Expected effort to reach Petascaling potential	small
Expected potential for Optimisation	medium
Expected effort to reach Optimisation potential	medium
PORTING REPORT	For platform #2
Porting platform	MPP-BG
Details porting platform	Blue Gene/P 32-bit PowerPC 450 core 850 MHz
Overall porting result	Succesfull
General comments	There were some small problems due to features/bugs in BlueGene's math-library. For example, with certain input the pow function took huge amount of time, and these problems required workarounds in the source code. Cross-compilation in Blue Gene was challenging especially as by accident it was possible to use wrong libraries (i.e. libraries not build for the compute nodes) without clear problems. With certain input code behaved correctly, while other input resulted in undefined behavior.
Porting report on programming language constructs in general	Python itself was not a major problem after all, but problems were related just to C-code and C-libraries
Porting report on libraries used	
Porting report on parallelisation method	
Porting report on IO	
PERFORMANCE RESULTS	For platform #2
Execution platform	MPP-BG
Details execution platform	
Performance details	Only initial tests have been run
RECOMMENDATIONS	For Petascaling and optimisation on platform #2
Expected potential for Petascaling	
Expected effort to reach Petascaling potential	
Expected potential for Optimisation	
Expected effort to reach Optimisation potential	
PORTING REPORT	For platform #3
Porting platform	FatNode-Pwr6
Details porting platform	IBM dual-core Power6, 4.7 GHz
Overall porting result	Successful
General comments	
Porting report on programming language constructs in general	
Porting report on libraries used	
Porting report on parallelisation method	
Porting report on IO	

PERFORMANCE RESULTS	For platform #3
Execution platform	FatNode-Pwr6
Details execution platform	
Performance details	Only initial tests have been run
RECOMMENDATIONS	For Petascaling and optimisation on platform #3
Expected potential for Petascaling	
Expected effort to reach Petascaling potential	
Expected potential for Optimisation	
Expected effort to reach Optimisation potential	

7.2.18 *ALYA*

Guillaume Houzeaux and Raúl de la Cruz
BSC-CNS

GENERAL	
Name of Code, Abbreviation	ALYA
Application area(s)	Computational Mechanics
Key numerical method(s)	Sparse linear algebra, Unstructured mesh
Origin (developers, institute)	G. Houzeaux, M. Vázquez, BSC-CNS (Spain)
Current developers	G. Houzeaux, M. Vázquez
Contact person	<guillaume.houzeaux@bsc.es>, <mariano.vazquez@bsc.es>, <josem.cela@bsc.es>
License policy	This code has access restrictions: permission for use must be obtained from Contact persons.
Copyright	BSC-CNS
Usage rules (within PRACE, outside PRACE, …)	Free within PRACE. No disclosure/No propagation of the source code.
PRACE INFORMATION	
BCO: name, email, institute	Raúl de la Cruz <raul.delacruz@bsc.es> BSC-CNS, (Spain)
Contributors (PRACE partners)	
Targeted hardware platforms as in BCO list	Cell
CODE STATISTICS	
Programming language(s)	F90
Amount of source lines	200000
Libraries	Metis
Parallellization method	MPI/OpenMP

Development platform(s)	Marenostrum (Power PC970), Windows, BlueGene L/P, Linux clusters
IO characteristics	Write once every several time steps. The output frequency needs to be specified in the source code by assign a particular parameter.
PORTING REPORT	CELL
Porting platform	Maricel
Details porting platform	Hardware: • Model: IBM Cell/B.E. cluster • Proc Type: Cell/B.E. • Clock rate: 3.2 GHz • Total Cores: 72 • Cores Per Chip: 1 PPU + 8 SPU's • Cores Per Node: 2 PPU's + 2x8 SPU's • Memory per core: 4 GB • Total Memory: 4 GB • Cache: L1 (32 KB)/L2 (2 MB) associated to PPU. Local store associated to SPU'S. • Interconnect: InfiniBand • I/O: Hypernode Software details: • OS version: Redhat 5.2, Fedora 7 • Compiler versions: ppu-gcc, spu-gcc (SDK3.1)
Overall porting result	Basic porting: code is running on PPE. Porting to one SPE almost done. Full porting (use all SPE's) is in progress.
General comments	Missing wrappers for some libraries of the f90 compiler
Porting report on programming language constructs in general	
Porting report on libraries used	METIS: no specific problem
Porting report on parallelisation method	Porting has been carried out on only one node and one SPE's Vectorization still to be done. Full prototype will be available in december.
Porting report on IO	
PERFORMANCE RESULTS	
Execution platform	
Details execution platform	
Performance details	
RECOMMENDATIONS	
Expected potential for Petascaling	Medium to high

Expected effort to reach Petascaling potential	Alya is already running on thousands of processors. Explicit solver: perfect speedup up to 5000 CPU's on MareNostrum and BG/L. Implicit solver: speedup between 40% and 95% efficiency depending on the test case. IO strategy must be specially designed for petascaling: NETCDF format may be chosen. Amount of pm's: 2pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Element loops have been rewritten to prepare the code for vectorization. More work must be done. Relation memory access/computing still too high. For explicit solver, work concentrate on 2 subroutines: 2 element loops. For implicit solver, work should concentrate on 4 subroutines (2 element loops+2 algebraic solvers): divide elements loops. For solvers: Cell version of some iterative and direct algebraic solvers have already been ported to Cell (Linpack). Amount of pm's: 4pm.

7.2.19 *SIESTA*

M.Cytowski, M. Filocha, M.Szpindler
PCSS Poland

GENERAL	
Name of Code, Abbreviation	SIESTA
Application area(s)	Ab initio molecular dynamics simulations of molecules and solids.
Key numerical method(s)	Dense matrices, FFTs, ..
Origin (developers, institute)	Departamento de Fisica de la Materia Condensada, Universidad Autonoma de Madrid Jose M. Soler
Current developers	• Emilio Artacho (Department of Earth Sciences, University of Cambridge) • Julian Gale (Nanochemistry Research Institute, Department of Applied Chemistry, Curtin University of Technology) • Alberto Garcia (Institut de Ciencia de Materials, CSIC - Universidad Autónoma de Barcelona) • Javier Junquera (Departamento de Ciencias de la Tierra y Física de la Materia Condensada, Universidad de Cantabria) • Richard M. Martin (Department of Physics, University of Illinois at Urbana-Champaign) • Pablo Ordejon (Institut de Ciencia de Materials, CSIC - Universidad Autónoma de Barcelona) • Daniel Sanchez-Portal (Unidad de Física de Materiales, Universidad del País Vasco) • Jose M. Soler (Departamento de Física de la Materia Condensada, Universidad Autónoma de Madrid)

Contact person	siesta@uam.es
License policy	Tree kinds of licenses: • Academic License for Individuals (register required; code modification allowed) • Academic License for Computing Centers (register required; only licensed and registers users; unlimited number of workstations; code modifications allowed) • Non academic – commercial use through Nanotec Electronica (http://www.nanotec.es)
Copyright	The Copyright-Holder is The Fundacion General de la Universidad Autonoma de Madrid (FGUAM – http://ewan.fg.uam.es)
Usage rules (within PRACE, outside PRACE, …)	Free within PRACE, not free outside PRACE
PRACE INFORMATION	
BCO: name, email, institute	M. Filocha, m.filocha@icm.edu.pl, PCSS, Poland
Contributors (PRACE partners)	
Targeted hardware platforms as in BCO list	Cell
CODE STATISTICS	
Programming language(s)	Fortran 90
Amount of source lines	More than 105 000
Libraries	Metis, BLAS, SCALAPACK
Parallellization method	MPI
Development platform(s)	Prepared compiling directions for following architectures: Cray XT-3, Cray T3e, Itanium Cluster, Altix, IBM SP2 with PPC3, MacOS X, JS21 (Marenostrum) and other
IO characteristics	Read at start, write periodically
PORTING REPORT	For platform #1 (repeat for platform #2, #3, …)
Porting platform	Cell
Details porting platform	3 IBM blades located in BladeCenter H chasis: • 2xQS21 • 1xLS21 Hardware configuration of the target platform: • Model: IBM QS21 blades • CPU Type: Cell BE • Clock rate: 3.2 GHz • Total Cores: 36 • Cores per Chip: 9 • Cores per Node: 18 • Memory per Node: 2 GB • Total Memory: 4 GB • Infiniband Software details: • OS: Fedora 8 • Compiler version: IBM XLF for Cell 11.1
Overall porting result	Successful
General comments	We defined the porting step to be a process of compiling Siesta code on PPE with the use of IBM XLF compiler. In this terminology we see the usage of SPEs and vectorization to be an optimisation step. This is currently work in progress. One compiler bug has been found. The code was modified in order to prevent compiler from crash. The bug reason was segmented out of the code and will be reported to IBM developer groups.

Porting report on programming language constructs in general	Siesta is written in Fortran 90. Dynamic memory allocation is used. Cell XLFortran PPU compiler crashed on some of the specific Fortran 90 constructs (to be reported elsewhere).
Porting report on libraries used	Till now the following libraries where used: • Metis v. 4.0.1 – downloaded from http://glaros.dtc.umn.edu/gkhome/metis/metis/download and compiled with the use of GCC on PPU • LAPACK – standard Fedora package was used (this will be probably exchanged with specific Cell Lapack implementation) • BLAS – standard Fedora package was used (this will be probably exchanged with specific Cell Blas implementation) The main problem we encountered during the PPU porting step was that specific Cell Lapack library is not a full Lapack implementation. The complex functions are still missing. These will have to be specially linked from other source.
Porting report on parallelisation method	The parallelisation method used in Siesta is MPI.
Porting report on IO	
PERFORMANCE RESULTS	For platform #1 (repeat for platform #2, #3, …)
Execution platform	Cell
Details execution platform	
Performance details	The standard Siesta benchmark suite is used (Tests/ directory).
RECOMMENDATIONS	For Petascaling and optimisation on platform #1 (repeat for platform #2, #3, …)
Expected potential for Petascaling	Medium
Expected effort to reach Petascaling potential	Approximately 2 or 3 pm's. The parallel version should be tested across few QS21 blades. QS22 blades should be also taken into consideration/tests.
Expected potential for Optimisation	Large
Expected effort to reach Optimisation potential	Approximately 2 pm's. The code should be implemented on the whole Cell BE architecture (PPE + 8 SPEs). Some of the library calls could be taken to SPE side. It is highly possible that some parts of the code will have to be rewritten to C (for SPE compatibility) and called from within Fortran code.

7.2.20 *BSIT*

Mauricio Araya
BSC-CNS

GENERAL	
Name of Code, Abbreviation	BSIT
Application area(s)	Computational Geophysics
Key numerical method(s)	Explicit finite difference scheme, Structured mesh
Origin (developers, institute)	M. Araya, M. Hanzich, F. Rubio, A.C. Lesage BSC-CNS (Spain)
Current developers	M. Araya, M. Hanzich, F. Rubio, A.C. Lesage BSC-CNS (Spain)
Contact person	<mauricio.araya@bsc.es>, <mauricio.hanzich@bsc.es>, <josem.cela@bsc.es>

License policy	This code has access restrictions: no permission for use.
Copyright	Repsol YPF and BSC-CNS
Usage rules (within PRACE, outside PRACE, …)	No disclosure/No propagation of the source code.
PRACE INFORMATION	
BCO: name, email, institute	Mauricio Araya <mauricio.araya@bsc.es> BSC-CNS, (Spain)
Contributors (PRACE partners)	
Targeted hardware platforms as in BCO list	Cell/B.E.
CODE STATISTICS	
Programming language(s)	C and F90
Amount of source lines	40000
Libraries	Librt
Parallellization method	MPI
Development platform(s)	Maricel Cell/B.E.
IO characteristics	Write once every several time steps. The output frequency needs to be specified in the source code by assign a particular parameter.
PORTING REPORT	CELL
Porting platform	Maricel
Details porting platform	Hardware: • Model: IBM Cell/B.E. cluster • Proc Type: Cell/B.E. • Clock rate: 3.2 GHz • Total Cores: 72 • Cores Per Chip: 1 PPU + 8 SPU's • Cores Per Node: 2 PPU's + 2x8 SPU's • Memory per core: 4 GB • Total Memory: 4 GB • Cache: L1 (32 KB)/L2 (2 MB) associated to PPU. Local store associated to SPU'S. • Interconnect: InfiniBand • I/O: Hypernode Software details: • OS version: Redhat 5.2, Fedora 7 • Compiler versions: ppu-gcc, spu-gcc (SDK3.1)
Overall porting result	Full porting done

General comments	Performance analysis underway
Porting report on programming language constructs in general	
Porting report on libraries used	General purpose library utilization
Porting report on parallelisation method	MPI not thoroughly tested
Porting report on IO	Good expected results
PERFORMANCE RESULTS	
Execution platform	QS20 blades
Details execution platform	
Performance details	30% of the machine peak performance achieved
RECOMMENDATIONS	
Expected potential for Petascaling	High
Expected effort to reach Petascaling potential	Developments in the queue management system for the embarrassingly parallel character of the external layer of BSIT. Amount of pm's: 1pm
Expected potential for Optimisation	Medium
Expected effort to reach Optimisation potential	Optimisation has reached a good performance for this kind of algorithm (explicit finite difference). Further optimisation can be expected but in an marginal range. Amount of pm's: 2pm.

7.3 Example of Code Integration into JuBE

The following is an example of setting up a benchmark application for a single platform (EPSRC/EPCC's HPCx, an IBM Power5 system). The benchmark is a small scaling example (using two processor counts) of HELIUM.
The first step was to create add HPCx to the platform.xml file:

```
<platform name="IBM-SP5-HPCx">
    <params
        make        = "make"
        rm          = "rm -f"
        ar          = "ar"
        arflags     = "-rs"
        ranlib      = "/usr/bin/ranlib"
        cpp         = "/usr/bin/cpp"
        cppflags    = "-P"
        f77         = "xlf_r"
        f77flags    = "-qtune=pwr5 -qarch=pwr5"
        f90         = "xlf90_r"
        f90flags    = "-qtune=pwr5 -qarch=pwr5"
        cc          = "xlc_r"
        cflags      = "-qtune=pwr5 -qarch=pwr5"
        cxx         = "xlC_r"
        cxxflags    = "-qtune=pwr5 -qarch=pwr5"
        mpi_f90     = "mpxlf90_r"
        mpi_f77     = "mpxlf_r"
        mpi_cc      = "mpcc_r"
        mpi_cxx     = "mpCC_r"
        ldflags     = "-qtune=pwr5 -qarch=pwr5"
        mpi_dir     = ""
        mpi_lib     = ""
        mpi_inc     = ""
        mpi_bin     = ""
        blas_dir    = ""
        blas_lib    = "-lessl"
        lapack_dir  = "-L/usr/local/lib"
        lapack_lib  = "-lessl -llapack"
        fftw3_dir   = "-L/usr/local/packages/fftw/lib"
        fftw3_lib   = "-ldfftw -lm"
        fftw3_inc   = "-I/usr/local/packages/fftw/include"
        fftw2_dir   = ""
        fftw2_lib   = ""
        fftw2_inc   = ""
        netcdf3_dir = ""
        netcdf3_lib = ""
        netcdf3_inc = ""
        hdf5_dir    = "-L/usr/local/packages/hdf5/lib -
    L/usr/local/packages/hdf5/zlib/lib"
        hdf5_lib    = "-lhdf5_fortran -lhdf5 -lgpfs -lz"
        hdf5_inc    = "-I/usr/local/packages/hdf5/include"
        module_cmd  = ""
    />
</platform>

```

This creates several variables that can be used in later XML files. Step two was to create the XML files needed for HELIUM. These are:

- bench-platform.xml
- compile.xml
- prepare.xml
- execute.xml

- verify.xml
- analyse.xml

7.3.1 *bench-platform.xml*

```
1   <!--
2       PRACE Benchmark Suite
3
4       JUBE benchmark configuration schema for: HELIUM
5
6       Contact: jon@epcc.ed.ac.uk
7   -->
8   <bench name    = "HELIUM" platform= "IBM-SP5-HPCx" >
9
10  <!-- ******************************************************* -->
11
12  <benchmark name="strong" active="1">
13      <compile     cname="$platform" version="new"
14                   nblocks="`(-1+sqrt(8*$tasks+1))/2`"
15                   xlast="`1364/((-1+sqrt(8*$tasks+1))/2)`" />
16      <tasks       threadspertask="1" taskspernode="16" nodes="15.8125,31" />
17      <prepare     cname="standard" />
18      <execution   cname="$platform" iteration="1" />
19      <verify      cname="standard" />
20      <analyse     cname="standard" />
21  </benchmark>
22
23  <!-- ******************************************************* -->
24
25  </bench>
```

The above benchmark consists of two runs; one of 253 processors and one of 496 processors on the platform labelled IBM-SP5-HPCx and defined above. A code can have several benchmarks, which are kept in the same bench.xml file. Each platform has a separate bench.xml file. These should be named with the platform name, e.g. bench-IBM-SP5-HPCx.xml, bench-Cray-XT4-Louhi.xml, etc.
HELIUM requires that the size of a "block" (`nblocks`) be fixed for a certain number of processors and that 'xlast' (`xlast`) is related to the problem size and processor count. HPCx (the platform for this test) has 16 processors per node, therefore the two processor counts needed require 15.8125 and 31 nodes respectively. JuBE will multiply `threadspertask` (1), `taskspernode` (16), and nodes to work out the number of tasks for each run (line 15). This number is available as a variable, `$tasks`. As HELIUM requires that `nblocks` be set according to the number of processors in the source code, this number is set up as a variable `$nblock` on line 14. In order to keep a fixed problem size, $xlast is also altered.This will be available in later XML files.

7.3.2 *compile.xml*

For each platform that the benchmark code is to be run on, an XML definition is required. All platforms are kept in the same file.

```
26  <compilation>
27
28  <!-- predefined vars:
29     $outdir -> output directory for temporary compile files
30     $id     -> identifier of this benchmark run
31  -->
32
33  <compile cname="IBM-SP5-HPCx">
34      <!-- Specification of source files to copy into temporary build
35           directory -->
36      <src directory="./src" files="*.f90.in Makefile.in Makefile.defs.in" />
```

```
37
38      <substitute infile="helium.f90.in" outfile="helium.f90">
39          <sub from="#NBLOCKS#"         to="$nblocks" />
40          <sub from="#X_LAST#"          to="$xlast" />
41      </substitute>
42
43      <!-- Create Makefile and substitute parameters -->
44      <substitute infile="Makefile.in" outfile="Makefile">
45          <sub from="#EXECNAME#"      to="$execname" />
46          <sub from="#OUTDIR#"        to="$outdir" />
47      </substitute>
48
49      <substitute infile="Makefile.defs.in" outfile="Makefile.var">
50          <sub from="#MAKE#"          to="$make" />
51          <sub from="#RM#"            to="$rm" />
52          <sub from="#AR#"            to="$ar" />
53          <sub from="#ARFLAGS#"       to="$arflags" />
54          <sub from="#RANLIB#"        to="$ranlib" />
55          <sub from="#CPP#"           to="$cpp" />
56          <sub from="#CPPFLAGS#"      to="$cppflags" />
57          <sub from="#F77#"           to="$f77" />
58          <sub from="#FFLAGS#"        to="-q64 $f77flags" />
59          <sub from="#F90#"           to="$f90" />
60          <sub from="#F90FLAGS#"      to="-q64 $f90flags -O3 -qsuffix=cpp=F90" />
61          <sub from="#MYFLAGS#"      to="-qlanglvl=extended -qfree=f90 -q64 -qrealsize=8
    -O4 -qarch=pwr5 -qtune=pwr5 -qessl -qsuffix=f=f90" />
62          <sub from="#CC#"            to="$cc" />
63          <sub from="#CFLAGS#"        to="-q64 $cflags" />
64          <sub from="#CXX#"           to="$cxx" />
65          <sub from="#CXXFLAGS#"      to="-q64 $cxxflags" />
66          <sub from="#MPI_F90#"       to="$mpi_f90" />
67          <sub from="#MPI_F77#"       to="$mpi_f77" />
68          <sub from="#MPI_CC#"        to="$mpi_cc" />
69          <sub from="#MPI_CXX#"       to="$mpi_cxx" />
70          <sub from="#LD#"            to="$mpi_f90" />
71          <sub from="#LDFLAGS#"       to="-q64 $ldflags -O3" />
72          <sub from="#MPI_DIR#"       to="$mpi_dir" />
73          <sub from="#MPI_LIB#"       to="$mpi_lib" />
74          <sub from="#MPI_INC#"       to="$mpi_inc" />
75          <sub from="#MPI_BIN#"       to="$mpi_bin" />
76          <sub from="#BLAS_DIR#"      to="" />
77          <sub from="#BLAS_LIB#"      to="" />
78          <sub from="#LAPACK_DIR#"    to="" />
79          <sub from="#LAPACK_LIB#"    to="" />
80          <sub from="#FFTW_DIR#"      to="" />
81          <sub from="#FFTW_LIB#"      to="" />
82          <sub from="#FFTW_INC#"      to="" />
83          <sub from="#MODULE_CMD#"     to="$module_cmd" />
84          <sub from="#MODULE_FILES#"   to="" />
85      </substitute>
86
87
88      <!-- issue build command -->
89      <command>make -f Makefile</command>
90  </compile>
91
92  </compilation>
```

The compile.xml file carries out two tasks: substitution of platform-related variables, such as compiler and compile flags; and the command to compile the code. For HELIUM three files requires substitution: the Helium source code file, which requires the `$nblocks` and `$xlast` variables to be set appropriately; the makefile which needs the executable name (which is benchmark dependant) and the makefile.var file, which is where the compiler and flags are set. Most of the platform specific tools are set in the platform.xml file and the variable used appropriately (see line 29 for example, where the C++ compiler is set). However, particular variables can be overridden, such as the compile flags. If the platform.xml specifies, say, `-fast`, but this does not work on a particular code, this can be overridden here.

Substitution is done by specifying a keyword in the file (e.g. `#LD#`) which JuBE searches for and replaces with the variable specified in the XML file. A variable not specified in the XML file is not substituted. A variable specified in the XML file, but is not in the input file, is ignored.

Finally, the command tag is used to issue the command to build the application. This can point to a makefile, configure script, or bash script, for example.

7.3.3 *prepare.xml*

This file carries out any steps required before the application is executed.

```
93  <preparation>
94
95  <!-- ************************************************************************ -->
96
97  <prepare cname="standard">
98
99      <mkdir directory="graph" />
100     <mkdir directory="ground" />
101     <mkdir directory="state" />
102     <mkdir directory="data" />
103     <command>(cd $rundir;chmod u+x ./src/make_file.sh;./src/make_file.sh)</command>
104
105 </prepare>
106
107 <!-- ************************************************************************ -->
108
109 </preparation>
```

For HELIUM this requires four directories are created as defined above, whilst the script creates the necessary input/output files.

7.3.4 *execute.xml*

This file contains steps to execute the application. Like the compile.xml file this comprises of substitutions to the platform's skeleton job submission script and the submit command. Multilpe platforms can be contained in the same file.

```
110 <execution>
111 <!-- ************************************************************************ -->
112 <execute cname="IBM-SP5-HPCx">
113     <input files="../../platform/IBM-SP5-HPCx/ibm_llsubmit.job.in" />
114
115     <substitute infile="ibm_llsubmit.job.in" outfile="ibm_llsubmit.job">
116         <sub from="#OUTDIR#"            to="$outdir" />
117         <sub from="#STDOUTLOGFILE#"     to="$stdoutlogfile" />
118         <sub from="#STDERRLOGFILE#"     to="$stderrlogfile" />
119         <sub from="#BENCHNAME#"         to="$benchname $subid" />
120         <sub from="#NODEUSAGE#"         to="not_shared" />
121         <sub from="#TOTALTASKS#"        to="$tasks" />
122         <sub from="#TIME_LIMIT#"        to="02:00:00" />
123         <sub from="#NODES#"             to="$nodes" />
124         <sub from="#TASKSPERNODE#"      to="$taskspernode" />
125         <sub from="#NOTIFICATION#"      to="never" />
126         <sub from="#THREADSPERTASK#"    to="$threadspertask" />
127         <sub from="#STACK#"             to="400mb" />
128         <sub from="#EXECUTABLE#"        to="$executable" />
129         <sub from="#ENV#"               to="$env" />
130         <sub from="#PREPROCESS#"        to="" />
131         <sub from="#POSTPROCESS#"       to="" />
132         <sub from="#STARTER#"           to="poe" />
133         <sub from="#ARGS_STARTER#"      to="" />
134         <sub from="#MEASUREMENT#"       to="time /usr/local/bin/hpmcount" />
135         <sub from="#ARGS_EXECUTABLE#"    to="" />
136     </substitute>
137
138     <environment>
139         <env var="MP_LABELIO" value="yes" />
140         <env var="MP_INFOLEVEL" value="2" />
141         <env var="MP_SHARED_MEMORY" value="yes" />
142         <env var="MP_TASK_AFFINITY" value="MCM" />
143         <env var="MP_EAGER_LIMIT"  value="65536" />
144         <env var="MEMORY_AFFINITY"  value="MCM" />
145         <env var="OMP_NUM_THREADS"  value="$threadspertask" />
146         <env var="TRACE_TEXTONLY"  value="1" />
```

```
147     </environment>
148
149     <command>llsubmit ibm_llsubmit.job</command>
150 </execute>
151
152 </execution>
```

Line 113 points to the skeleton job submission script for the platform in question. Variables in that file are substituted as per the compile.xml file. The `environment` tag allows environment variables to be set and are placed in the job submission script. Finally, the command tag contains the command to submit the job to the batch system.

7.3.5 *verify.xml*

Checks that the run has completed successfully.

```
153 <verification>
154
155 <!-- ************************************************************************ -->
156 <!-- predefined vars:
157      $subdir      -> execution dir of benchmark run
158      $stdoutfile  -> $stdout file of benchmark run
159      $stderrfile  -> $stderr file of benchmark run
160      $...         -> params from benchmark specification in toplevel dir
161 -->
162
163 <verify cname="HELIUM">
164   <command>run/check_results_helium.pl $subdir/verify.xml $stdoutfile $stderrfile
    $subdir $totaltasks</command>
165 </verify>
166
167 <!-- ************************************************************************ -->
168
169 </verification>
```

The verification for HELIUM requires that the `Total Population` variable is equal to 1. This is done in a Perl script kept in the `run` directory. The script also checks the correct number of cores were used.

```
170 #!/usr/bin/perl -w
171
172 use strict;
173 use Carp;
174
175 my $patint="([\\+\\-\\d]+)";    # Pattern for Integer number
176 my $patfp ="([\\+\\-\\d.Ee]+)"; # Pattern for Floating Point number
177 my $patwrd="([\^\\s]+)";        # Pattern for Work (all noblank characters)
178 my $patnint="[\\+\\-\\d]+";     # Pattern for Integer number, no ()
179 my $patnfp ="[\\+\\-\\d.Ee]+";  # Pattern for Floating Point number, no ()
180 my $patnwrd="[\^\\s]+";         # Pattern for Work (all noblank characters), no ()
181 my $patbl ="\\s+";              # Pattern for blank space (variable length)
182
183 if(scalar(@ARGV) != 5) {
184     printf(STDERR "incorrect number of parameters(%d) of $0 (5 required)\n",
185         scalar @ARGV);
186     exit(-1);
187 }
188
189 my $xmloutfile = $ARGV[0];
190 my $stdoutfile = $ARGV[1];
191 my $stderrfile = $ARGV[2];
192 my $subdir     = $ARGV[3];
193 my $totcores   = $ARGV[4];
194 my $vcheck=0;
195 my $vcomment="not tested";
196 my $vval1=0;
197 my $vval2=0;
198 my $vval3=0;
199 my $vvalref1=0;
200 my $vvalref2=0;
201 my $vvalref3=0;
202 my $limit=1.e-15;
203 my $outptfile="$subdir/hstat.prace";
```

```
print "$outptfile";

if(-f $outptfile) {
    open(OUT,"$outptfile") || die "$outptfile not found!";
    $vcheck = 1;
    $vcomment = "Result verified";
    while($vcheck) {
       my $outline = <OUT>;
       if (m/Number of PEs/) {
       my @cores = split(/=/,$outline);
       if ($cores[1] != $totcores) {
           $vcheck=0;
           $vcomment="Verification failed: Number of cores was not as expected";
           $vval1 = $outline;
           $vvalref1 = $totcores;
           last;
           }
    }
    close(OUT);
} else {
    $vcheck=0; $vcomment="no output file found";
}

$outputfile="$subdir/$stdoutfile";
print "$outptfile";
if(-f $outptfile) {
    open(OUT,"$outptfile") || die "$outptfile not found!";
    $vcheck = 1;
    $vcomment = "Result verified";
    while($vcheck) {
       my $outline = <OUT>;
       if (m/Total Population/) {
       my @answer = split(/=/,$outline);
       if ($answer[1] < 0.999999 || $answer[1] > 1.00001) {
           $vcheck=0;
           $vcomment="Verification failed: Incorrect answer";
           $vval2 = $outline;
           $vvalref2 = "1";
           last;
           }
    }
    close(OUT);
} else {
    $vcheck=0; $vcomment="no output file found";
}

open(XMLOUT,"> $xmloutfile") || die "cannot open file $xmloutfile";
print XMLOUT "<verify>\n";
print XMLOUT " <parm name=\"vcheck\" value=\"$vcheck\" type=\"bool\" unit=\"\" />\n";
print XMLOUT " <parm name=\"vcomment\" value=\"$vcomment\" type=\"string\"
   unit=\"\"/>\n";
print XMLOUT " <parm name=\"vval1\" value=\"$vval1\" type=\"float\" unit=\"\"/>\n";
print XMLOUT " <parm name=\"vvalref1\" value=\"$vvalref1\" type=\"float\"
   unit=\"\"/>\n";
print XMLOUT " <parm name=\"vval2\" value=\"$vval2\" type=\"float\" unit=\"\"/>\n";
print XMLOUT " <parm name=\"vvalref2\" value=\"$vvalref2\" type=\"float\"
   unit=\"\"/>\n";
print XMLOUT "</verify>\n";
print XMLOUT "\n";
close(XMLOUT);


exit(0);
```

7.3.6 *analyse.xml*

This file contains instructions on how to scan the output to extract the maingful data. Each platform has a separate entry in this file.

```
<analyzer>

<!-- ************************************************************************ -->
<!-- Input is stdout and stderr of benchmark run -->
<!-- Standard result parameter:
     - walltime -->
```

```
<!-- ************************************************************************ -->

<analyse cname="IBM-SP5-HPCx">
    <includepattern file="./analyse-pattern-helium.xml" />
    <includepattern file="../../skel/hpm3patterns.xml" />
</analyse>

<!-- ************************************************************************ -->


</analyzer>
```

The actual work is done using the `analyse-pattern-app.xml` file.

```
<patterns>

<!-- ************************************************************************ -->
<!-- *   application specific patterns for analyse of HELIUM results       * -->
<!-- ************************************************************************ -->

<parm name="walltime" unit="s" mode="line,last" type="float">
WallClock Time.*=  $patfp
</parm>
<!-- ************************************************************************ -->

</patterns>
```

The above contains a search for the wallclock time using Perl regular expressions.

7.4 CrayPat Case study: High-Performance Linpack Benchmark (HPL)

To showcase some Craypat features the High-Performance Linpack Benchmark (HPL) was run on a Cray XT5. The order of the coefficient matrix was 20000, the partitioning blocking factor was 80 and the number of process rows and columns were 4 and 8. The tools are very versatile and only a few reports and results can be shown here. In all examples the program was instrumented to produce tracing experiments, sampling was not used.

The first example shows load balance across processes by function group. The group USER show how much time (absolute time and percentage value from the TOTAL time) the numerical calculations took (89.2%) and the group MPI show same information for the MPI communication (10.8%). The rows that ends up with character string "pe.number" will show three processes having the maximum, median, and minimum times for each task.

Time %	Time	Calls	Group
100.0%	29.569722	3803836	Total
89.2%	26.365399	2	USER
3.0%	28.014343	1	pe.1
2.8%	26.134757	1	pe.16
2.7%	25.571915	1	pe.18
10.8%	3.204323	3803834	MPI
5.4%	1.582981	649	MPI_Recv
0.2%	1.950585	637	pe.18
0.2%	1.747239	641	pe.15
0.1%	0.935157	654	pe.1
3.5%	1.030649	6387	MPI_Send
0.1%	1.146458	6321	pe.7
0.1%	1.032805	6310	pe.26
0.1%	0.954339	6502	pe.25

Table 12: MPI-profile.

The second example shows a profile of the time spent in the USER group, comprising user defined functions.

Time %	Time	Imb. Time	Imb. Time %	Calls	Group Function PE='HIDE'
100.0%	53.691687	--	--	104950799	Total
100.0%	53.691683	--	--	104950797	USER
40.1%	21.556808	0.631945	2.9%	2684	HPL_dgemm
23.7%	12.725938	0.469476	3.7%	25318835	HPL_setran
10.7%	5.751859	0.234090	4.0%	2	HPL_pdmatgen
8.1%	4.359664	0.303725	6.7%	25001250	HPL_rand
5.3%	2.868864	1.915342	41.3%	14	HPL_broadcast
2.1%	1.113295	0.078952	6.8%	1752595	HPL_bcast_1ring
1.7%	0.928316	0.105795	10.6%	920	HPL_spreadT
1.3%	0.702743	0.058028	7.9%	246	HPL_rollT
1.2%	0.639501	0.024283	3.8%	25403473	HPL_lmul
1.0%	0.534679	0.020632	3.8%	25361158	HPL_ladd

Table 13: Function profile.

There are four functions that are called over 25 million times in the initialization stage; even a small error in the estimation of the measurement overhead will skew the results. By excluding these functions we get the following more accurate profile.

Time %	Time	Imb. Time	Imb. Time %	Calls	Group Function PE='HIDE'
100.0%	29.225464	--	--	3667922	Total
100.0%	29.225458	--	--	3667920	USER
73.8%	21.556087	0.630827	2.9%	2684	HPL_dgemm
8.1%	2.355882	0.079908	3.4%	2	HPL_pdmatgen
3.7%	1.075673	0.115775	10.0%	1653514	HPL_bcast_1ring
3.2%	0.939993	0.076457	7.8%	920	HPL_spreadT
2.5%	0.744454	0.061003	7.8%	246	HPL_rollT
1.8%	0.516974	0.315525	39.1%	14	HPL_broadcast
1.4%	0.410194	0.087556	18.2%	1	HPL_pdgesv0
1.4%	0.405300	0.015176	3.7%	2684	HPL_dtrsm

Function-profile where HPL_setran, HPL_rand, HPL_lmul and HPL_ladd have been excluded

Table 14: Function profile, less complex.

The next example shows statistics for sent messages. One can stufy the statistics for the whole program, but also to look at statistics for a certain process.

Totals for program	
Sent Msg Total Bytes	568112259
Sent Msg Count	6387
MsgSz <16B Bytes	128
16B<= MsgSz <256B Bytes	129
256B<= MsgSz <4KB Bytes	4638316
4KB<= MsgSz <64KB Bytes	9352709
64KB<= MsgSz <1MB Bytes	231788537
1MB<= MsgSz <16MB Bytes	322332441

Table 15: Program totals.

Sent Msg Total Bytes	580424824
Sent Msg Count	6538
MsgSz <16B Bytes	32
256B<= MsgSz <4KB Bytes	6784640
4KB<= MsgSz <64KB Bytes	9840008
64KB<= MsgSz <1MB Bytes	237865496
1MB<= MsgSz <16MB Bytes	325934648

Table 16: All sent message statistics from process number 9.

Sent Msg Total Bytes	457665112
Sent Msg Count	3283
256B<= MsgSz <4KB Bytes	3371520
4KB<= MsgSz <64KB Bytes	4359688
64KB<= MsgSz <1MB Bytes	123999256
1MB<= MsgSz <16MB Bytes	325934648

Table 17: Sent message statistics from process 9 to 2.

The third example shows hardware performance counter (HWPC) statistics. Here the default experiment is shown. It includes an overview of the most important counters. The counters are:

- PAPI_FP_OPS (Floating point operations)
- PAPI_L1_DCA (L1 data cache accesses)
- PAPI_L1_DCM (L1 data cache misses)
- PAPI_TLB_DM (Data translation lookaside buffer misses)

The example shows statistics gathered for the whole program, but one can also extract such data on a function or block level.

```
PAPI_L1_DCM                37.450M/sec    1135703222 misses
PAPI_TLB_DM                 0.375M/sec      11378421 misses
PAPI_L1_DCA              2035.276M/sec   61721014271 refs
PAPI_FP_OPS              5626.109M/sec  170615261610 ops
  User time (approx)        30.326 secs   69748937500 cycles
  Average Time per Call                      0.076976 sec/call
  Overhead / Time                               0.0%
  Cycles                    30.326 secs   69748937500 cycles
  User time (approx)        30.326 secs   69748937500 cycles
  Utilization rate                             97.5%
  HW FP Ops / Cycles                           2.45 ops/cycle
  HW FP Ops / User time   5626.109M/sec  170615261610 ops 61.2%peak(DP)
  HW FP Ops / WCT         5486.302M/sec
  Computation intensity                        2.76 ops/ref
  MFLOPS (aggregate)     180035.48M/sec
  LD & ST per TLB miss                      5424.39 refs/miss
  LD & ST per D1 miss                         54.35 refs/miss
  D1 cache hit ratio                          98.2%
  % TLB misses / cycle                         0.0%
```

Table 18: Performance results with the default HWPC experiment.

The fourth example shows some screenshots from the graphical Apprentice2 tool. Only summarized data was collected when running HPL, thus traffic-data and other analysis modes that show the behaviour of the program on a time-line are not available.

The pie chart in figure 14 on the left shows functions in the xhpl program, sorted by the number of times the functions were invoked. The pie chart on the right the data is sorted by the amount of time spent performing the functions. The icons in the toolbar show the analysis modes that are available for this data.

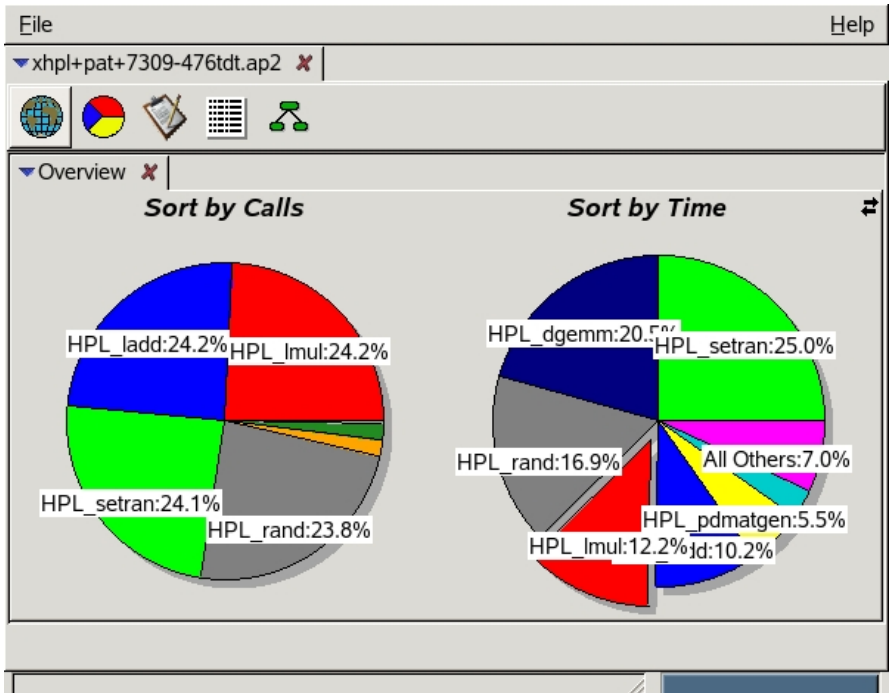

Figure 14: Pie chart example.

Figure 15: Load balance example of HLP_dgemm.

Figure 15 shows load balance of the HPL_dgemm (matrix multiply) routine. On the left hand side one can see the process numbers.

7.5 IBM HPCT Assessment

7.5.1 *Hardware Performance Monitor (HPM)*

The default output of HPM looks like:

```
hpmcount v3.2.2 (IHPCT v2.2.0) summary

########  Resource Usage Statistics  ########

Total amount of time in user mode            : 3.333841 seconds
Total amount of time in system mode          : 0.729397 seconds
Maximum resident set size                    : n/a
```

```
Average shared memory use in text segment    : n/a
Average unshared memory use in data segment  : n/a
Number of page faults without I/O activity   : 83016
Number of page faults with I/O activity      : 0
Number of times process was swapped out      : 0
Number of times file system performed INPUT  : n/a
Number of times file system performed OUTPUT : n/a
Number of IPC messages sent                  : n/a
Number of IPC messages received              : n/a
Number of signals delivered                  : n/a
Number of voluntary context switches         : n/a
Number of involuntary context switches       : n/a

#######  End of Resource Statistics  ########

Execution time (wall clock time)     : 6.53337597846985 seconds

PM_FPU_1FLOP (FPU executed one flop instruction )               :        42233432
PM_FPU_FMA (FPU executed multiply-add instruction)              :     11040495281
PM_FPU_FSQRT_FDIV (FPU executed FSQRT or FDIV instruction)      :           79074
PM_FPU_FLOP (FPU executed 1FLOP, FMA, FSQRT or FDIV instruction) :     11082807787
PM_RUN_INST_CMPL (Run instructions completed)                   :     10414883074
PM_RUN_CYC (Run cycles)                                         :      6273057293

Utilization rate                               :          20.411 %
Instructions per run cycle                     :           1.660
Total floating point operations                :       22123.303 M
Flop rate (flops / WCT)                        :        3386.198 Mflop/s
Flops / user time                              :       16589.681 Mflop/s
Algebraic floating point operations            :       22123.224 M
Algebraic flop rate (flops / WCT)              :        3386.186 Mflop/s
Algebraic flops / user time                    :       16589.621 Mflop/s
FMA percentage                                 :          99.809 %
% of peak performance                          :          88.168 %
```

7.5.2 *MPI Profiler*

The instrumented application creates MPI profiles files like:

```
-----------------------------------------------------------------
MPI Routine                  #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                    32            0.0          0.000
MPI_Comm_rank                    32            0.0          0.000
MPI_Send                       1119         2269.4          0.006
MPI_Recv                        376         4007.9          0.936
MPI_Irecv                       739         1383.3          0.002
MPI_Iprobe                 20906040            0.0         21.923
MPI_Wait                       739            0.0         18.627
-----------------------------------------------------------------
total communication time = 41.494 seconds.
total elapsed time       = 50.675 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Send                 #calls    avg. bytes      time(sec)
                              4           4.0          0.000
                             16           8.0          0.000
                              2          12.0          0.000
                            126          32.0          0.000
                              4          52.0          0.000
                            503          96.0          0.002
                              2         164.0          0.000
                             12         366.7          0.000
                             25         738.2          0.000
                             63        1542.1          0.000
                            119        3100.7          0.001
                            146        6057.4          0.001
                             97       11475.1          0.002
```

```
MPI_Recv                  #calls    avg. bytes      time(sec)
                               2           4.0          0.354
                              10           8.0          0.070
                               2          12.0          0.000
                             126          32.0          0.002
                               1         168.0          0.000
                               2         360.0          0.000
                              11         735.3          0.000
                              24        1518.3          0.000
                              57        3185.8          0.507
                              70        6092.2          0.000
                              71       11962.9          0.003

MPI_Irecv                 #calls    avg. bytes      time(sec)
                             501          64.0          0.001
                               1          96.0          0.000
                               5         198.4          0.000
                               5         380.8          0.000
                              14         750.3          0.000
                              46        1570.4          0.000
                              73        3039.7          0.000
                              69        6209.4          0.000
                              25       10164.8          0.000

-----------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 41.494 sec for task 0
  median  communication time = 42.109 sec for task 1
  maximum communication time = 42.390 sec for task 2
```

The instrumented application also generates trace files that can be visualized using PeekPerf in figure 16:

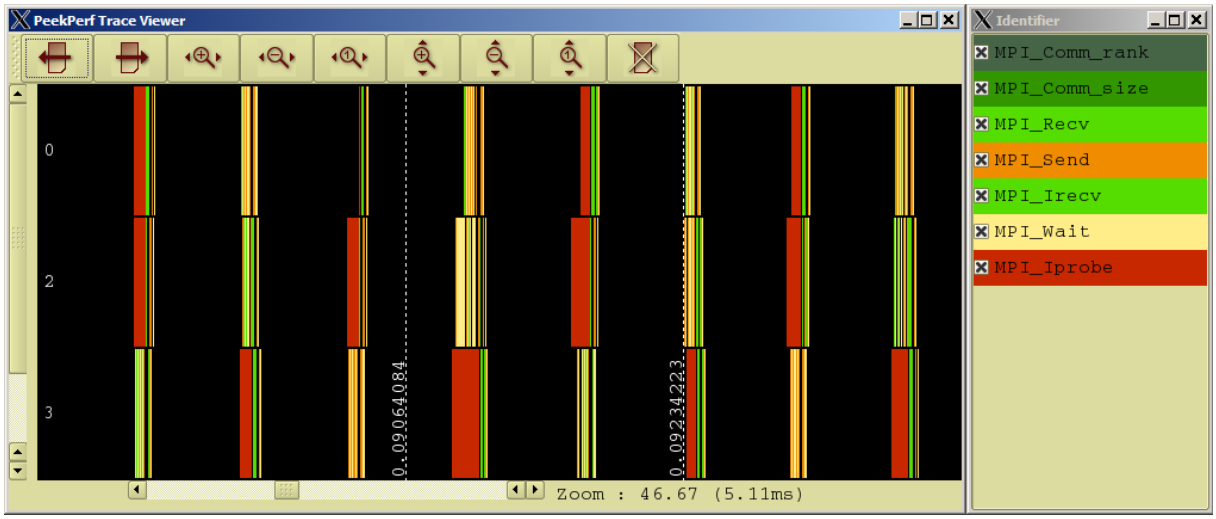

Figure 16: Peekperf visualisation of trace files.

By inspecting the communication patterns, these visualizations allow for detection of communication bottlenecks.

7.5.3 *Xprofiler*

Xprofiler is a visualization tool for `gmon.out` profiling data created by applications compiled with the `-pg` flag. Xprofiler is a nice GUI that gives the information that is also provided by the well-known `gprof` command line tool for displaying call graph profile data. Xprofiler does not yet work for 64-bit executables, which is a pity on large memory nodes

(the machine we used for the assessment has both 128 and 256 GB nodes). An example is shown in figure 17.

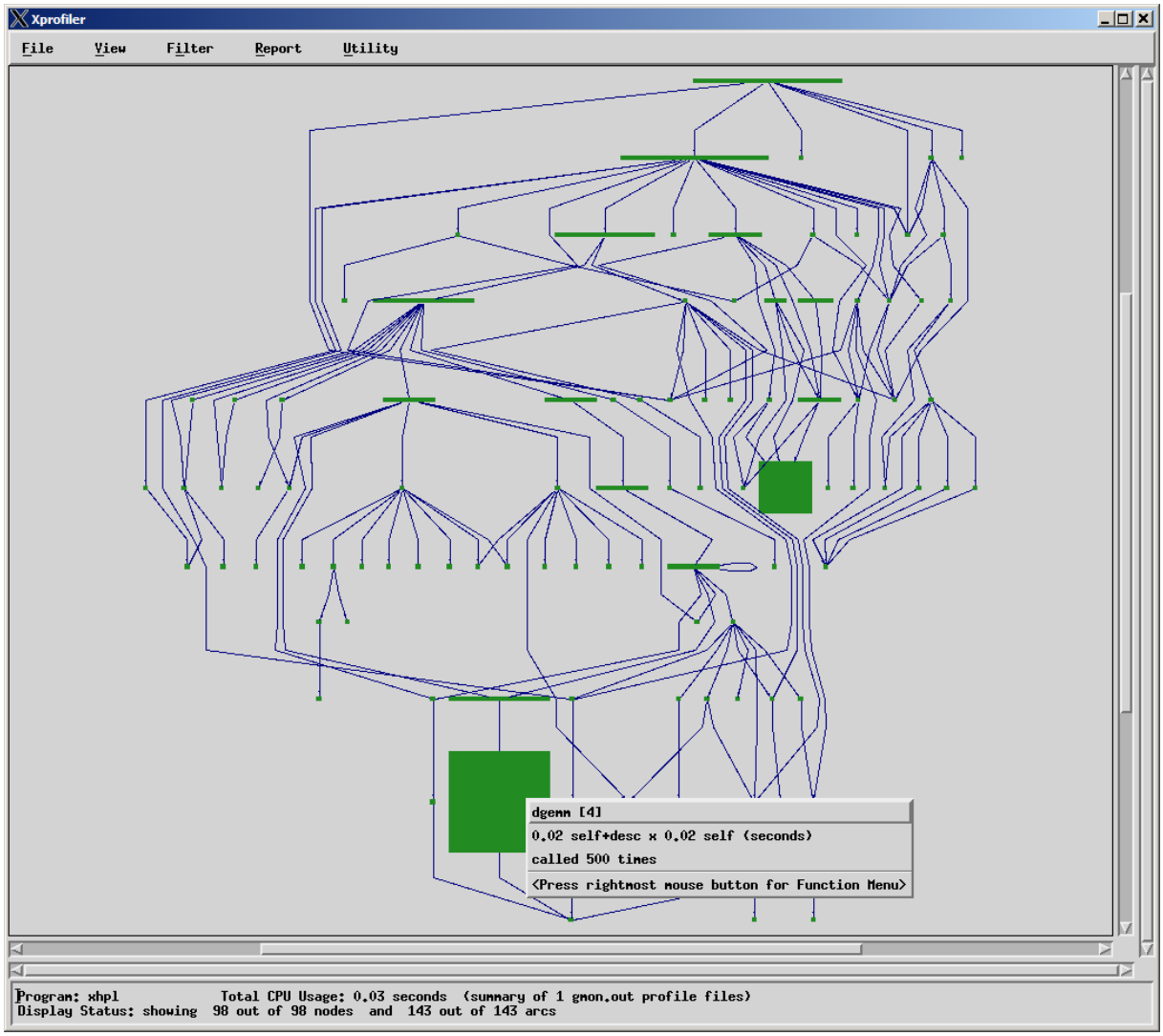

Figure 17: Xprofiler example.

7.6 IPM Assessment

We used the implementation of the High-Performance Linpack benchmark for IBM QS22 systems with two PowerXCell 8i processors available in [34].

7.6.1 *MPI subs*

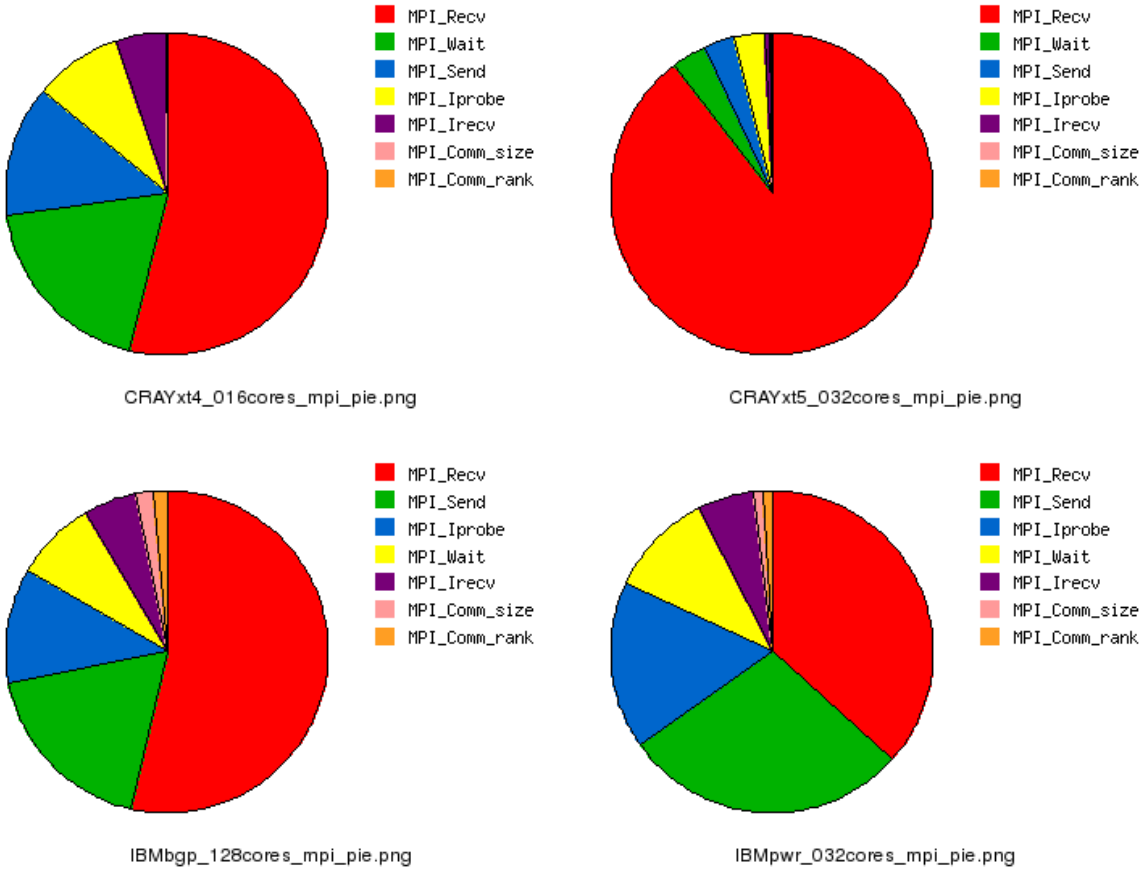

Figure 18: IPM pie charts.

7.6.2 *MPI topology*

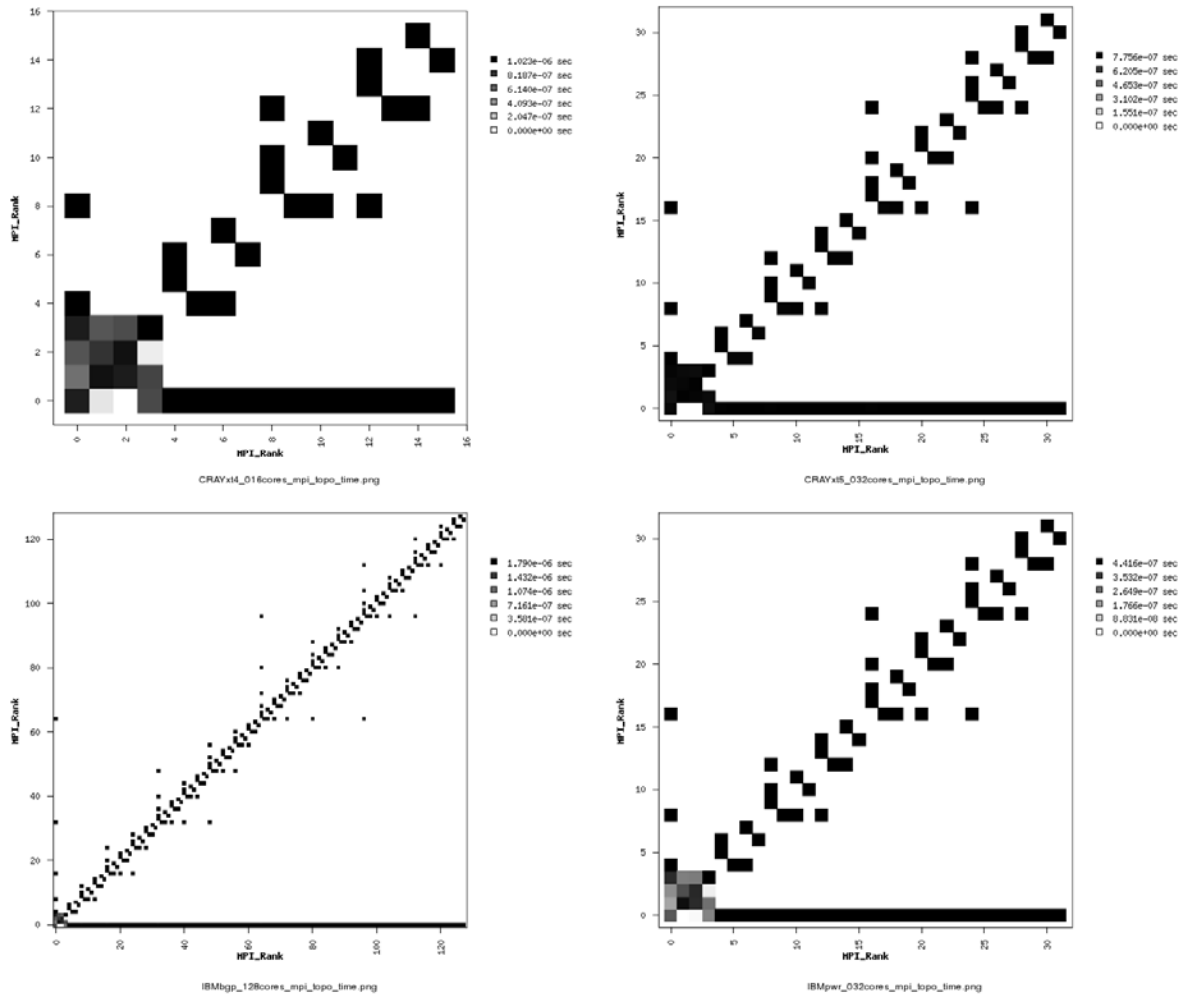

Figure 19: IPM MPI topology overview.

7.6.3 *MPI message sizes*

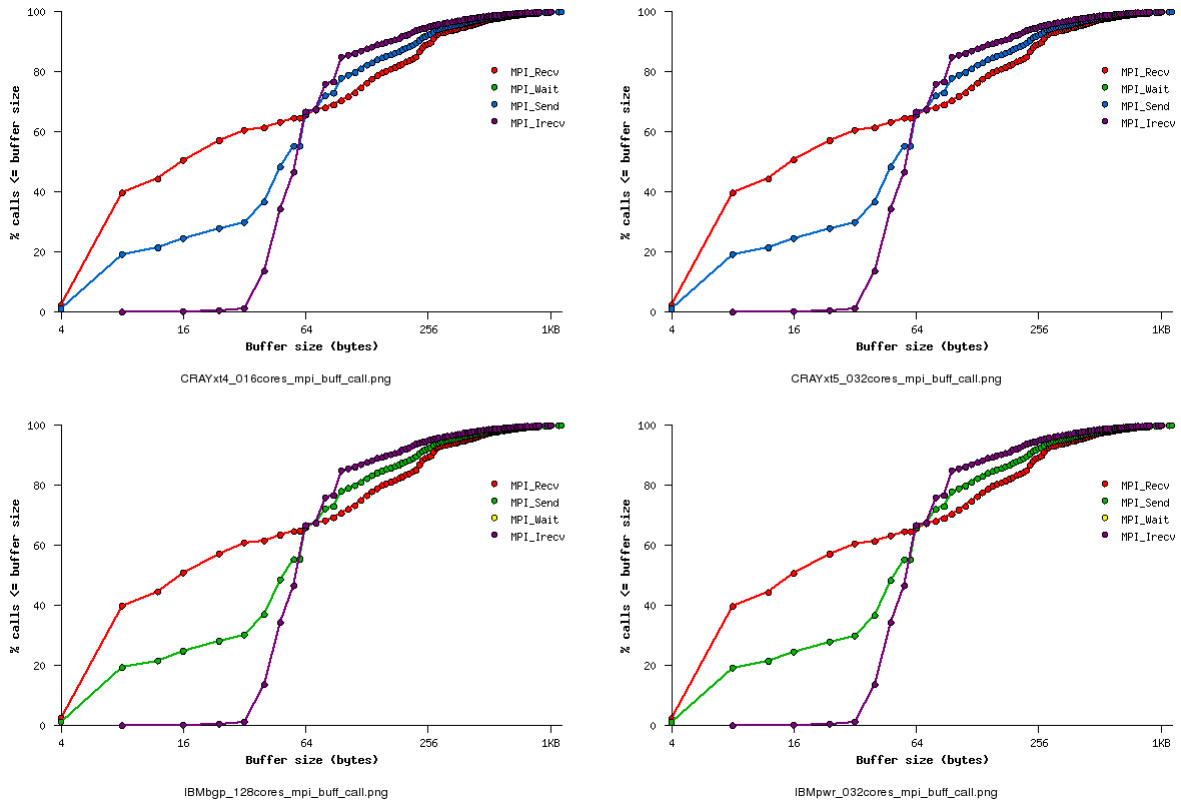

Figure 20: IPM message sizes graphs.

7.6.4 *MPI load balance*

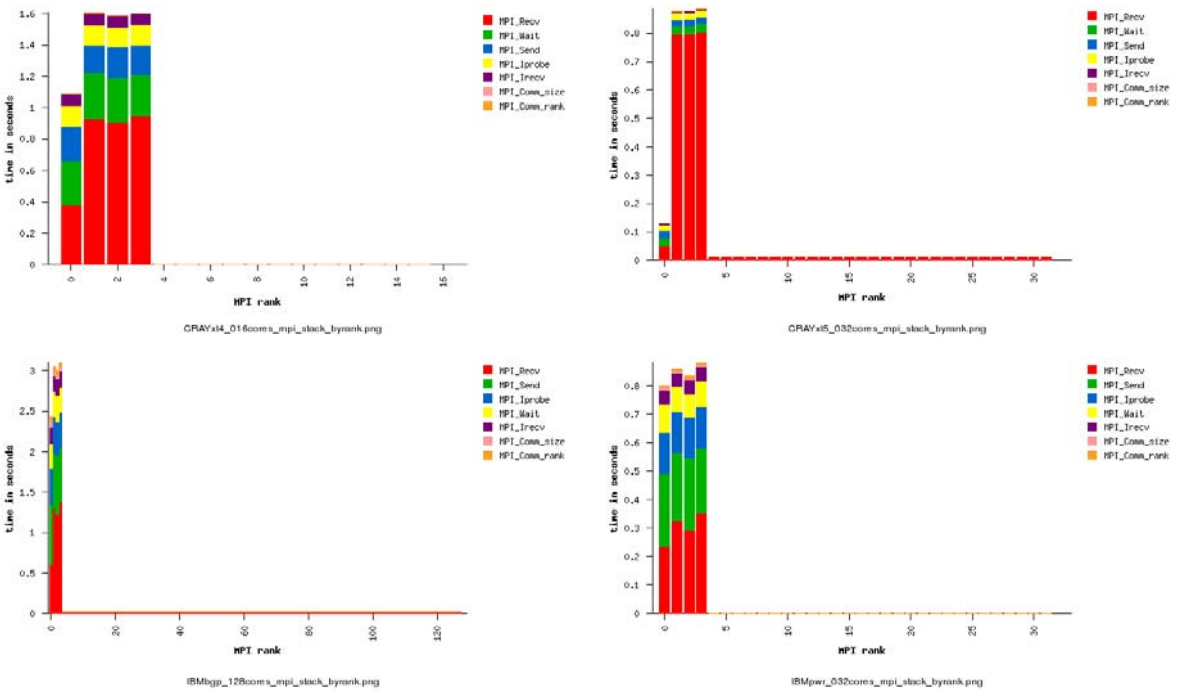

Figure 21: IPM MPI load balance information.