



## **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**

**D6.3.2**  
**Final Benchmark Suite**

**Final**

Version: 1.1  
Author(s): Peter Michielse (NCF), Lukas Arnold (FZJ), Olli-Pekka Lehto (CSC), Walter Lioen (SARA)  
Date: 18/06/2010

## Project and Deliverable Information Sheet

<b>PRACE Project</b>	<b>Project Ref. №: RI-211528</b>	
	<b>Project Title: Partnership for Advanced Computing in Europe</b>	
	<b>Project Web Site:</b> <a href="http://www.prace-project.eu">http://www.prace-project.eu</a>	
	<b>Deliverable ID:</b> D6.3.2	
	<b>Deliverable Nature:</b> Report, Final Benchmark Suite	
	<b>Deliverable Level:</b> PU	<b>Contractual Date of Delivery:</b> 30 / June / 2010
		<b>Actual Date of Delivery:</b> 30 / June / 2010
<b>EC Project Officer: Bernhard Fabianek</b>		

\* - 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

<b>Document</b>	<b>Title:</b> Final Benchmark Suite	
	<b>ID:</b> D6.3.2	
	<b>Version:</b> 1.1	<b>Status:</b> Final
	<b>Available at:</b> <a href="http://www.prace-project.eu">http://www.prace-project.eu</a>	
	<b>Software Tool:</b> Microsoft Word 2003	
	<b>File(s):</b> D6.3.2 Final Benchmark Suite.doc	
<b>Authorship</b>	<b>Written by:</b>	Peter Michielse (NCF), Lukas Arnold (FZJ), Olli-Pekka Lehto (CSC), Walter Lioen (SARA)
	<b>Contributors:</b>	Sebastian von Alfthan, CSC, Finland Mauricio Araya, BSC, Spain Eric Boyer, CINES, France Carlo Cavazzoni, CINECA, Italy Bertrand Cirou, CINES, France Raúl de la Cruz, BSC, Spain Jacques David, CEA, France John Donners, SARA, The Netherlands Jussi Enkovaara, CSC, Finland Albert Farres, BSC, Spain Maciej Filocha, PSNC, Poland Xu Guo, EPCC, United Kingdom Joachim Hein, EPCC, United Kingdom Guillaume Houzeaux, BSC, Spain Harald Klimach, HLRS, Germany Mikko Lyly, CSC, Finland Pekka Manninen, CSC, Finland Charles Moulinec, STFC, United Kingdom Fernando Nogueira, UC-LCA, Portugal Jean-Guillaume Piccinali, CSCS, Switzerland Martin Polak, GUP, Austria Andrew Porter, STFC, United Kingdom Orlando Rivera, LRZ, Germany Xavier Saez, BSC, Spain Andrew Sunderland, STFC Daresbury Laboratory, United Kingdom Maciej Szpindler, PSNC, Poland
	<b>Reviewed by:</b>	Tim Stitt (CSCS), Thomas Eickermann (FZJ)
	<b>Approved by:</b>	Technical Board

## Document Status Sheet

Version	Date	Status	Comments
0.1	04/August/2009	Draft	Outline version
0.2	14/September/2009	Draft	Draft for: chapters 1, 2, 3 chapters 4 and 5 Appendix (not complete yet) Distribution to task leaders WP6
0.3	15/October/2009	Draft	Appendix completed
0.4	30/October/2009	Draft	Chapter 3 completed
0.5	25/November/2009	Draft	Included subtask text for chapters 4, 5 and 6
0.9	30/November/2009	Final Draft WP6	Lay-out checked
0.95	04/December/2009	Final Draft PMO	Comments and last updates WP6 included
1.0	21/December/2009	Final version	
1.05	04/June/2010	Draft in Extension Phase (1H2010)	Relevant tables in chapters 3 and 4 updated based on work in extension phase (1H2010)
1.07	11/June/2010	Review in WP6	Minor modifications, check of results
1.09	16/June/2010	Review in PMO	Minor modifications
1.1	18/June/2010	Final version	

## Document Keywords and Abstract

<b>Keywords:</b>	PRACE, HPC, Research Infrastructure, Applications, Benchmark Suite, Synthetic Benchmarks, Prototypes, Performance Analysis Tools
<b>Abstract:</b>	<p>This deliverable D6.3.2 takes its predecessor D6.3.1 as the starting point for the definition of the final PRACE Application Benchmark Suite (PABS). This report describes the process from initial to final benchmark suite in detail. Scalability input has been received from task 5.4, which is responsible for the assessment of the PRACE prototypes, using the PRACE benchmark suite. Ease of use of the final benchmark suite in future situations is assured by the integration of the final benchmark suite into a framework, including testing procedures. The final PABS is part of D6.3.2, including its integration in the benchmark framework.</p> <p>Accompanying the process to the final benchmark suite, other software aspects are relevant as well. These include synthetic benchmarking and performance analysis tools, which are both important when the procurement of Tier-0 systems is prepared. D6.3.2, again with D6.3.1 as a starting point, describes the required contents of a synthetic benchmark, which is able to assess key hardware and system software aspects of Tier-0 systems; the actual assessment of the PRACE prototype systems is done in WP5. With respect to performance analysis tools, combinations of PRACE prototypes and performance analysis tools are tested and reported upon.</p> <p>The PABS has been prepared in such a way that it offers optimal flexibility. This includes the possibility to use a subset of the PABS for procurement of specific Tier-0 architectures, and also the possibility to adapt the contents of the subset towards the specific wishes of the installing Tier-0 Principal Partner. Industrial and pan-European coverage of the applications in the PABS is another aspect which has been included, especially when proceeding from the initial PABS to the final PABS. Yet another aspect is how to weight the various applications in the benchmark subset, together with the synthetic benchmark results and possibly less technical aspects. It should be clear that the PABS is instrumental in the selection process of Tier-0 systems, but has to be accompanied by other requirements.</p> <p>This document includes the results of efforts carried out during the Extension Phase in the first half of 2010.</p>

### Copyright notices

© 2010 PRACE Consortium Partners. All rights reserved. This document is a project document of the PRACE project. All contents are reserved by default and may not be disclosed to third parties without the written consent of the PRACE partners, except as mandated by the European Commission contract RI-211528 for reviewing and dissemination purposes.

All trademarks and other rights on third party products mentioned in this document are acknowledged as owned by the respective holders.

## Table of Contents

Project and Deliverable Information Sheet .....	i
Document Control Sheet.....	ii
Document Status Sheet .....	iii
Document Keywords and Abstract.....	iv
Table of Contents .....	v
List of Figures.....	vii
List of Tables.....	viii
References and Applicable Documents .....	viii
List of Acronyms and Abbreviations.....	ix
Executive Summary .....	1
<b>1 Introduction .....</b>	<b>2</b>
1.1 Structure of the Report .....	2
<b>2 Definitions, Objectives and Methodology.....</b>	<b>3</b>
2.1 Definitions .....	3
2.2 Objectives and Methodology .....	4
<b>3 Final Benchmark Suite .....</b>	<b>6</b>
3.1 Starting Point: Initial Benchmark Suite.....	6
3.2 Experiences with the Initial Benchmark Suite.....	8
3.3 Analysis and update of the Initial Benchmark Suite .....	12
3.4 Final Benchmark Suite.....	14
3.5 Conclusions and future work.....	15
<b>4 Integration in Benchmark Suite.....</b>	<b>17</b>
4.1 Integration Overview .....	18
<i>Benchmark Execution Standardization.....</i>	<i>19</i>
<i>Integration Status .....</i>	<i>20</i>
4.2 Extensions to standard Integration.....	21
4.3 Acceptance test .....	24
<i>Testing Procedure .....</i>	<i>24</i>
<i>Testing Status .....</i>	<i>24</i>
4.4 Conclusions .....	24
<b>5 Synthetic Benchmarks .....</b>	<b>26</b>
5.1 Introduction .....	26
5.2 Descriptions of Synthetic Benchmarks .....	26
5.3 Conclusions and Further Work.....	30
<b>6 Performance Analysis Tools .....</b>	<b>31</b>
6.1 Introduction .....	31
6.2 Allinea Optimization and Profiling Tool (OPT) .....	31
<i>Introduction .....</i>	<i>31</i>
<i>Assessment Environment .....</i>	<i>31</i>
<i>Conclusions .....</i>	<i>33</i>
<i>Suggestions for Future Improvements .....</i>	<i>34</i>
<i>Acknowledgements .....</i>	<i>34</i>
6.3 Cray Performance Analysis Framework.....	34

	<i>Features</i>	34
	<i>Feedback and experiences</i>	35
<b>6.4</b>	<b>Dewiz</b>	<b>35</b>
<b>6.5</b>	<b>IBM HPCT: High Performance Computing Toolkit</b>	<b>36</b>
	<i>Introduction</i>	36
	<i>Instrumenting the application</i>	37
	<i>Feedback to Vendor</i>	37
	<i>Conclusions</i>	37
<b>6.6</b>	<b>IPM: Integrated Performance Monitoring</b>	<b>38</b>
	<i>Introduction</i>	38
	<i>Deployment Status</i>	38
	<i>Considerations for Future Work</i>	38
<b>6.7</b>	<b>Scalasca</b>	<b>38</b>
	<i>Introduction</i>	38
	<i>Platforms and applications used for this investigation</i>	39
	<i>Scalasca Details</i>	39
	<i>Scalasca summary</i>	39
	<i>Acknowledgement</i>	40
<b>6.8</b>	<b>Vampir VNG</b>	<b>40</b>
<b>6.9</b>	<b>VPA: Visual Performance Analyzer</b>	<b>42</b>
	<i>Overview</i>	42
	<i>VPA Conclusion</i>	43
<b>6.10</b>	<b>General Conclusions</b>	<b>43</b>
<b>7</b>	<b>Annex</b>	<b>44</b>
<b>7.1</b>	<b>Scalability Experiments with Allinea OPT</b>	<b>44</b>
	<b>Linking Senga2 with OPT on the BlueGene/P</b>	<b>44</b>
	<b>Managing OPT profiling data for large-scale runs</b>	<b>45</b>
	<b>Timeline View</b>	<b>47</b>
	<b>MPI Summary View</b>	<b>49</b>
	<b>Histogram View</b>	<b>49</b>
	<b>Message Profile View</b>	<b>50</b>
<b>7.2</b>	<b>Scalability Experiments with CrayPat and Apprentice2</b>	<b>52</b>
<b>7.3</b>	<b>Scalability Experiments with IBM HPCT</b>	<b>59</b>
<b>7.4</b>	<b>Scalability Experiments with IPM</b>	<b>61</b>
<b>7.5</b>	<b>Scalability Experiments with Scalasca</b>	<b>63</b>
<b>7.6</b>	<b>Scalability Experiments with VPA</b>	<b>68</b>
<b>7.7</b>	<b>Descriptions of Application Benchmark Codes</b>	<b>70</b>
	<i>7.7.1 QCD</i>	70
	<i>7.7.2 Quantum_Espresso</i>	72
	<i>7.7.3 NAMD</i>	75
	<i>7.7.4 CPMD</i>	75
	<i>7.7.5 Code_Saturne</i>	76
	<i>7.7.6 GADGET</i>	77
	<i>7.7.7 EUTERPE (TORB)</i>	78
	<i>7.7.8 WRF</i>	78
	<i>7.7.9 NEMO</i>	79
	<i>7.7.10 CP2K</i>	80
	<i>7.7.11 GROMACS</i>	80
	<i>7.7.12 NS3D</i>	82
	<i>7.7.13 AVBP</i>	82
	<i>7.7.14 HELIUM</i>	83

7.7.15	<i>TRIPOLI-4</i> .....	83
7.7.16	<i>PEPC</i> .....	84
7.7.17	<i>GPAW</i> .....	84
7.7.18	<i>ALYA</i> .....	85
7.7.19	<i>OCTOPUS</i> .....	85
7.7.20	<i>BSIT</i> .....	86
7.7.21	<i>ELMER</i> .....	87
7.7.22	<i>SPECFEM3D</i> .....	88

## List of Figures

Figure 1: JuBE workflow .....	17
Figure 2: JuBE substitution strategy .....	18
Figure 3: Vampir Total time line view, traces on Juropa, visualization on HLRB2. ....	41
Figure 4: Session Manager of OPT summarizing a range of different jobs. ....	47
Figure 5: OPT Timeline view of a 64 process job with tracing restricted to 2000 functions. ....	48
Figure 6: OPT Timeline view of a 64 process job with tracing restricted to 3 time-steps. ....	48
Figure 7: Zoomed timeline view for halo exchange communication pattern on 64 processes. ....	48
Figure 8: MPI Summary View for 256 process run of Senga2. ....	49
Figure 9: Variance of time spent in MPI_ALLREDUCE for 256 process run of Senga2. ....	50
Figure 10: Stacked Histogram of Time Spent in all MPI Calls for 256 process run on Blue Gene/P... ..	50
Figure 11: Message Profile View for 64 process run of Senga2. ....	51
Figure 12: Message Profile View for 256 process run of Senga2. ....	51
Figure 13: Profile of execution as shown by Apprentice2. ....	55
Figure 14: Load balance view in Apprentice 2. ....	56
Figure 15: Callgraph view in Apprentice 2. ....	57
Figure 16: Timeline of Gromacs visualized using Apprentice 2. ....	58
Figure 17: Main window of peekperf. ....	60
Figure 18: Scalability details of IPM. ....	62
Figure 19: Viewing the HELIUM profile via Scalasca GUI on the Huygens system. ....	64
Figure 20: MPI profiling of NAMD on Juropa. ....	66
Figure 21: MPI profiling of NAMD on HECToR. ....	67
Figure 22: Screenshot: Profile Analyzer with sample-based profile. ....	68
Figure 23: Screenshot: Profile Analyzer showing profile-based annotated source code. ....	69



## List of Tables

Table 1: Overview of initial PABS. ....	7
Table 2: Porting status of initial PABS, as of November 2008. ....	7
Table 3: Porting status of initial PABS as of June 2009. ....	8
Table 4: Actual task 5.4 scalability results and licensing information on the initial PABS codes. ....	11
Table 5: Industrial usage of the initial PABS applications. ....	12
Table 6: Replacement codes for the final PABS. ....	13
Table 7: Extension codes for the final PABS. ....	13
Table 8: Overview of final PABS. ....	14
Table 9: Actual porting status of the final PABS per June 2010. ....	15
Table 10: Integration status as of June 2010. ....	20
Table 11: Status of acceptance tests as of June 2010. ....	25
Table 12: Runtime details – 64 process jobs. ....	46
Table 13: Runtime details – 256 process jobs. ....	46
Table 14: Runtime details – 1000 process jobs. ....	47
Table 15: Profile report for Gromacs. ....	53
Table 16: HW counter information for Gromacs. ....	53
Table 17: MPI information for Gromacs. ....	53
Table 18: Summary of profile information for Gromacs. ....	54
Table 19: Comparison of overhead introduced by CrayPat and IPM. ....	63

## References and Applicable Documents

- [1] PRACE: <http://www.prace-project.eu>
- [2] TRAC: <http://trac.edgewall.org/>
- [3] *The Scientific Case for a European Super Computing Industry*, HET (HPC in Europe Taskforce, 2006)
- [4] JuBE: <http://www.fz-juelich.de/jsc/jube/>
- [5] <http://sourceware.org/binutils/docs/gprof/index.html>
- [6] <http://ipm-hpc.sourceforge.net>
- [7] *Performance measurement and analysis of large-scale parallel applications on leadership computing systems* Brian J.N. Wylie, Markus Geimer, Felix Wolf, Scientific Programming **16** (2008) 167-181.
- [8] *Numerical integration of the time-dependent Schrödinger equation for laser-driven helium*, Edward S. Smyth, Jonathan S. Parker, K.T. Taylor, Computer Physics Communications **114** (1998) 1-14.
- [9] *Scalable Molecular Dynamics with NAMD*, J. Phillips et al., Journal of Computational Chemistry **26**, 1781 (2005).
- [10] *Charm++: Parallel Programming with Message-Driven Objects*, L. Kalé, S. Krishnan, in: *Parallel Programming using C++*, by G.V. Wilson and P. Lu. MIT Press, 175 (1996).
- [11] *Integrated Performance Views in Charm++: Projections Meets TAU*, Scott Biersdorff, Chee Wai Lee, Allen D. Malony, Laxmikant V. Kale, PPL Paper Number: 09-06, <http://charm.cs.uiuc.edu/papers/TauICPP09.shtml>.

## List of Acronyms and Abbreviations

BCO	Benchmark Code Owner.
BLAS	Basic Linear Algebra Subprograms (basic library).
BSCW	Basic Support for Cooperative Work, a collaborative workspace software package.
CAF	Co-Array Fortran
CPU	Central Processing Unit.
CSC	Center for Scientific Computing (Finland).
EC	European Commission.
FZJ	Forschungszentrum Juelich (Germany).
Gflop/s	$10^9$ floating-point operations per second (Gigaflop/s).
GUI	Graphical User Interface.
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym of Supercomputing.
HPL	High Performance Linpack benchmark.
IO	Input-Output
JuBE	Juelich Benchmarking Environment.
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.
NERSC	National Energy Research Scientific Computing Center.
OpenMP	Open Multi-Processing. An API for shared-memory parallel programming.
PABS	PRACE Application Benchmark Suite.
PAPI	Parallel Application Programming Interface.
PAT	Performance Analysis Tool.
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.
QCD	Quantum Chromo Dynamics.
SARA	SARA Computing and Networking Services Amsterdam (the Netherlands).
PRACE-SBM	PRACE Synthetic Benchmarks.
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.
UPC	Unified Parallel C
Wiki	Web page or collection of web pages for creating collaborative web sites.
WP	Work Package.

## Executive Summary

In order to be a lasting success, PRACE needs to understand the software requirements for future Petaflop/s systems. Apart from system software requirements, the performance (both actual and potential) of scientific applications on Petaflop/s architectures is of key importance, not only for procurement processes, but also to inform the PRACE peer review processes.

This deliverable D6.3.2 takes its predecessor D6.3.1 as the starting point for the definition of the final PRACE Application Benchmark Suite (PABS). Based on coverage of relevant application areas, scalability towards petascale (and beyond) architectures, licensing and industrial and global usage, D6.3.2 describes the process from initial to final benchmark suite in detail. With respect to scalability and performance, input has been received from task 5.4, which is responsible for the assessment of the PRACE prototypes, using the PRACE benchmark suite. Ease of use of the final benchmark suite in future situations is assured by the integration of the final benchmark suite into a flexible framework, including testing procedures.

Accompanying the process to the final benchmark suite, other software aspects are relevant as well. These include synthetic benchmarking and performance analysis tools, which are both important when the procurement of Tier-0 systems is planned. D6.3.2, again with D6.3.1 as the starting point, describes the required contents of a synthetic benchmark, which is able to assess key hardware and system software aspects of Tier-0 systems; the actual assessment of the PRACE prototype systems is done in WP5. With respect to performance analysis tools, combinations of PRACE prototypes and performance analysis tools are tested and reported upon.

The PABS has been prepared in such a way that it offers high flexibility. This includes the possibility to use a subset of the PABS for procurement of specific Tier-0 architectures, and also the possibility to adapt the contents of the subset towards the specific requirements of the installing Tier-0 Principal Partner. Industrial and pan-European coverage of the applications in the PABS is another aspect which has been included, especially when proceeding from the initial PABS to the final PABS. Yet another element is how to weight the various applications in the benchmark subset, together with the synthetic benchmark results and possibly other less technical aspects. It should be clear that the PABS is instrumental in the selection process of Tier-0 systems, but has to be accompanied by other requirements and conditions.

This document includes the results of efforts carried out during the Extension Phase in the first half of 2010. Tables 9, 10 and 11 actually reflect the work carried out in the Extension Phase.

# 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 consists of eight 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 Petaflop/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 PRACE prototype systems. The benchmark suite should represent application areas from the likely user base, but should also be such that the applications have enough scalability potential to run on Tier-0 systems, as aimed for by PRACE. Another aspect, which may become important when the actual benchmark suite will be used for procurement processes by the Principal Partners for petascale systems, is the possibility to use only a subset of the full benchmark suite. Integration of these and additional benchmark applications into an easy-to-use benchmark suite is also part of task 6.3.

Throughout the PRACE project, the applications in the benchmark suite have been used by WP5 task 5.4, in its testing and analysing of the prototype systems, and by tasks 6.4 and 6.5, which covered scalability to Petaflop/s systems and optimisation of applications.

These efforts cannot be undertaken without both suitable software tools and a thorough understanding of the underlying hardware. Thus, task 6.3 also covers performance analysis tools and synthetic benchmarking. 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 the results of task 6.3 is not only within PRACE, but hopefully also a wider HPC audience, as it offers characteristics and analysis for deployment of specific, heavily-used applications codes on future Petaflop/s systems.

Tables 9, 10 and 11 in sections 3 and 4 actually reflect the work carried out in the Extension Phase.

## 1.1 Structure of the Report

This document is structured as follows. In section 2, some important definitions will be given, and there will be a discussion on the refinement of objectives and the methodology to arrive at this. Early in the process in task 6.3, it was decided to subdivide the full task into a number of subtasks. Within this report, subdivision into subtasks is represented by the subsequent sections. Section 3 covers the actual transfer from the initial PRACE Application Benchmark Suite (PABS) into the final one. We report on the experiences with the applications in the initial PABS, on our analysis of licensing and scalability of the applications, and on our process to reach a final version of the PABS. Section 4 covers the actual integration of benchmark codes into the PABS, including the execution of acceptance tests to ensure the quality of the PABS with respect to both usage by others and long-term maintainability. Section 5 describes the actual contents of a synthetic benchmark suite within PRACE, which is clearly focusing on petascale and larger systems. Section 6 will take its initial survey of Performance Analysis Tools (PATs) and reports on several combinations of PAT and PRACE

prototype systems. Each of the sections 3 to 6 ends with conclusions and further remarks, as the topics under discussion are dynamic and will change over time.

In the Annex, we will give a brief description of each of the applications in the final PABS.

## 2 Definitions, Objectives and Methodology

### 2.1 Definitions

The procurement process of large HPC computer systems (either focussed on capacity or capability) is generally supported by the execution of a set of both synthetic benchmarks and 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 for real applications.

In D6.1 and D6.3.1, we have carefully laid out the definitions we have used with respect to benchmarking. For the readers' convenience, we repeat them here:

- 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 the application benchmark suite;
- A performance analysis tool is a tool 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 define 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;
- Optimisation is generally considered as the improvement of typically single-CPU (or single-core) performance (including IO aspects) of a 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.
- Petascaling is the process of scaling the performance of applications (including IO aspects) to petascale level, and is typically expressed in the number of cores which can still be efficiently used for the execution of the benchmark code. This is most likely dependent on actual input sets. In our view this includes node optimization and communication optimization.

## 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 (the previous deliverable D6.3.1);
- To eventually create a benchmark suite which becomes the PRACE benchmark suite for Tier-0 procurements (this deliverable D6.3.2 at the end of the PRACE project);
- To investigate important aspects which support the process of benchmarking and assessment of systems, including synthetic benchmarking, performance analysis tools and integration of the benchmark applications into an easy-to-use framework.

The initial PRACE Application Benchmark Suite (PABS) has been defined in the May-June 2008 timeframe. Since then, work has been done on the porting, optimisation and scalability of the benchmark codes. In May-June 2009, we have evaluated each code of the initial PABS on the following aspects:

- Actual scalability results and potential on Petascale systems and beyond;
- Actual licensing policy.

Based on this evaluation, and taking into account helpful comments from an EC review in March 2009, like extending the range of application areas and linkage to global scaling efforts and industrial usage, we have considered for each benchmark code whether to keep it in the PABS or to replace it by another code in the same application area. Such a replacement code must have demonstrated clear scalability and must have flexible licensing policies. Independent of that, we have studied possible extensions of the PABS into other application areas, under the condition of available knowledgeable human resources (Benchmark Code Owner, BCO) in the PRACE project. Chapter 3 describes this process and its results in detail.

In D6.3.1, not only the contents and status of the initial PABS have been described, but also the following aspects:

- Integration of codes into the applications benchmark suite;
- Synthetic benchmark suite;
- Performance analysis tools.

The integration of codes into the applications benchmark suite is meant to enable easy future usage of the PABS. To verify correct integration of the benchmark codes into the PABS, we have defined and executed a testing procedure, which can be viewed as an internal acceptance test for the integration of the benchmark codes.

Based on the survey in D6.3.1 on synthetic benchmarks, chapter 4 reports on the actual contents of the synthetic benchmark suite to be used in PRACE. The synthetic benchmark suite in PRACE is designed such that all performance aspects of petascale systems can be tested. In chapter 4 the contents of the synthetic benchmark suite will be described, while assessment of the systems will be reported upon in the deliverables of WP5.

It has been shown in D6.3.1 that there are numerous available Performance Analysis Tools. Given the number of prototype platforms (as defined by WP7 and used in WP5 and WP6), it is practically impossible for the available human resources to investigate each Performance Analysis Tool (PAT) on each prototype. We have chosen to investigate each PAT on at least one of the prototype architectures, and also the other way around: each prototype will be used by at least one PAT.

Apart from substantial technical work on the benchmark codes and subtasks in task 6.3, considerable organisational effort was needed for the definition and distribution of benchmark codes, to monitor progress, and to collect results. We followed the BCO concept, which has turned out to be very successful.

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 different people contribute to shared activities. Within WP6, we have continued 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 administrative overhead. It provides an integrated Wiki, an interface to version control systems, and a number of convenient ways to stay on top of events and changes within a project. For more details, we refer to D6.1.

### 3 Final Benchmark Suite

In the May-June 2008 timeframe, a variety of criteria could have been chosen to define the contents of the initial PABS. The most obvious criteria include coverage of application areas, actual usage of applications in Europe, and scalability potential of the applications to Petascale systems. In WP6, it was decided to use an analytical approach: based on a survey of application areas and particular applications, an initial PABS was defined. This could be done in a relatively short period of time, hence enabling the launch of other tasks within WP6 rather quickly. Practical investigation of scalability of applications towards Petascale level is important but takes much more time. It was therefore decided to adapt the initial PABS to the final PABS, based on, e.g., actual scalability results of the codes. This section covers the process that WP6 has taken to evaluate the initial PABS and to build up the final PABS.

From a logistical point of view, the actual information was collected by the WP6 task leaders during the April-June 2009 time frame. This has been summarized and discussed in a WP6 face-to-face meeting in Helsinki on June 9 and 10, 2009. Some additional information has been collected on some codes, after which a proposal for the final PABS has been prepared. All WP6 participants viewed this proposal as the best way forward, and accepted it without objections at the end of June 2009.

#### 3.1 Starting Point: Initial Benchmark Suite

Deliverable D6.3.1 documented the contents of the initial PABS. Based on a survey on actual usage of codes on European HPC systems, we have identified the most important application areas and most used applications in these areas. Full results can be found in D6.1 and D6.3.1.

As a reminder, Table 1 contains a listing of the benchmark codes, application areas and the responsible BCO, as covered in the initial PABS.



Application	Application Area	BCO
QCD	Particle physics	FZJ
VASP	Computational chemistry	BSC
NAMD	Computational chemistry	EPSRC
CPMD	Computational chemistry	BSC
Code_Saturne	Computational fluid dynamics	EPSRC
GADGET	Astronomy and cosmology	LRZ
EUTERPE	Plasma physics	BSC
ECHAM5	Atmospheric modelling	CSCS
NEMO	Ocean modelling	NCF
CP2K	Computational chemistry	CSC
GROMACS	Computational chemistry	CSC
NS3D	Computational fluid dynamics	HLRS
AVBP	Computational fluid dynamics	GENCI
HELIUM	Computational physics	EPSRC
TRIPOLI_4	Computational engineering	GENCI
PEPC	Plasma physics	FZJ
GPAW	Computational chemistry	CSC
ALYA	Computational mechanics	BSC
SIESTA	Computational chemistry	BSC
BSIT	Computational geophysics	BSC

Table 1: Overview of initial PABS.

In D6.3.1, we have introduced the concept of a core list (applications marked green in Table 1 and an extended list (applications marked yellow). Also in D6.3.1, we have reported on the actual porting status of the benchmark codes on the PRACE prototype architectures, as of November 2008. Table 2 repeats these results:

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		
EUTERPE	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		
NS3D		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 2: Porting status of initial PABS, as of November 2008.

Once the contents of the initial PABS had been defined, several other tasks in the PRACE project could take off. These tasks were 6.4 (petascaling of applications), 6.5 (optimisation of applications) and 5.4 (assessment of prototype systems). Within WP6, we have defined June 2009 as the month to collect results from these tasks, together with the results of 6.3 with respect to porting the codes. This choice has enabled both a significant amount of time to work on the initial PABS and to obtain valuable information, as well as leaving significant time to incorporate potential new applications into the final PABS. In practice, this means that BCOs have been able to work on their benchmark codes for 6 to 12 months, so it can be expected that relevant information is available. This information, together with licensing information and EC review comments, has been taken as input for defining the final PABS.

In section 3.2, we list the information as mentioned above. Section 3.3 analyses the information, while section 3.4 discusses the final PABS.

### 3.2 Experiences with the Initial Benchmark Suite

This section covers the various pieces of input required for a well-balanced analysis of the initial PABS, which ultimately leads to the definition of the final PABS. The input consists of:

1. Actual porting status of the benchmark codes;
2. Survey of licensing policy for the benchmark codes;
3. Scalability results of the benchmark codes;
4. EC review comments:
  - Increase the pool of applications;
  - Investigate industrial usage;

The first piece of input for working towards a final PABS is the actual porting status of the benchmark codes as of June 2009 to different prototype architectures. These results are listed in Table 3.

Application	MPP-BG	MPP-Cray	SMP-TN-x86	SMP-FN-pwr6	SMP-FN+Cell	SMP-TN+vector
QCD	Done	Done		Done		
VASP	Done			Done	Stopped	Stopped
NAMD	Done	Done		Done	To start	
CPMD	Done			Done	Done	Done
Code_Saturne	Done	Done		Done	Stopped	Done
GADGET	Done		Done	Done		
EUTERPE	Done			Done	In progress	
ECHAM5	Stopped	Done	Stopped	Done		To start
NEMO	Done	Done		Done		Done
CP2K	Done	Done		Done		
GROMACS	Done	Done		Done		
NS3D		Done	In progress	Done		Done
AVBP	Done		Done	Done		
HELIUM	Done	Done		Done		Done
TRIPOLI_4	In progress		Done			
PEPC	Done	Done		Done		
GPAW	Done	Done		Done		
ALYA				Done	Done	
SIESTA					Done	
BSIT					Done	

Table 3: Porting status of initial PABS as of June 2009.

The second piece of input we have collected is a survey on the licensing policy for the benchmark codes. This basically comes down to what type of licensing is applied by the code owners and developers. The following aspects are relevant within the current PRACE project, but also when benchmarking is required in the follow-up implementation phase of PRACE:

- Usage of the code, scripts and input sets in current PRACE deliverables and in potential future benchmark efforts;
- Publication of results with respect to scalability and optimisation of the code in current and potential future PRACE deliverables.

This information has been obtained in direct contact with the code developers and is shown in Table 4 under the column licensing policy. The colour scheme for licensing is: orange in case of (severe) restrictions or no response from code developers, yellow with light obstacles and green for basically public licensing.

The third piece of input has to do with the actual scalability results of the benchmark codes on one or more of the prototype platforms. The BCOs have been doing these experiments in the context of task 5.4 in the April-June 2009 time frame. Within task 5.4, it has been observed that comparing scalability results on the different prototype platforms should not be done by numbers of cores only, since the individual core performances of the various prototypes differ significantly. To overcome this problem, task 5.4 has defined scalability as follows:

*An application scales to  $n$  Tflop/s if it achieves a speed-up in excess of 1.6x between running on a system partition of peak performance of  $n/2$  TFlop/s and running on a system partition of  $n$  TFlop/s.*

Before analysing the scalability results, it is important to define a threshold for sufficient scalability. This threshold should be such that a benchmark code which passes the threshold has potential for further scalability towards Petascale systems. We have decided to use the threshold at  $n=10$  Tflop/s, as has been explained in D5.4. The reason this approach has been taken is that the size of the smallest PRACE prototype system is between 10 and 20 Tflop/s, system peak performance. This means that a code which passes this number shows a speed-up of at least 1.6 when going from a 5 Tflop/s system partition to a 10 Tflop/s system partition, according to the definition above. In terms of IBM POWER6 (SMP-FN-pwr6) cores, for instance, this means a speed-up of at least 1.6 when going from 266 cores to 532 cores<sup>1</sup>. For MPP-BG, MPP-Cray and SMP-TN-x86 the number of cores is higher, for SMP-FN+Cell and SMP-TN+vector the number of cores needed is lower. We have used this threshold in Table 4 for a colour scheme: we have used green for the system which shows maximum scalability above the threshold of 10 Tflop/s, and orange for staying below the threshold. Blank cells indicate that there are no results for the code on the particular hardware platform – given the limited resources, it was not foreseen to run each code on each platform. Table 4 shows the results.

The fourth piece of input considers the EC review comments. These include extension of the pool of applications towards additional ones with global scientific and industrial relevance. Looking into the initial PABS, one may argue that the areas of earth sciences and engineering are not well represented. Therefore, to expand the pool of applications, we have looked into applications in these areas, which could fill the gap, but which also have suitable scaling and licensing characteristics. An additional advantage here is that typically applications from earth sciences and engineering tend to be used outside of academic environments as well, e.g.,

<sup>1</sup> 5 Tflop/s divided by 18.8 Gflop/s (peak POWER6 core) equals 266 cores.

meteorological offices (non-commercial and commercial), oil companies (seismic processing) and engineering companies.

Another aspect that plays a role here is the replacement of codes from the initial PABS. Where this turns out to be the case (see section 3.3), our policy is to replace a benchmark code with a code which basically covers the same application area.

Application	MPP-BG	MPP-Cray	SMP-TN-x86	SMP-FN-pwr6	SMP-FN+Cell	SMP-TN+vector	Licensing and response
QCD	n=80-160, depending on kernel	n=20		n=10			OK.
VASP	n<10			n<10			Difficult. No response from developers.
NAMD	n=5	n=20		n=20			OK.
CPMD				n=10	n<10	n=10	OK.
Code_Saturne	n=80	n=10		n=20			OK, GNU General Public License.
GADGET	n=10			n=40			OK.
EUTERPE	n=80			n=40			OK.
ECHAM5		n<5		n<5			No response from developers.
NEMO				n=10			All OK, under CeCILL licence.
CP2K		n=80 (new version)		n=5 (old version)			OK, GNU General Public License (no execs without sources).
GROMACS	n=20	n=40		n=40			OK, GNU General Public License.
NS3D		n=20		n=20		n=10	OK.
AVBP	n=20		n=40				OK, but also between benchmarker and CERFACS
HELIUM	n=10	n=10		n=20			OK, but no publication on analysis of code.
TRIPOLI_4			n>10				OK, but also between benchmarker and CEA
PEPC		n=5		n=10			OK, GNU General Public License.
GPAW	n=40	n=20		n=20			OK, GNU General Public License.
ALYA				n=20	n=10		OK.
SIESTA							OK.
BSIT					n=10		OK.

Table 4: Actual task 5.4 scalability results and licensing information on the initial PABS codes.

### 3.3 Analysis and update of the Initial Benchmark Suite

This section analyses the results as obtained in section 3.2, and which have been referred to as pieces of input.

First, comparison of Tables 2 and 3 shows that much progress has been made with porting the applications to the prototype platforms. Both tables reflect the actual fact that MPP-BG, MPP-Cray and SMP-FN-pwr6 are the most commonly used general-purpose platforms in the top of the HPC arena. Systems with vectorprocessors are typically doing well for selected applications (many of these CFD-based). The results also show that porting applications to the SM-FN+Cell architecture appears to be a time consuming task, due to both its lack of programming tools and its non-standard programming model, and which requires a lot of manpower.

Secondly, with respect to licensing, it has turned out to be very difficult to get response from the VASP developers. This has been tried multiple times. For ECHAM5, the BCO left his institute, and could not be replaced in due time.

Thirdly, the scalability results in Table 4 show that three of the applications remain under the 10 Tflop/s threshold: VASP, ECHAM5 and SIESTA. It has been impossible in the available timeframe to obtain scalability results for these codes which pass this threshold, for which the reason is basically twofold: lack of scalability of the numerical algorithm and an implementation which is not advantageous to parallel processing.

The fourth aspect of the previous section covered the EC comments on extension of the pool of applications towards further global and industrial usage of the applications. We have identified the current industrial usage in Table 5:

Application area	Actual applications	Industry usage
Chemistry	NAMD, CPMD, CP2K, GPAW, Gromacs, Helium, VASP, SIESTA	Chemical, pharmaceutical and life sciences (typically each company has its “own flavour”)
Computational fluid dynamics	Code_Saturne, NS3D, AVBP	Engineering industry, combustion, gas turbines, nuclear power, aerospace
Earth sciences	BSIT, ECHAM5, NEMO	Meteorological and seismic communities
Engineering	ALYA, TRIPOLI-4	Similar to CFD codes, energy (nuclear power) industry
Physics/astrophysics/cosmology	GADGET, PEPC, QCD, EUTERPE	Scientific codes, in general not used by industry

**Table 5: Industrial usage of the initial PABS applications.**

The result of our analysis was that we considered replacing VASP, ECHAM5 and SIESTA by alternatives. As has been discussed in the previous section, we searched for possible replacements of these three codes, which had to satisfy the following criteria:

- Scalability above the 10 Tflop/s threshold;

- Public licensing or comparable;
- Industrial usage and/or interest;
- Availability of BCO to do the work.

This has led to the following replacement codes (Table 6):

<b>New Code</b>	<b>Replaces</b>	<b>Scalability</b>	<b>Licensing</b>	<b>BCO</b>
Quantum_Espresso	VASP	n=20 on SMP-FN-pwr6	OK. GNU Public License	CINECA Italy
WRF	ECHAM5	n>100 on MPP-Cray	OK	STFC UK
Octopus	SIESTA	n=20 on SMP-FN-pwr6	OK GNU Public License	UC-LCA Portugal

**Table 6: Replacement codes for the final PABS.**

Alternatives for the codes in Table 6 have been considered: alternatives for Quantum\_Espresso and Octopus have been MPQC and NWCHEM; for WRF, this has been IFS.

Extensions of the pool of applications have been searched for in the areas of earth science and engineering. For an application to qualify, we have used the same criteria as for the replacement of the codes. This has led to Table 7:

<b>Code</b>	<b>Application area</b>	<b>Scalability</b>	<b>Licensing</b>	<b>BCO</b>
ELMER	Multi-physics engineering	n=40 on MPP-Cray	OK GNU Public License	CSC Finland
SPECFEM3D	Earthquake simulation	n=160 on MPP-Cray	OK GNU Public License	CINES France

**Table 7: Extension codes for the final PABS.**

### 3.4 Final Benchmark Suite

The analysis and update process of the initial PABS towards the final PABS, as described in the previous section, has led to the following applications which form the final PABS (Table 8):

Application	Application Area	BCO
QCD	Particle physics	FZJ
Quantum_Espresso	Computational chemistry	CINECA
NAMD	Computational chemistry	EPSRC
CPMD	Computational chemistry	BSC
Code_Saturne	Computational fluid dynamics	EPSRC
GADGET	Astronomy and cosmology	LRZ
EUTERPE	Plasma physics	BSC
WRF	Atmospheric modelling	EPSRC
NEMO	Ocean modelling	NCF
CP2K	Computational chemistry	CSC
GROMACS	Computational chemistry	CSC
NS3D	Computational fluid dynamics	HLRS
AVBP	Computational fluid dynamics	GENCI
HELIUM	Computational physics/chemistry	EPSRC
TRIPOLI-4	Computational engineering	GENCI
PEPC	Plasma physics	FZJ
GPAW	Computational chemistry	CSC
ALYA	Computational mechanics	BSC
OCTOPUS	Computational chemistry	UC-LCA
BSIT	Computational geophysics	BSC
ELMER	Computational engineering	CSC
SPECFEM3D	Computational geophysics	GENCI

**Table 8: Overview of final PABS.**

Table 8 shows that the actual coverage of the application areas is in accordance with the areas that have been identified in the scientific case report [3], as prepared in 2006. Compared to the initial PABS, we have strengthened the fields of earth sciences and engineering, as is shown by the replacement of ECHAM5 by WRF, and the addition of ELMER and SPECFEM3D.

With respect to the area of computational chemistry, this area contains 8 applications out of 22. On an absolute basis, this is a large number, which can be justified by the fact that computational chemistry applications are using 30-40% (see for instance D6.2) of the available cycles on many of the large national supercomputing installations (and in some cases even more). Also, computational chemistry can be viewed as a collective name for molecular dynamics, ab initio calculations, density functional theory, Car-Parinello modeling and even life sciences applications.



Finally, the actual porting status of the applications in the final PABS is shown in Table 9 (with indicated the progress ('New') since October 2009):

Application	MPP-BG	MPP-Cray	SMP-TN-x86	SMP-FN-pwr6	SMP-FN+Cell	SMP-TN+vector
QCD	Done	Done	Done	Done		
Quantum_Espresso	Done	Done	New	Done		Done
NAMD	Done	Done	Done	Done		
CPMD	Done			Done	Done	Done
Code_Saturne	Done	Done	New	Done		
GADGET	Done		Done	Done	New	
EUTERPE	Done	New		Done	New	
WRF	Done	Done	New	Done		
NEMO	Done	Done		Done		Done
CP2K	Done	Done		Done		
GROMACS	Done	Done		Done		
NS3D	New	Done		Done		Done
AVBP	Done		Done	Done		
HELIUM	Done	Done	Done	Done		Done
TRIPOLI_4			Done			
PEPC	Done	Done	Done	Done		
GPAW	Done	Done	New	Done		
ALYA		New		Done	Done	
OCTOPUS	New	New	New	Done		
BSIT					Done	
ELMER		Done	New	New		
SPECFEM3D			Done			

Table 9: Actual porting status of the final PABS per June 2010.

Blank fields in Table 9 denote that the particular code has not (yet) been ported to the hardware platform. For some applications that were “in progress” in December 2009, it has turned out to be not feasible to complete the porting in the extension phase. The fields have become blank as well. Note that actual assessment of the prototype systems with the PABS have been reported in D5.4.

### 3.5 Conclusions and future work

In this chapter, we have described the process from the initial PABS towards the final PABS, with explicit reference to the various sources of input needed to guide this process. We have created a final PABS which has the following characteristics:

- Coverage of all scientific application areas, which need sophisticated HPC equipment today and in the near future;
- Coverage of a broad range of algorithmic classes;
- Balanced coverage of all PRACE prototype architectures, as defined in WP7 (production systems in 2010);
- Each application has the potential to scale to petascale systems;
- Flexible and tested benchmark environment;
- Potential to combine all applications into a throughput benchmark, depending on the anticipated type of architecture to be purchased;
- Flexible enough to enable usage of subsets of the PABS in actual procurements.

Apart from a technically sound benchmark, there are more aspects which need to be covered when actually using the benchmark in a procurement process. These include:

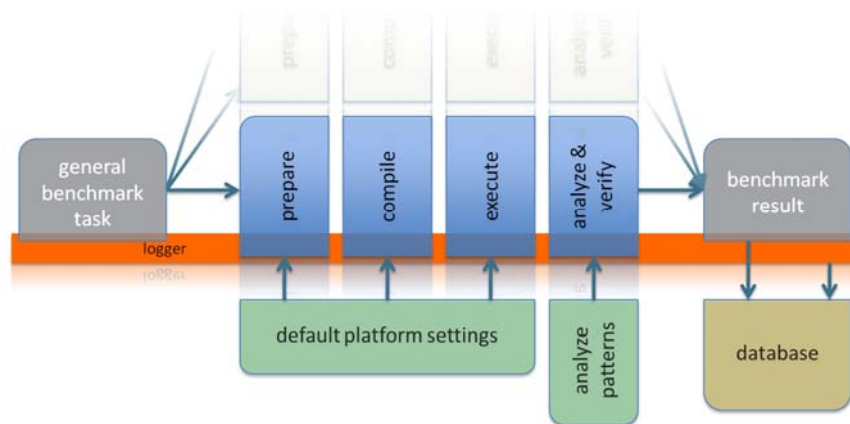
- The actual weight the benchmark results will have in the procurement process. This not only holds for the application results, but also for the synthetic benchmark results;
- The actual weight of each application in the benchmark, which may typically be influenced by requirements from the actual Hosting Partner.

With respect to the contents of the PABS, it must always be checked against actual practice and usage. This means the PABS needs to be adapted and/or extended if needed, to remain relevant while covering all aspects we have used to come to the current final PABS.

## 4 Integration in Benchmark Suite

The benchmark codes from WP6 need to be integrated into a suite as part of task 6.3, in order to facilitate usage within other tasks in WP6, within other work packages within PRACE and in future projects or procurements. 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. The basic integration strategy has already been presented in deliverable D6.3.1. This chapter describes the final integration status, the inclusion of performance measuring tools into the benchmark suite as well as the final acceptance test.

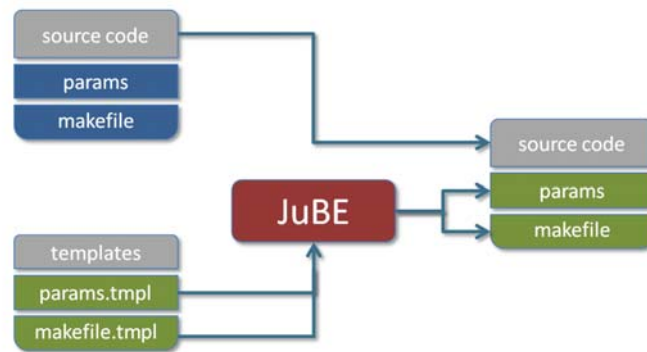
To recall the basic strategy of the chosen benchmarking framework JuBE [4], the framework's workflow and strategy is shown in Figure 1 and Figure 2; a detailed description is given in D6.3.1 and references therein.



**Figure 1: JuBE workflow**

The usage idea of JuBE, or its workflow, is to set up a benchmark (suite), in which each benchmark is defined in a coherent way. All steps to realize the benchmark results are processed by JuBE, these steps are: preparation, compilation, execution, verification and analysis. These steps are performed for all runs needed to fulfil the general benchmark task. A benchmark script runs on a variety of partition sizes, automatically sets up the appropriate number of jobs, submits them to the supercomputer and then combines the results into one or more tables. This leads to a multidimensional parameter space which allows for a simple way of scanning large parameter spaces. Thus a scaling benchmark at various compiler optimization levels and problem sizes, can be defined in one step.

The strategy to realize the automatic execution of the steps between the general benchmark definition and the benchmark result is to utilize templates, including all relevant parts (files) of the application, i.e. those which might include varying values need to be represented by templates. These templates will be used to generate the final versions for each point in the parameter space. Figure 2 illustrates this strategy.



**Figure 2: JuBE substitution strategy**

Figure 2 shows a template for some application parameters and for the makefile. The way JuBE works is that, once the template files, here `params.tmpl` and `makefile.tmpl`, are provided together with the according substitution rules, no further modifications are needed. In this example the content of the template files would be substituted and would replace the files `params` and `makefile` needed by the application. This substitution is done, if needed, for each point in the parameter space.

Section 4.1 provides an integration overview, containing information on how to get the benchmark suite and the current integration status. An example on how to include performance measurement tools into an integrated benchmark is presented in section 4.2, which is followed by the description of the carried out acceptance test (section 4.3).

## 4.1 Integration Overview

The benchmark suite is currently available in a subversion repository

**`https://trac.csc.fi/pracewp6`**

hosted by CSC. As the benchmark suite is currently not freely available, but restricted to PRACE internal use only, access permission must be granted.

Access to the subversion repository is possible on all PRACE prototypes via the following command

```
$> svn checkout https://trac.csc.fi/pracewp6/svn/benchmark
```

The current size is in the order of 3 GB, which does not include the applications' source code, as these must be gathered separately. Information on how to get each application's source code is located in each benchmark's directory, i.e. as a **`HowToGet.txt`** file. In most cases, these files contain commands to checkout the source code from external subversion repositories.

As already stated in PRACE deliverable D6.3.1 each benchmark has to provide general information on how the benchmark should be run. This information is located in the top-level xml files. To follow a unified, i.e. standardized way of benchmarking, the aims and requirements of such a benchmark run have been defined. The following section (4.1.1) describes this definition. In section 4.1.2 an overview of the current integration status is given.

## Benchmark Execution Standardization

In order to standardize the benchmark suite and thus ease its use for non-BCOs, the following two predefined benchmark rules are set up. These benchmark types can be used as starting points for future benchmarking tasks as well as reference values in verification procedures. In addition to these aims, the two benchmark runs are the minimal required integration level and will be used for the acceptance test, see section 4.3.

### a. Functional Benchmark

This setup should just verify that the benchmark is able to be compiled and executed on the target architecture. The problem size and the computing partition is chosen to be small; thus this test should consume only very little resources. However, the verification and analysis steps have to be considered too. The result description file (**result.xml**) contains the following table definition:

```
<result>
  <show active = "1" colw="10" title="prace reference timing">
    ncpus, time, vcheck
  </show>
  ...
</result>
```

This unifies the benchmark output. In this example, it is the number of tasks, benchmark timing and the verification flag, to be able to directly compare the timing with the reference values measured by the BCOs. The timing stated by the benchmark is defined by the BCOs; typical examples are the total execution time, the time in the inner loop of an application or the average time for a single main loop. The top-level xml file is named **prace-functional-\$ARCHNAME.xml** and the command to start the benchmark from the application's directory is

```
$> ../../bench/jube -start prace-functional-$ARCHNAME.xml
```

### b. Scaling Benchmark

The only difference to the functional benchmark aim is to demonstrate the benchmark's (strong) scaling behaviour on small and medium size partitions. The problem size may be adapted as well. According to the functional benchmark, the naming of the benchmark definition file is **prace-scaling-\$ARCHNAME.xml** with a similar initialization command.

Thus the following steps are needed to run these benchmarks from scratch. The place holders **\$BMNAME** and **\$ARCHNAME** have to be replaced by the target benchmark and prototype.

1. Get the benchmark suite; access rights assumed.  
\$> **svn checkout https://trac.csc.fi/pracewp6/svn/benchmark PABS**
2. Change to the benchmarks directory.  
\$> **cd PABS/applications/\$BMNAME/**
3. Get the benchmarks source code; follow the instructions in **HowToGet.txt**
4. Start the functional benchmark on the target prototype  
\$> **../../bench/jube -start prace-functional-\$ARCHNAME.xml**  
or the scaling benchmark  
\$> **../../bench/jube -start prace-scaling-\$ARCHNAME.xml**
5. Obtain the results.  
\$> **../../bench/jube -update -result**

### Integration Status

The minimal integration requirements for a benchmark are the realization of the two benchmarks described in section 4.1. The integration status for each benchmark on each prototype is shown in Table 10: Integration status as of June 2010.

These levels are indicated by green (standardized integration), and red (no final integration available yet), whereas blank cells indicate that the application has not (yet) been ported to this platform, also as a result that efforts on certain applications had to be stopped in the extension phase. Compared to the previous version of D6.3.2, we have checked the integration of the applications and have adapted their status from completed (green) to almost completed (which is red).

application	MPP-BG	MPP-Cray	SMP- ThinNode- x86	SMP- FatNode- pwr6	SMP- FatNode+ Cell	SMP- ThinNode+ Vector
QCD						
Quantum_Espresso			New			
NAMD						
CPMD						
Code_Saturne	New	New	New	New		
GADGET			New		New	
EUTERPE					New	
WRF						
NEMO						
CP2K						
GROMACS						
NS3D						
AVBP						
HELIUM						
TRIPOLI-4						
PEPC						
GPAW						
ALYA						
OCTOPUS	New	New	New	New		
BSIT						
ELMER		New	New	New		
SPECFEM3D						

Table 10: Integration status as of June 2010.

Further information on how to use the benchmark, the exact measurement and the main benchmarking parameters are given in the **README** file in each application's directory.

## 4.2 Extensions to standard Integration

The usage of the benchmarking framework JuBE is not restricted to the variation of compilation or input parameters, but also allows the direct utilization of performance measuring tools. With this integration it becomes very easy to collect data of the application's performance on a wide parameter regime and to allow comparison between runs with and without tools.

The main strategy is to include flags which alter, for example, the compilation or which add libraries into the linkage stage. The exact impact of such flags depends on the performance tool they are attached to. During the analysis step, the tool's output can be parsed as usual and displayed in the result phase. To illustrate this strategy in more detail, a representative integration example of the GNU profiler [5] is given hereafter.

The GNU profiler gprof is a profiling tool which analyses the application's procedure call distribution. This tool provides data on the time spent in a procedure, the number of each procedure calls as well as a calling graph. More information on this tool is given at the above referenced site.

To generate a procedure profile, the compiler needs to instrument the code: the flags to do so are, in this example for the IBM XL compiler, **-g -pg**. There is no need to include any libraries and the executable can be run as usual. After the execution, there are some profile output files in the executable's current directory. As the last step, these files have to be processed by gprof to generate a human readable profile.

The steps needed to integrate gprof might be the following:

1. Add a flag at the parameter section of the top-level xml file, to switch on/off the profiler.
2. Add additional compiler flags, depending on the flag status.
3. Provide a script to run the profiler in the analysis step.
4. Provide patterns to grab the profile output.
5. Add a profiler table to the result section.

These steps will be demonstrated by an example, which will be kept as general as possible, but some explicit code references are based on the PEPC integration on the PRACE prototype system at SARA (IBM POWER6, Huygens).

### Step 1

First of all, a gprof flag (**GPROF**) has to be defined in the top-level xml file.

```
<params
...
GPROF = "on,off"
... >
```

This statement will add another dimension to the parameter space, namely the usage of gprof. In this example, this will trigger all benchmark runs to be run with and without gprof. As gprof influences the compilation step, the compilation has to be triggered if the parameter **GPROF** changes. To realize this, a new variable **CMP\_GPROF** is defined in the compile section of the top-level xml file. Note that this is just a dummy variable to determine if a recompilation is needed.

```
<compile
```

```
...
CMP_GPROF = "$GPROF"
... >
```

### Step 2

With the definition in step 1, the **GPROF** flag will be used to determine additional compilation arguments. A possible realization in the **compile.xml** file is to extend the substitution of the compiler flags by

```
<sub from = "#F90FLAGS#"
      to = "$f90flags $optflags
           `index('$GPROF','on')==0 ? '-g -pg' : ' '`" />
```

This substitution demonstrates the usage of the perl evaluation possibility in JuBE. The placeholder **#F90FLAGS#** will be substituted by the values of the two variables **\$f90flags** and **\$optflags**, followed by an additional **-g -pg** option, but only if the value of **GPROF** is equal to **on**.

### Step 3

After the successful execution, the profiler data has to be processed. This will be done in the analysis phase. During this phase, a shell script will be called to generate a readable profile, via **gprof**, and from this a JuBE parsable file, i.e. containing unique text line identification. As the **gprof** output differs on the various architectures, a template is needed which considers different architectures. A possible structure might be the following template file (**collectData.sh.jube**)

```
...
#COLLECT_GPROF# #HUYGENS# gprof #EXECUTABLE# profdir*/gmon.out
> GPROF.dat; \
#PERL# #BENCHHOME#../../../../utils/gprof/parseGPROF.pl -1 1
GPROF.dat > GPROF.log

#COLLECT_GPROF# #JUGENE# gprof #EXECUTABLE# gmon.out.*
> GPROF.dat; \
#PERL# #BENCHHOME#../../../../utils/gprof/parseGPROF.pl -1 1
GPROF.dat > GPROF.log

#COLLECT_GPROF# #JUMP# gprof #EXECUTABLE# gmon.*.out
> GPROF.dat; \
#PERL# #BENCHHOME#../../../../utils/gprof/parseGPROF.pl -2 1
GPROF.dat > GPROF
...
```

However, before the script can be used in the analysis phase, all placeholders must be substituted in the preparation phase. The basic idea is to substitute **#COLLECT\_GPROF#** by the **#**-character if **GPROF** is not on. In addition to this, only the current architecture placeholder (e.g. **#HUYGENS#**) is replaced by a **`**-character, leaving only one command line to be executed. The substitution of the remaining placeholder is straight forward. The first part of an execution line is the call of **gprof** to generate the **GPROF.dat** file, which contains the collected readable profile. In the second step, the perl script **utils/gprof/parseGPROF.pl** is used to label each line and thus to be able to use the



search pattern shown in the next step. This final information will be stored in the file **GPROF.log**, which leads to the following extension of the preparation step:

```
<substitute infile="collectData.sh.jube" outfile="collectData.sh">

  <sub from="#BENCHHOME#"          to="$benchhome" />
  <sub from="#EXECUTABLE#"         to="pepc.exe" />

  <sub from="#PERL#"               to="$PERL_CMD" />

  <sub from="#COLLECT_GPROF#" to="\index('$GPROF','on')>-1 ? ' ' : '#`' />
  <sub from="#JUGENE#" to="\index('$platform','Jugene')>-1 ? ' ' : '#`' />
  <sub from="#JUMP#" to="\index('$platform','Jump')>-1 ? ' ' : '#`' />
  <sub from="#HUYGENS#" to="\index('$platform','Huygens')>-1 ? ' ' : '#`' />
</substitute>
```

Then, the analysis step needs the following additional command:

```
<analyse cname="IBM-SP6-Huygens">
  <precommand>(cd $outdir; bash collectData.sh)</precommand>

  <input addfiles="$subdir/GPROF.log" />

  <includepattern file="./patterns-gprof-pepc.xml" />
</analyse>
```

First, an initial command will be executed, i.e. change to the working directory and execute the prepared data collection script. As described above, the collection script produces the file **GPROF.log**, which is added to the list of files to be parsed for the pattern defined in the **patterns-gprof-pepc.xml** file, as described in step 4:

#### Step 4

The preprocessed file can now be parsed using these JuBE patterns:

```
<parm name = "GPROF_01_NAME" unit = "" mode = "line,last"
type="string" >JuBE: gprof: proc 1:\s*$patwrd\s*$patnfp</parm>

<parm name = "GPROF_01_PART" unit = "%" mode = "line,last"
type="float" >JuBE: gprof: proc 1:\s*$patnwrd\s*$patfp</parm>
...
```

The variables **GPROF\_01\_NAME** and **GPROF\_01\_PART** will contain the name and the execution time fraction of the first procedure, sorted by the time spent in this procedure. This pattern list can now be extended to the desired depth.

#### Step 5

The final step is to report on this data, by including a section to the **result.xml** file in the following way:

```
<show active="1">
  GPROF_01_NAME, GPROF_01_PART, GPROF_02_NAME, GPROF_02_PART,
  GPROF_03_NAME, GPROF_03_PART
</show>
```

This will produce a table containing the function names and the total time fraction of the three (in this example) functions consuming most of the CPU time.

### 4.3 Acceptance test

The joint efforts of a large number of BCOs in building up the large benchmark suite PABS might result in an inhomogeneous usage quality. To ensure a high quality, i.e. all benchmarks are fully working also for non-BCOs and are able to reproduce the reference timings, an acceptance test for each code in the full benchmark suite was organised. This quality check ensures that future users, which in general are not the BCOs, of the benchmark suite are using a cross-checked benchmark suite and will thus be confronted with fewer difficulties. By executing the quality check as a non-BCO, all parts which might depend on the knowledge or setup of the BCO, e.g. environment variables or file permissions, are removed. As the reference timing is provided, not only will the functionality be ensured, but also a general estimate of the performance to make sure that the benchmark performance is similar to that measured by the BCO.

#### Testing Procedure

The testing procedure consists of the following steps:

1. Each BCO chooses a problem size suited to run the functional and scaling benchmark described in 4.1.1 and generates the reference timing. This timing, together with the used subversion repository status (PABS and benchmark source code) is reported to the testing team;
2. The testing team will execute both benchmarks (functional and scaling) on default accounts at the corresponding prototypes and will compare the timings to the BCO reference values. The reference timings are provided in the WP6 trac system.

#### Testing Status

At the time of writing this deliverable, not all benchmarks have yet been tested. The current status is shown in Table 11. Green cells mean that the actual test has been passed, red cells indicate that the tests have not been completed yet and blank cells will not be filled, since the particular combination of application and platform does not exist.

### 4.4 Conclusions

The PRACE Application Benchmark Suite (PABS) has been successfully integrated in the JuBE (Juelich Benchmarking Environment) framework. As shown in this section, the benchmarking environment may also be used for extended automated tasks, like the usage of performance measuring tools; indeed, some of the benchmarks integrated in PABS did reach this advanced integration status. The integration is a dynamical process, in which new code versions or integration features are added continuously. In order to define a reference point, an acceptance test procedure has been defined and executed. Although this test is still in progress, it became clear that it is of high importance to finish it and thus guarantee a high level of quality. It should be noted that the JuBE integration and acceptance tests were not part of the original work plan, but was introduced as an additional measure to ensure the long-term sustainability of PABS. Therefore the still incomplete acceptance test does not influence the fulfillment of the work package's work plan.

Application	MPP-BG	MPP-Cray	SMP- ThinNode- x86	SMP- FatNode- pwr6	SMP- FatNode+ Cell	SMP- ThinNode+ Vector
QCD	3.11.2009	17.11.2009	3.11.2009	5.11.2009		
Quantum_Espresso	12.11.2009	26.11.2009	2.12.2009	21.12.2009		
NAMD	26.11.2009	26.11.2009	11.11.2009	6.11.2009		
CPMD	9.12.2009			21.12.2009	9.12.2009	
Code_Saturne	12.01.2010	19.01.2010	1.6.2010	20.05.2010		
GADGET	2.12.2009		2.12.2009	4.12.2009		
EUTERPE	1.12.2009	25.05.2010		4.12.2009	17.12.2009	
WRF	8.12.2009	8.12.2009	8.12.2009	23.12.2009		
NEMO				14.12.2009		
CP2K	18.11.2009	25.11.2009		21.12.2009		
GROMACS	17.11.2009	22.11.2009		8.12.2009		
NS3D		8.12.2009		21.12.2009		12.12.2009
AVBP	9.11.2009		9.11.2009	23.12.2009		
HELIUM	9.12.2009	23.11.2009	9.11.2009	21.12.2009		
TRIPOLI-4						
PEPC	3.11.2009	17.11.2009	3.11.2009	6.11.2009		
GPAW	16.11.2009	26.11.2009	16.11.2009	21.12.2009		
ALYA				23.12.2009	4.12.2009	
OCTOPUS						
BSIT						
ELMER		26.11.2009	25.05.2010	25.05.2010		
SPECFEM3D						

Table 11: Status of acceptance tests as of June 2010.

## 5 Synthetic Benchmarks

### 5.1 Introduction

The analysis of synthetic benchmarks in D6.3.1 presented the relevance of using synthetic benchmarks in HPC performance measurements to complement application benchmarks, outlined the key performance characteristics of HPC systems and listed the most important existing synthetic benchmarks. Based on this analysis a synthetic benchmark set, called PRACE-SBM was proposed.

While the initial proposal consisted of existing benchmarks, it was also realized that there are several key areas for which no benchmarks exist or the current benchmarks lack a key feature. Therefore it was planned that WP6 should pursue implementing these benchmarks. The following benchmarks were successfully implemented:

- Parallelized version of STREAM2 (using MPI)
- OpenMP+MPI hybrid benchmarks
- Parallelized version of the Bonnie++ filesystem benchmark (using MPI)

The first practical application of PRACE-SBM was the evaluation of the PRACE WP7 prototypes in tasks 5.2 and 5.3. There has been close collaboration with the contributors to these tasks in adjusting details of the PRACE-SBM to best meet their requirements. The actual execution of the benchmarks on the prototypes was performed by WP5, while WP6 provided the integration of the benchmarks into JuBE. WP6 also provided support for the WP5 members tasked with running the benchmarks, as well as assisted with the final analysis of the combined results from the different prototype systems.

The next section describes the benchmarks we have chosen for the PRACE-SBM, including their descriptions.

### 5.2 Descriptions of Synthetic Benchmarks

The initial implementation of the PRACE-SBM contains the following individual benchmarks:

1. Computational kernels
  - a. Euroben-shm - OpenMP parallelized kernels
  - b. Euroben-dm - MPI parallelized kernels
2. Internode communication
  - a. SkaMPI - Latency and bandwidth of MPI routines and communication patterns
  - b. SMB - Overlap of asynchronous MPI communication and computation
  - c. MixedMode - Performance when combining OpenMP and MPI
3. Memory
  - a. MPI STREAM2 - Bandwidth of different memory hierarchy levels (MPI parallelized version)

4. Disk I/O
  - a. IOR - Disk I/O bandwidth when using Posix, MPI-IO or HDF5
  - b. MPI Bonnie++ - Disk I/O metadata performance (MPI parallelized version)
5. OS Jitter
  - a. P-SNAP – Variance of computational performance
  - b. Selfish – Amount of individual interruptions in processing
6. Combined benchmarks
  - a. HPCC – A popular benchmark suite for characterizing HPC system performance

The source codes for all the benchmarks in the package are freely available, most under the GNU Public License (GPL).

The synthetic benchmark codes are stored in the TRAC SVN repository and are integrated into the main benchmark suite in the same manner as the application benchmarks. The integration is described in chapter 4.

Due to the large amount of numerical results produced by the benchmarks, the implementation of output processing and analysis with JuBE proved to be difficult. This problem was alleviated by the use of separate postprocessing scripts. Support for external output processors has been added to JuBE, so that these scripts can be integrated into the framework.

At the PRACE general face-to-face meeting in Jülich, WP6 and WP8 made a joint decision that a subset of the Euroben benchmarks would be used for basic evaluation of the WP8 prototypes and T6.6 programming models. The benchmarks chosen were: mod2am (sparse matrix multiply), mod2as (dense matrix multiply) and mod2f (Fourier Transformation). A new branch, called “euroben-ports” was added to the TRAC repository which contains subdirectories for the ports to different architectures. To facilitate easier porting to C and C++-based languages, both serial and MPI parallelized versions of the selected Euroben kernels were ported from Fortran to C.

Actual results of running the PRACE-SBM (as part of the assessment of the PRACE prototype systems by WP5) can be found in D5.2. This section continues with the individual descriptions.

## **Euroben**

For more information, visit: <http://www.euroben.nl/>

The Euroben suite, developed by Aad van der Steen of NCF, is a collection of benchmarks covering a variety of categories:

- Computational kernels (matrix-matrix multiply, FFT etc.)
- Performance and accuracy of intrinsic functions
- Communication performance measurements
- Memory performance measurements

There are serial, OpenMP-parallel and MPI-parallel versions of the suite available. There is some variation between the contents of different versions as each version contains only the parts relevant for it. For example, the MPI-parallel version does not include the serial intrinsic benchmarks.

In the scope of PRACE-SBM, the primary area of interest in the Euroben suite are the serial and parallel performance of various computational kernels as well as measuring the

performance of intrinsic functions. Both MPI and OpenMP parallel versions of the benchmarks were included in the PRACE-SBM package.

Additional ports of selected kernels were made by T6.6 and WP8 to novel programming languages.

### SkaMPI

For more information, visit: <http://linwww.ira.uka.de/~skampi/>

SkaMPI is a benchmark for testing MPI communication performance. The benchmark covers most of MPI 1 as well as parts of MPI 2, including:

- point-to-point communication
- collective communications
- derived datatypes
- one-sided communication
- MPI I/O

While there are several other MPI benchmark suites available, SkaMPI was chosen because it is independent from any specific vendor as well as extensible: Using the built-in scripting language and well-documented programming interfaces, it is relatively easy to add new features.

### SMB

For more information, visit: <http://www.cs.sandia.gov/smb/overhead.html>

SMB (Sandia MPI Benchmark) measures the ability of a system to overlap MPI communication when using asynchronous calls with computation. The measurement is performed by using a post-work-wait loop:

- Call MPI\_Isend() and MPI\_Irecv(), to initiate the respective transfer
- Perform some work on the CPU (busy loop)
- Wait for the transfer to complete using MPI\_Wait()

During each iteration, the amount of work is increased and until the message transfer time is smaller than the work time. The processing overhead can then be calculated by measuring the amount of time used to perform the same amount of work without overlapping a message transfer and subtracting this value from the loop time. Results are reported in both absolute time needed for processing the packet and as a percentage of the processor availability (100% means that communication does not require any intervention from the CPU whereas 0% indicates that communication processing ties up the CPU completely)

As no traditional MPI benchmark suite has facilities to measure MPI overlap, this additional benchmark was needed to complement SkaMPI.

### MixedMode

The MixedMode benchmark, produced by EPCC, is a set of microbenchmarks which measure the ability of the MPI stack to handle OpenMP/MPI hybrid parallelization. The benchmarks include several point-to-point (pingpong, pingping, halo) and collective (scatter-gather, reduction, broadcast, barrier, alltoall) measurements.

## MPI STREAM2

For more information, visit: <http://www.cs.virginia.edu/stream/stream2/>

The Stream2 benchmark measures the bandwidth of 4 memory operations (FILL, COPY, DAXPY, SUM) using a number of problem sizes. The results illustrate the sustained memory bandwidth on all levels of the cache hierarchy.

Using the original, serial version of STREAM2 benchmark to measure aggregate performance of multicore systems by running several instances of it in parallel is time-consuming and prone to inaccuracy: A number of independent output files is produced and there is no guaranteed synchronization between the tasks: Some tasks may complete before others have the chance to start which skews the results as we want to measure contention caused by all the tasks simultaneously.

To address this problem, STREAM2 was parallelized by the subtask using MPI to form MPI STREAM2. In the new version, each MPI task calls the original serial benchmark routines after which and then the results from each task are aggregated into an average. The output style is identical to the original version.

## IOR

For more information, visit: <http://sourceforge.net/projects/ior-sio>

The IOR is the de facto standard tool for measuring disk bandwidth performance in HPC systems. In addition to POSIX IO, it can be used to measure HDF5 and MPI-IO performance.

## MPI Bonnie++

For more information, visit: <http://sourceforge.net/projects/ior-sio>

The Bonnie++ benchmark is a disk benchmark which measures both data and metadata handling performance of disk systems. It was included in the suite for its ability to measure metadata performance, something which IOR lacks. The operations measured are file reads, creates and deletes. Results are in ops/second.

In HPC systems, a number of processes usually tend to metadata operations simultaneously. However, Bonnie++ does not support distributed-memory parallelization which limits its usefulness to do real stress tests on petascale systems. To address this, an MPI parallel version of Bonnie++, called MPI Bonnie++ was produced by the subtask.

## P-SNAP

For more information, visit: <http://www.ccs3.lanl.gov/pal/software/psnap/>

P-SNAP measures the variation in the execution time, or „jitter“, by running a fixed size calculation repeatedly on a single CPU. The magnitude of the variation can be visualized with distribution graphs of the execution times.

### Selfish

For more information, visit: <http://www.mcs.anl.gov/research/projects/zeptoos/projects>

The Selfish measures „detours“: the fraction of time the CPU spends executing instructions not part of the user's application. The output can be examined to determine the length of each interruption. Compared to P-SNAP this is a more qualitative approach to measuring jitter effects. It can be used to track down jitter patterns, such as timer-based interrupts.

### HPCC

For more information, visit: <http://icl.cs.utk.edu/hpcc/>

The HPCC benchmark suite is a widely-used collection of HPC benchmarks. While the HPCC overlaps with a number of benchmarks in the suite, it contains some individual benchmarks (HPL, MPI ring-patterns) which were deemed useful.

## 5.3 Conclusions and Further Work

It is clear that with the further evolution of high-end computer systems from Petaflop/s into Exaflop/s careful performance testing of the basic components of the systems becomes more and more important. With possibly hundreds of thousands, perhaps millions of cores, actual component performance is critical for sustained system performance. Further evolution of synthetic benchmarks must be taken into account as well.

The introduction of other components, like graphical cards, FPGA's and Cell-type of processing goes even further: development of synthetic benchmarks for these types of hardware needs to be started to enable future performance testing as well.



## 6 Performance Analysis Tools

### 6.1 Introduction

It has been shown in D6.3.1 that there are many available Performance Analysis Tools. Given the amount of prototype platforms (6 in total, as defined by WP7 and used in WP5 and WP6), it is practically impossible from an available human resources point of view to investigate each Performance Analysis Tool (PAT) on each prototype platform. We have chosen to re-assess each D6.3.1 PAT on at least one of the WP7 prototype architectures. However, due to limited resources, we did not do any of the assessments on the NEC SX-9 prototype, or to re-assess the CEPBA-Tools (Paraver & Dimemas). Finally, instead of the High-Performance Linpack Benchmark (HPL), for every PAT-WP7 prototype combination, we selected a relevant PABS benchmark as a testbed for the assessment. Most of the PATs have been used in tasks 6.4 and 6.5, leading to more details for task 6.3 as well. Based on these experiences and the additional experiences during this separate assessment, we have been able to report our experiences to the vendors and open source developers.

### 6.2 Allinea Optimization and Profiling Tool (OPT)

#### Introduction

##### Developer

Allinea Software, [www.allinea.com](http://www.allinea.com).

##### Availability

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

##### Supported Platforms

Allinea OPT is available for almost every flavor 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.

##### Assessment Environment

- **IBM BlueGene/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. The software and hardware environment of the STFC BlueGene/P is almost identical to the PRACE prototype at FZ Juelich, however, being a smaller machine it is more conducive to rapid turnaround and testing of profiling software such as OPT.

- **Allinea OPT v1.4.3**

Recent updates have added support for IBM BlueGene/P, SGI Altix and IBM Cell BE platforms.

- **Large-scale Application Software: Senga2b**

The Senga2 application developed at the University of Cambridge uses Direct Numerical Simulation (DNS) with desired levels of chemistry in order to model combustion processes. The original plan was to use Code\_Saturne for these

experiments. After consultation with the developers of Allinea OPT, it was expected that the complexity of Code\_Saturne (500k lines of Fortran, C and Python) along with the magnitude of available datasets would be too burdensome to demonstrate useful scaling features of OPT. For these reasons, we have used Senga2, which is also a parallel CFD code used for large-scale real-life calculations, but with more manageable datasets and a less complex code structure than Code\_Saturne (around 20k lines of Fortran only).

### **Description of the tool**

The Allinea Optimization and Profiling Tool (OPT) is a development tool for analyzing and 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. All stages of the analysis described in this report were undertaken on the Blue Gene/P.

### **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.

### **Description of the Large-scale Application Software: Senga2**

The Senga2 code has been developed at The University of Cambridge and has been designed to facilitate combustion DNS with any desired level of chemistry, from single-step Arrhenius mechanisms through all classes of reduced reaction mechanisms up to fully detailed reaction mechanisms. The Navier-Stokes momentum equations are solved in fully compressible form together with the continuity equation and a conservation equation for the stagnation internal energy, as well as any required number of balance equations for species mass fraction. The numerical framework is based on a finite-difference approach for spatial discretization together with a Runge-Kutta algorithm for time-stepping. High-order explicit schemes are preferred due to their speed of execution and ease of parallel implementation, and a 10th order explicit scheme is standard for interior points. The code is fully parallel using domain decomposition over a cubic topology. The code is written in Fortran 77 with MPI library routines used for passing data between processors.

### **Computational Characteristics of the Application**

In common with other finite-difference structured grid-based codes, a typical Senga2-based computation consists of 3 stages:

1. Set up problem from parameters passed through input files.
2. Simulate combustion processes over a series of timesteps.
3. Gather data and output results to disk.

In real-life problems the calculation time is dominated by stage 2 above. For reasons of accuracy and numerical stability, the time-step size undertaken must be relatively small and therefore the number of time-steps required to model short periods of time can be very large (e.g. 10 000). The principal communications structure within Senga2 is a halo-exchange in three dimensions between adjacent computational sub-domains in order to update the periodic boundary conditions. This point-to-point data transfer takes place via corresponding MPI\_SENDs and MPI\_RECVs in the program. These are therefore the dominant MPI routines used in the code. The benchmarks investigated here use a global grid size of  $100^3$  for 64 processor runs and a global grid size of  $200^3$  for 256 and 1000 processor runs on the Blue Gene/P system. In order to reduce runtimes, the number of time-steps is limited to 10. This number is sufficient for profiling tests, as communication patterns for each time-step are identical. Therefore analyzing a small number of time-steps (even only 1) is representative of a full run involving many thousands of time-steps.

Senga2b has recently been run on several petascale architectures, including the Blue Gene/P, and demonstrates excellent scaling up to tens of thousands of processor cores.

In this analysis of OPT on the Blue Gene/P, both approaches have been used and effects on the code run times are shown in section 7.1.

## Conclusions

OPT provides a user-friendly environment for profiling and analysis of application codes on HPC systems. Its mode of usage is very straightforward and the package was installed quickly and easily on the Blue Gene/P. The tool facilitates a wide range of features, from a detailed analysis of individual messages between processes in the timeline view to summary views from across all processes, from which users can identify load-imbalances at a glance. The ability to switch easily between multiple runs in the database and view profiles from different jobs side-by-side is particularly useful when gauging communication overheads during application scaling tests. OPT also has the facility to incorporate hardware event counter information, such as PAPI, but this feature was not investigated during this assessment.

Due to scalability issues, the current version of OPT would probably be unsuitable for general profiling analysis for computing at the petascale. The timings reported in Table 12 - Table 14 in Annex 7.1, comparing run-times for Senga2 with and without OPT, show that the overheads associated with the collection of OPT trace events are greatly increasing when profiling large numbers of MPI processes. However, it should be noted that these performance overheads may not be necessarily prohibitive: users may need to profile a high-core count job only once in order to glean useful information about the communication structure and parallel code behavior. It should also be noted that during this analysis, OPT always completed its profiling on high core counts (no crashes were observed) and the OPT viewer was stable throughout its usage.

Responsive, interactive usage of the OPT viewer was reaching its limitation at the 256 process count on Blue Gene/P, even when invoking features that reduce profiling overheads (e.g. setting maxfuncs, stop/start logging) and run-time overheads associated with tracing jobs involving more than 100 processes were substantial. A trace file was generated for a Senga2 job involving 1000 processes. However unless the investigator has in mind a specific point of

interest (e.g. the behavior of a specific function) general profiling and analysis of a 1000 process job via the OPT viewer would be impractical. Allinea are aware of these scalability limitations and future development of OPT will attempt to improve performance in this area.

### **Suggestions for Future Improvements**

Whilst there exist several methods for restricting the number of MPI Events logged during runtime, there appear to be few ways of reducing the amount of information loaded and shown in the OPT viewer. For example, it would be convenient if users could load into the viewer only communication data from a subset of processes. Likewise, once profiling data is loaded into the viewer, a feature should be added to enable users to filter the data on view, for example, to show only communications between two named processes. Features such as these would both improve the reduce overheads on the OPT viewer when analyzing high core count runs and provide clearer profiling data for the user. The documentation for OPT is also rather incomplete and the author feels that detailed explanations of many of the features available are not provided.

### **Acknowledgements**

The author would like to thank the Allinea Support team for their prompt and informative in-depth assistance.

## **6.3 Cray Performance Analysis Framework**

The Cray Performance Analysis Framework consists of the Cray Performance Analysis Tools (CrayPat) and Cray Apprentice 2. CrayPat is a suite of programs that can be used to analyse the performance of parallel applications on Cray XT supercomputers. The suite includes a tool for instrumenting applications (`pat_build`), a run-time library for measurements, a text-based tool for analyzing the results (`pat_report`), and an online help utility (`pat_help`). The focus areas in developing the tools have been ease of use and scalability, CrayPat has successfully been used to analyse the performance of applications running at scale with tens of thousands of processes. Apprentice 2 is a tool with a graphical user interface that can be used to analyse and visualize performance data. The current version of the tools, 5.0, was released in September 2009.

### **Features**

The basic usage of CrayPat does not require one to do any source code modifications. It turns out to be enough to load the appropriate CrayPat module, re-link the application and thereafter instrument the application binary using `pat_build`. CrayPat produces a standalone instrumented program that can be run independently of the original binary and object files. It is also able to instrument applications compiled with optimization flags which is important in order to get realistic performance measurements.

CrayPat is able to provide performance information on a wide range of metrics defined via so called tracegroups. They cover MPI tracing, OpenMP, Co-Array Fortran and other PGAS languages, numerical libraries such as BLAS and FFTW, memory allocated from the heap and finally metrics concerning I/O. Naturally the tool can also provide profiles for user level functions. CrayPat uses PAPI to measure HW counter information. The latest version of CrayPat supports multiplexing HW counters, allowing one to measure any combination of HW counters.

CrayPat supports two kinds of performance analysis experiments: tracing (synchronous) experiments and sampling (asynchronous) experiments. The tracing experiments instrument

the program to trigger a measurement each time the event occurs. Sampling experiments capture performance data at specified time intervals, or when some counter overflows.

The default mode of operation for CrayPat is to only collect a profile of the application performance. This is a summation of the events over time and does not provide information about the sequence of events. Simply by setting an environment variable to 0 (PAT\_RT\_SUMMARY), one can produce a trace of the sequence of events over time. This tends to significantly increase the amount of captured data, which makes it in many cases unfeasible to trace the complete execution of a large-scale application. One can use the CrayPat API and insert commands in the application source code to only measure shorter segments of the program execution. This allows producing trace files even for very large scale applications.

For report on the actual experiments with CrayPat and Apprentice2, we refer to section 7.2.

### Feedback and experiences

In general the CrayPat tools have proved to be stable, reliable and scalable. There is still room for improvement though.

The views in Apprentice 2 showing information on a per process basis, e.g. load balance, cannot be zoomed out very far. This means that one need to do a lot of scrolling to see the values for all processes. It would be very useful to be able to zoom out so that one process would be represented by only a few pixels, or even just one. One could even extend this by using grouping to enable one to get an overview of tens of thousands of processes.

As the time-sequence experiments tend to produce a lot of data, one often needs to use the CrayPat API to only capture short segments. It would be useful to have the option to only capture short time segments without having to do changes to the source code of an application.

## 6.4 Dewiz

Dewiz is a tool-set developed by partner GUP over the last years. Its main purpose is to provide means of debugging and performance optimisation of large scale applications by analysing their communication patterns.

Many bugs causing undeterministic behavior can be found more easily by looking at traces of communication events between the involved MPI processes. However, with the rise of petascale machines running large simulations, the human eye might be overwhelmed with data because of the huge number of events in large traces and the impossibility of displaying them in a meaningful way. This is where the pattern matching component becomes active. It includes several pattern matching functionalities to hint the user where expected stencil-like communication behavior (i.e., constantly repeating communication patterns from e.g. parallel PDE solvers) exhibits irregular communication structures, imposing a possible bug or unwanted behavior. There has been recent research showing that this approach is applicable and yields good results also for large message traces.

A shortcoming of the tool at its present development stage is that its tracing library can only support a very basic set of communication event types that can be traced correctly. It shows that most of the production codes part of the PABS use a much larger set of communication primitives (i.e. collective communication operations) that are omitted in the generated traces, leading to unreliable/unuseful information and therefore unusability of the tool. There exist a few external tools that are able to trace the complete set of communication primitives;

however, those are unluckily still missing certain information in their traces that are needed for the pattern matcher to work on current real life codes. Once this situation has been improved in the future, the pattern matching component will be still of greater use for the HPC community.

Another tool of the Dewiz set tackles possible performance issues caused by programmers using wrong – in terms of communication efficiency – communication primitives (i.e. manually implemented communication using one to one operations instead of original, often more efficient, collective operations). The tool includes a transformation framework which transparently changes the implementation of collective communication (native or point-to-point based collectives / in a blocking or non-blocking fashion) depending on which choice provides the best performance in a particular application context. Experiments on the FFTW and GROMACS code base, running on a commodity based Opteron cluster system using an Infiniband interconnect have shown an improvement over the original codes. Tests on the Jugene and Louhi Prototypes however have shown, that their system implementations of collective operation are already performing best compared to other implementations. The system vendors obviously put lots of efforts in designing their communication networks and that highly tuned libraries like the tested FFTW make use of them. The evaluation of this component of the Dewiz set of tools has shown that they can most probably not improve the use of communication in parallel codes on the current prototype systems.

### Feedback and Conclusions

The most important drawback of the Dewiz tool is currently the lack of collective communication support. This makes Dewiz not really usable in practice for the PABS. Improvement is required in this area. On the other hand, an interesting feature is the analysis of communication primitives, which would be even more useful when collective communication could be traced as well.

## 6.5 IBM HPCT: High Performance Computing Toolkit

### Introduction

The IBM High Performance Computing Toolkit (HPCT) is a collection of tools that can be used to analyse the performance of both parallel and serial applications, written in C or Fortran, on the AIX or Linux operating systems on IBM Power Systems Servers. The tools should allow the user to do the following measurements of their application performance:

- access hardware performance counters, e.g. for analyzing cache utilization and floating point performance,
- profiling and tracing of MPI applications,
- profiling OpenMP applications,
- profiling I/O patterns.

Although IBM does not formally support hardware performance counter tools on the Linux platform, since it depends on the use of an unofficial patch (perfctr) of the kernel, the hardware counters do work on the Huygens system. Recently, the IBM Parallel Environment on the Huygens system has been upgraded from version 4.3 to version 5.1. IBM PE 5.1 includes a productized version of the HPCT, whereas HPCT previously was unsupported software from IBM alphaWorks. This upgraded edition includes e.g. the peekperf GUI, which

is the user interface to the toolkit from where it should be possible to instrument, run and analyse the performance measurements for your application.

We selected the scientific application NEMO to test the HPCT in a production environment. NEMO is an ocean model that is used by hundreds of scientists all over the world. It is developed in France and it is written in a modular fashion using Fortran 90. It consists of 120k lines of code and uses the MPI library for parallelization.

### **Instrumenting the application**

To be able to instrument the application, it needs to be compiled with IBM compilers with the extra flags '-g -emit-stub-syms'. Fortran programs furthermore require the PSIGMA\_MAIN environment variable to be set to the name of the main program. To enable the use of the hardware performance counters, the application needs to be linked with the perfctr library, which is not documented in the HPCT Guide.

The compiled application can be instrumented through the peekperf GUI or the command-line utility hpctInst. The peekperf utility is quite slow in opening binaries and sources for instrumenting or analyzing, which can take up to minutes for the NEMO executable.

For a report on the actual experiments with IBM HPCT, we refer to section 7.4.

### **Feedback to Vendor**

We have given the following feedback to the vendor:

- The requirement to link the application with -lperfctr when using hardware counters needs to be documented.
- When the instrumenter fails, it suggests to set the environment variable PSIGMA\_MAIN to the name of the Fortran program. However, another reason for the failure is that the Fortran application is not compiled with '-g -emit-stub-syms'. It would help to add this suggestion when instrumentation fails.
- HPCT complains about 'missing trampoline' when using 'include mpif.h'. Sometimes it helps to use 'use mpi', sometimes not.
- MPI instrumentation changes the behavior of MPI.
- I/O analysis seems not implemented, although instrumentation changes the behavior of the application.

### **Conclusions**

The High Performance Computing Toolkit from IBM contains several tools. The first tool is the Xprof tool, which can be used to analyse cpu time profiles from a serial or parallel application. The outdated GUI does not give a significant advantage over a flat profile of all routines. Its only useful property that sets it apart from the free GNU tools available is the source line profiling. The PeekPerf GUI has a few interesting features which sets it apart from other performance tools, e.g. the flexible instrumentation of the code with hardware performance counters, MPI tracing and I/O profiling through a modern-looking GUI. Disappointingly, only the hardware performance counters work satisfactorily on the Huygens system, even though all different measurements should work on the Linux-on-Power platform. The only feature that does work is the measurement of hardware counters. However, the GUI prevents the user from finding the most important routines in terms of used cpu time. As a result, apart from the source line profiling, the High Performance Computing Toolkit cannot be recommended to potential users.

## 6.6 IPM: Integrated Performance Monitoring

### Introduction

As described in D6.3.1, WP6 agreed to use the Integrated Performance Monitoring (IPM) toolkit from NERSC [6]. By using IPM, PRACE users will have access to an open source, portable and scalable profiling tool. IPM reports MPI function timings, memory usage, and hardware counters data.

### Deployment Status

In order to provide a uniform performance analysis environment to the users across all PRACE sites, IPM has been ported to most of the PRACE prototypes. To date, IPM has been installed at the following PRACE sites: CSC, FZJ, SARA, CEA and HLRS. Instructions to install and test IPM on the PRACE prototypes are available for the interested sites. In the few cases where PAPI was not available on the prototypes, IPM was installed in a lightweight mode (without support for hardware counters). IPM has not yet been set up for BSC. As part of the NSF and DOE "preparing for petascale" program, IPM will continue to be supported on various high performance computing architectures.

Experiments with the IPM tool are reported in section 7.5 on the ECHAM5 code, when this was still investigated as part of the initial PABS.

### Considerations for Future Work

IPM should be used as a general portable tool for performance monitoring across a number of platforms. It is possible to use IPM on many different architectures but currently it does not support user function profiling, OpenMP profiling or PGAS languages. IPM developers are working on the next release of the tool (IPM2) which will support I/O, additional visualization tools and the potential to make performance extrapolations based on a collection of runs.

## 6.7 Scalasca

### Introduction

Scalasca is a software tool designed to analyse the performance of HPC applications on a wide range of contemporary HPC platforms [7]. During its design, a particular emphasis was placed on the usability on large scale systems such as the IBM BlueGene/P and the Cray XT systems, both of which are deployed as prototypes within the PRACE project. Scalasca also aims to facilitate performance analysis on small- and medium scale HPC platforms. The software is free but copyrighted by Forschungszentrum Jülich GmbH, Germany and 2003-2008 by University of Tennessee, United States of America. The present version 1.2 was released in July 2009 in the middle of this investigation. For this reason this investigation has been using version 1.1, beta releases of version 1.2 and the present version 1.2, depending on what was available on the system at the time. The developers state they have successfully tested Scalasca on the following platforms: IBM BlueGene/P, IBM SP & BladeCentre clusters, Cray XT4/5, SGI Altix, NEC SX-8, SiCortex systems and x86, x86-64, IA64 & SPARC clusters. This list includes a number of the PRACE prototypes.



## Platforms and applications used for this investigation

For this part of the investigation we used Scalasca on the following HPC architectures:

- **Huygens**, IBM Power 6 system with InfiniBand interconnect at SARA in the Netherlands. This prototype had originally Scalasca v1.1 installed and got upgraded to Scalasca v1.2 recently;
- **Juropa**, Cluster using Intel Xeon X5570 (Nehalem) quadcore processors with Infiniband interconnect at JSC in Germany. This prototype has Scalasca v1.2 installed;
- **HECToR**, Cray XT4 system located at the University of Edinburgh in the UK. HECToR has a beta release of Scalasca v1.2 installed. This system is closely related to the PRACE prototype Louhi and findings are expected to apply to Louhi, if it had a public installation of Scalasca.

We used two of the PRACE application code to investigate the usability of Scalasca:

- **HELIUM**: The application HELIUM uses time-dependent solutions of the full-dimensional Schrödinger equation to simulate the behavior of helium atoms [8]. The source code was developed by Queen's University Belfast and has access restrictions. The HELIUM source code is written in Fortran 90 and uses MPI only for the parallelism. All source code is in one file with 14569 lines;
- **NAMD**: The application NAMD is a widely used molecular dynamics application designed to simulate bio-molecular systems on a wide variety of compute platforms [9]. NAMD is developed by the “Theoretical and Computational Biophysics Group” at the University of Illinois at Urbana Champaign. In the design of NAMD particular emphasis has been placed on scalability when utilizing a large number of processors. The application source is written in C++ using Charm++ parallel objects for the data exchange between the compute tasks [10].

## Scalasca Details

Scalasca presently allows analyzing codes written in C, C++ and Fortran using MPI or OpenMP for the communication. With some restrictions hybrid codes using MPI and OpenMP simultaneously may also be analysed.

Using Scalasca is typically a four staged process:

1. Adding the Scalasca directories to your path, on many systems this is facilitated by loading a module
2. Instrumenting the object files and executables, which is done by prepending “scalasca –instrument” to the compiler and linker call
3. Executing the instrumented code with “scalasca -analyse”
4. Examination of the results with “scalasca -examine”

The results of these case studies are reported in section 7.6.

## Scalasca summary

The case studies show how Scalasca has been used in PRACE to investigate the performance of the HELIUM and NAMD code on a range of architectures relevant to PRACE. This shows a particular strength of the tool, which is not closely linked to a hardware vendor. The tool is available on a wide range on hardware architectures. Experience with the tool gained on one

architecture can be transferred to a different architecture. As shown for the case of HELIUM, the tool is typically easy to use and results can be obtained quickly.

With NAMD the tool presently struggles to instrument the executable. This could be related to NAMD's complex build procedure. In this case only a subset of Scalasca's functionality could be obtained. However it should be noted that one of the authors has outside the PRACE project experienced the opposite situation, where the vendor supplied tools failed and Scalasca could be deployed to rescue the project. Having several tools of similar functionality installed provides a useful fall back option in case one of the tools fails.

In our experience Scalasca works well on large scale experiments of this investigation, which utilized hundreds or even thousands of tasks. The efficient use of colors guides the analyst quickly to potential trouble spots.

Concerning future developments of Scalasca we would like to see the following improvements:

- In the MPI profile information about all MPI calls consuming considerable amount of time is needed. In particular less common calls can be poorly optimized in many MPI libraries.
- It would be useful for the analysis of applications using an iterative algorithm, if one could easily visualize the difference between the results from two executions of different length. This difference is typically associated with proper "working" iterations and would disregard overheads due to program start and finalization. Such overheads can be substantial in a typical measurement and benchmarking run, but are most of the time negligible for a proper production run.

We have detailed the proposed improvements to the code authors already.

## Acknowledgement

We would like to thank Brian Wylie from Forschungszentrum Jülich for his extensive support during this investigation.

## 6.8 Vampir VNG

Apart from the tested and supported platforms Vampir-Trace 5.7 was compiled on Huygens (SARA) and on Juropa (FZJ). The building process on both platforms is relatively simple. One has to take special care of the flags to be used for a proper compilation, especially on Huygens. On this system the mpi/poe libraries had to be specified very detailed:

```
CC=xlc_r CFLAGS=-O2 -g -q64 CXX=xlc_r CXXFLAGS=-O2 -g -q64 F77=xlf_r
FFLAGS=-O2 -g -q64 FC=xlf90_r FCFLAGS=-O2 -g -q64 MPICC=mpicc
MPIF77=mpfort --disable-shared
--with-mpi-dir=/opt/ibmhpc/ppe.poe/
--with-mpi-lib-dir=/opt/ibmhpc/ppe.poe/lib/libmpi64
--with-mpi-lib=-lmpi_ibm -lpoe
--with-papi-dir=/sara/sw/papi/3.6.2/lib64
--with-papi-lib=-lpapi -lperfctr
```

This tool provides an interface to the PAPI performance library. Therefore, on Huygens, where this library is available, we could test this feature.

On Juropa it was necessary to pass the following flags to configure:

```
CC=icc CXX=icpc F77=ifort FC=ifort --enable-mpi --with-mpich2
```

Gadget was used to test this tool on both systems. On Huygens trace generation and all advanced-user features worked as expected. Since the trace files can contain several GB of information, filtering and the selection of parts of the code to be traced are rather important.

On Juropa, some of this functionality was working partially. For example: some advanced features like manual instrumentation and user counters worked as expected. However, intermediate flushing and buffer memory control were using the default setting only. The automatic unification, set by default, works only if it is explicitly set through the corresponding environment variable.

Although the tracing library is distributed under the BSD license, the visualization tool is not distributed under the GPL or BSD license, but it is a commercial product instead. Consequently, the postprocessing had to be done on HLRB2 at LRZ. As shown in Figure 3, the traces were generated from a run on Juropa with 512 MPI tasks. The trace files were 1.1 GB in size. The client server layout was used to analyse this information. The server used 8 MPI tasks for speed because of the size of the uncompressed files.

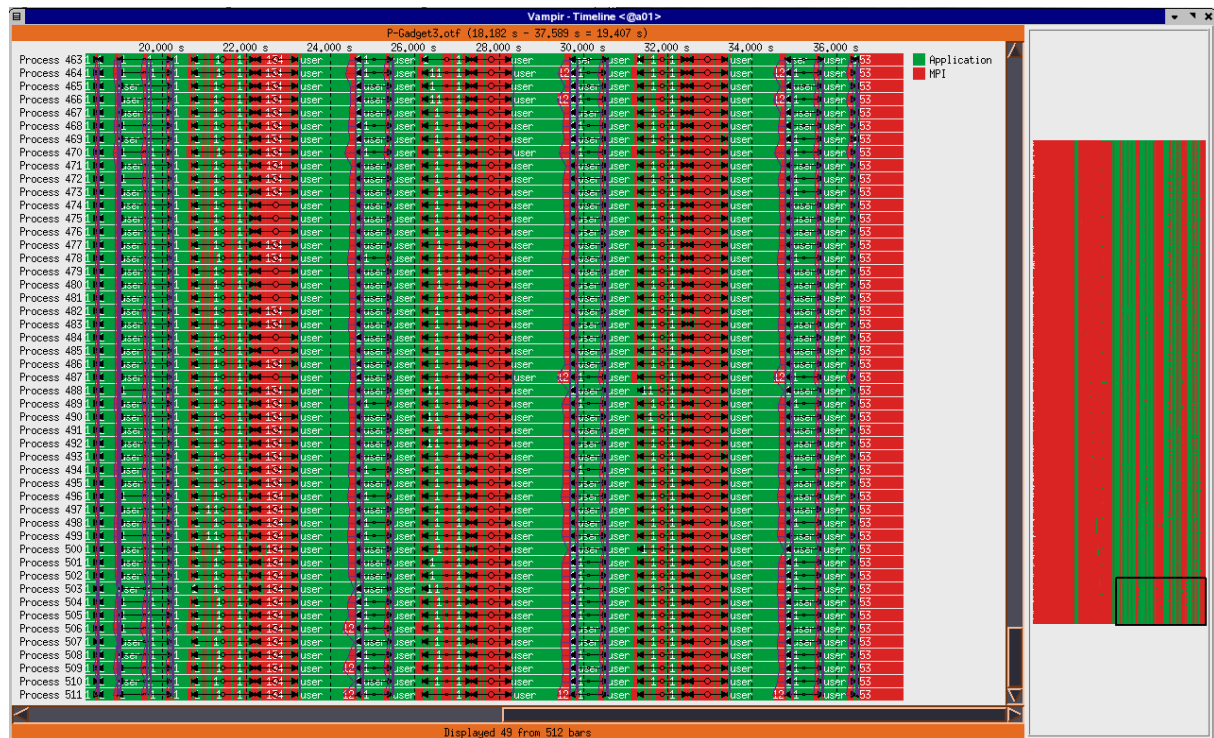


Figure 3: Vampir Total time line view, traces on Juropa, visualization on HLRB2.

## Conclusions

Based on the preliminary tests with Vampir, our first conclusion is that the tool is in development for Petascale systems, given the working features on IBM p575 (Huygens) and the only partially working features on IBM BlueGene (Juropa).

## 6.9 VPA: Visual Performance Analyzer

### Overview

VPA has been developed and is maintained by IBM alphaWorks.

Visual Performance Analyzer (VPA) is an Eclipse-based performance visualization toolkit. Each component of the package allows the user to read generated profiling data-sets produced by the user code and analyse them in a number of different ways. Major components of the package are the following:

- ***Profile Analyzer***

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) on IBM platforms and the profile tool oprofile on Linux systems.

- ***Code Analyzer***

Code Analyzer examines executable files and displays detailed information about functions, basic blocks, and assembly instructions. It is built on top of the FDPR-Pro, another feedback-based post-link optimization tool from IBM. 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***

Pipeline Analyzer is a port of the Performance Simulator for Linux on POWER, another IBM 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***

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.

- ***Trace Analyzer***

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.

- ***Flow Control Analyzer***

Control Flow Analyzer is a tool that analyzes call trace data collected by tools such as 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 a set of tables displaying the calling tree.

We have conducted experiments with VPA using the SIESTA code on the Cell platform (SIESTA was part of the initial PABS). These results are reported in section 7.8.

### **VPA Conclusion**

Visual Performance Analyzer is a very complex performance analysis package. It consists of a number of tools which enable very in-depth profiling and performance analysis. It is targeted for the POWER architecture but has a number of Cell processor specific optimization scenarios implemented. We have found these Cell-targeted functionalities very useful; especially SPU co-processors support is a really unique feature of the VPA package.

VPA has been found to be a very useful tool with Cell processor single-system (processor) performance analysis and instructive with progressive code optimizations. Multiprocessor parallel performance analysis was beyond the scope of our work with VPA.

## **6.10 General Conclusions**

For most of the performance tools, as considered in the previous subsections, we have been able to feedback our findings to the software vendors or open source developers. It is not necessary to repeat all previous subsection conclusions / recommendations here.

In general, we need ways to limit the trace files generated by the PATs when scaling to high number of cores. There seems to be a consensus on the fact that developers want an easy mechanism to limit this without doing any source modifications (either time limited or number of function calls limited, and configurable either via environment variables or configuration files). The rationale for this wish lies in the iterative nature of many algorithms: doing 10-100 iterations often reveals enough information for doing a performance analysis. For the real fine tuning we can still rely on the available PAT APIs.

Given large trace files, we want mechanisms to easily filter the events in the trace file (e.g. select only the events of the first 100-1000 tasks.) Without this, a GUI or batch analysis can easily take hours to process all data.

It is good to have multiple similar tools. Looking at the previous results these tools can be very resource demanding or require (sometimes undocumented) hooks in the compilers and/or MPI libraries. If one tool fails it is good to have other tools available.

Finally, something that is not completely obvious from the previous subsections: many PABS codes have been analyzed using Scalasca which scales very well and gives good insight into bottlenecks.

## 7 Annex

### 7.1 Scalability Experiments with Alinea OPT

Andrew Sunderland, STFC Daresbury Laboratory, United Kingdom

OPT consists of three components (that can reside on separate machines):

1. A library that is linked to the application code and records performance data.
2. A database component (the profiling server) to store profiling data. This consists of a PostgreSQL Database.
3. A graphical user interface that interprets 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 extra features.

#### Profiling Senga2 on the Blue Gene/P

On the IBM Blue Gene/P platform the following additions to the normal compile/link line for Senga2 were required:

- The OPT header-file include directory (only required if using the OPT API)
- The OPT static library directory, OPT and OPT-support libraries
- The GNU C++ stdc++ library
- The dynamic-linking library
- The "Fortran Wrapper" library (necessary when compiling Fortran codes)

#### Linking Senga2 with OPT on the BlueGene/P

When MPI profiling with OPT users can choose between linking with shared or static MPI libraries. For the Blue Gene/P platform the Alinea support team recommends using static MPI libraries.

On Blue Gene/P the link command for is:

```
mpixlf77 -g -O3 ${SENGA2_SRC_LIST} -o senga2.exe
-L${MPICHLIB_PATH} -lfmpich.cnk
-L${OPT_ROOTDIR}/opt/lib/static -lopt-bgp -lstdc++ -ldl
```

When using calls to the OPT API from within the source code, the OPT header-file include directory needs to be specified.

```
mpixlf77 -g -O3 ${SENGA2_SRC_LIST} -o senga2.exe
```

```
-I${OPT_ROOTDIR}/opt/include -L${MPICHLIB_PATH} -lfmpich.cnk
-L${OPT_ROOTDIR}/opt/lib/static -lopt-bgp -lstdc++ -ldl
```

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.

E.g. for Senga2b, the resulting executable can be run interactively or from within LoadLeveler scripts using the following command:

```
mpirun -np <nprocs> -env OPTPATH=/home/ags64/opt1.4/opt
-env BG_STACKGUARDENABLE=0 ./senga2.exe
```

(The run-time environment variable BG\_STACKGUARDENABLE is required for proper operation of OPT profiling on Blue Gene systems.)

All runs undertaken use one MPI process per Blue Gene/P core.

### Managing OPT profiling data for large-scale runs

In common with other tracing and sampling tools it is usually advisable to limit the amount of information collected when tracing relatively long jobs or jobs using large numbers of processors. This is necessary in order to prevent both the tracing process overhead and resulting trace files becoming prohibitively large.

OPT has two main methods to enable users to limit tracing:

1. *Selective OPT Logging.* This uses the OPT Fortran API to turn on and turn off logging via calls to OPT\_Stop\_Logging and OPT\_Start\_Logging function calls.
2. *Limiting the maximum number of functions that can be traced.* This is achieved by setting a parameter maxfuncs in the OPT configuration file opt.conf.

Both methods are effective at reducing tracing overheads. For users familiar with the program to be analysed, *Selective OPT Logging* is probably preferable, as this allows more specific control of tracing. For example, when tracing Senga2 runs, OPT logging can be limited to just one time-step, which in this case provides trace events representative of the communication pattern of the complete run. Limiting the maximum number of functions via maxfuncs is more arbitrary in that this method simply halts tracing in the code whenever the limit is reached. The default setting of maxfuncs is 100 000. The Allinea support team reports that the logging of 3 million functions in total is a level approaching the operational limit of OPT.

The timing results shown in Table 12 through Table 14 below report the observed runtime overheads associated with using OPT for runs involving 64, 256 and 1000 processes on BlueGene/P. As described in an earlier section of the report, it should be noted that the runs using 256 processes and 1000 processes simulate a problem size with double the dimensions of the problem using 64 processes. It can be seen that the overhead per time-step increases markedly when the code is run with 1000 processes. Here a time-step undertaken with OPT profiling takes around 9 times longer than a time-step with no OPT profiling. However, in the author's experience, this level of run-time overhead is not exceptional when compared to other profiling tools on large processor counts. From the 'Total Job Times' shown it can be seen that OPT logging introduces very large delays at the end of the parallel job – ranging

from 441 seconds for 64 processes to over 3 hours for 1000 processes. This extra time spent at the end of the run, presumably gathering the events logged, is out of proportion to the increase in the number of MPI events reported as logged in the OPT viewer.

	<b>No OPT</b>	<b>With OPT maxfuncs=100000 (default)</b>	<b>With OPT maxfuncs=2000</b>
<b>Time per TS (max / min secs)</b>	1.35 / 1.37	1.58 / 1.55	1.68 / 1.37
<b>Total CPU Time (secs)</b>	15.76	20.48	20.74
<b>Total Job Time (secs)</b>	17.15	478.35	441.21
<b>Logged MPI Function Calls</b>	N/A	401784	128064

Table 12: Runtime details – 64 process jobs.

	<b>No OPT</b>	<b>With OPT Logging for 1 time-step only</b>
<b>Time per TS (max / min secs)</b>	2.66 / 2.65	3.76 / 2.69
<b>Total CPU Time (secs)</b>	30.2	42.80
<b>Total Job Time (secs)</b>	40.58	3012.76
<b>Logged MPI Function Calls</b>	N/A	256512

Table 13: Runtime details – 256 process jobs.

	<b>No OPT</b>	<b>With OPT Logging for 1 time-step only</b>
<b>Time per TS (max / min secs)</b>	0.87 / 0.86	7.06 / 0.86
<b>Total CPU Time (secs)</b>	20.21	614.96
<b>Total Job Time (secs)</b>	35.58	12687.56
<b>Logged MPI Function Calls</b>	N/A	615000

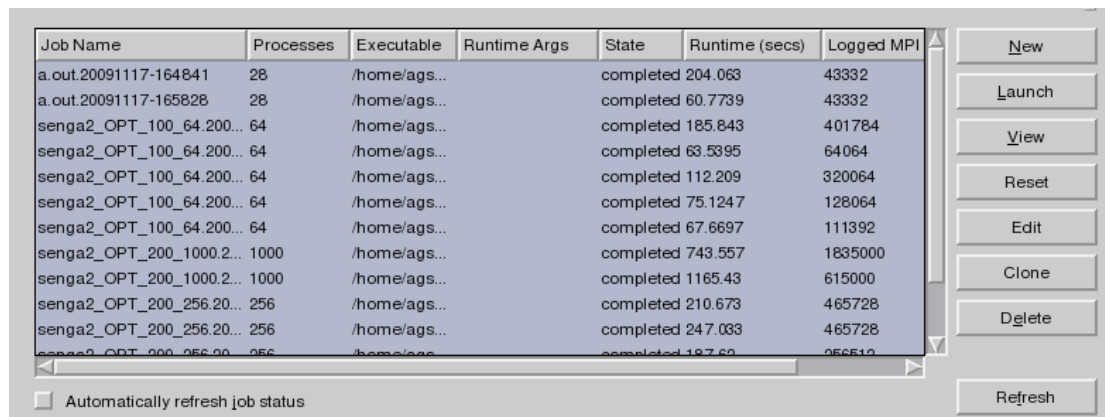


**Table 14: Runtime details – 1000 process jobs.**

Once the job has completed the MPI profiling data can be viewed in several different formats.

This section will discuss the different types of analysis that can be undertaken from the OPT viewer/GUI with an emphasis on ascertaining the suitability of OPT for high core counts – specifically the results from the 64 process and 256 process jobs on Blue Gene/P. Although tracing was completed for the 1000 process jobs, no attempt was made to view this information in the OPT viewer (it was concluded that this would be too demanding for the OPT viewer).

On starting the OPTGUI, the first window shown is the Session Manager screen. This database, shown in Figure 4, contains a list of all the user's jobs run with OPT tracing, along with associated details such as processors used, runtime, number of MPI events logged. This ability to easily switch between trace results obtained from different jobs is an extremely useful feature when analyzing communication patterns from scalability tests. Users can thus view tracings from different core counts side-by-side, organized conveniently within the viewer's tabbed environment.



Job Name	Processes	Executable	Runtime Args	State	Runtime (secs)	Logged MPI
a.out.20091117-164841	28	/home/ags...		completed	204.063	43332
a.out.20091117-165828	28	/home/ags...		completed	60.7739	43332
senga2_OPT_100_64.200...	64	/home/ags...		completed	185.843	401784
senga2_OPT_100_64.200...	64	/home/ags...		completed	63.5395	64064
senga2_OPT_100_64.200...	64	/home/ags...		completed	112.209	320064
senga2_OPT_100_64.200...	64	/home/ags...		completed	75.1247	128064
senga2_OPT_100_64.200...	64	/home/ags...		completed	67.6697	111392
senga2_OPT_200_1000.2...	1000	/home/ags...		completed	743.557	1835000
senga2_OPT_200_1000.2...	1000	/home/ags...		completed	1165.43	615000
senga2_OPT_200_256.20...	256	/home/ags...		completed	210.673	465728
senga2_OPT_200_256.20...	256	/home/ags...		completed	247.033	465728
senga2_OPT_200_256.20...	256	/home/ags...		completed	187.69	95512

☐ Automatically refresh job status

Buttons: New, Launch, View, Reset, Edit, Clone, Delete, Refresh

**Figure 4: Session Manager of OPT summarizing a range of different jobs.**

### 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 colored 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 behavior or load imbalances between processors.

The timeline is best when we consider only a small section of the actual run time of a program – as long as this section is representative of overall performance. Figure 5 shows a timeline for a 64 process run of Senga2 where `maxfuncs=2000` has been specified in the user's `opt.conf` configuration file. Figure 6 demonstrates a timeline for a 64 process run on Senga2 where the profiling has been restricted to three time-steps via calls in the source code to `OPT_START_LOGGING` and `OPT_STOP_LOGGING`. Non-traced sections of the timeline are represented by the light grey bars on either side of the traced time-steps in the centre of the view. As discussed in a previous section, this communication pattern is repeated

throughout the run and therefore tracing small numbers of time-steps provides an analysis representative of the complete run whilst reducing tracing overheads significantly.

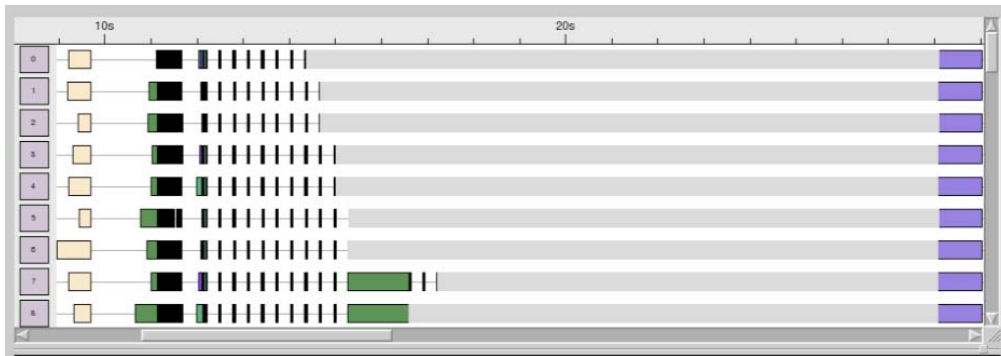


Figure 5: OPT Timeline view of a 64 process job with tracing restricted to 2000 functions.

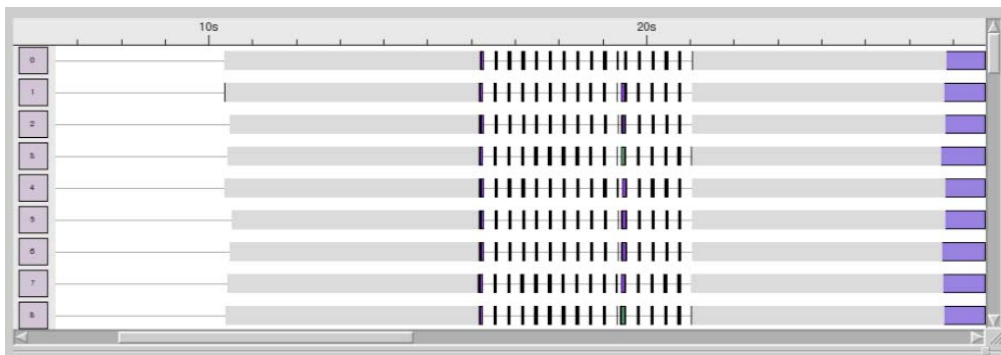


Figure 6: OPT Timeline view of a 64 process job with tracing restricted to 3 time-steps.

Figure 7 shows a zoomed timeline view of a halo-exchange between processors. The green and purple blocks represent time spent in MPI\_SENDS and associated MPI\_RECVs. The block lines represent the individual data transfers between processor cores. Details corresponding to an individual message can be shown by clicking on the associated line. The huge volume of messages in the larger process count run 256 processors meant that both communication patterns and individual messages became difficult to follow and some kind of filtering option in the viewer would have been useful (e.g. restricting messages shown to those above a certain size or between specific MPI ranks).

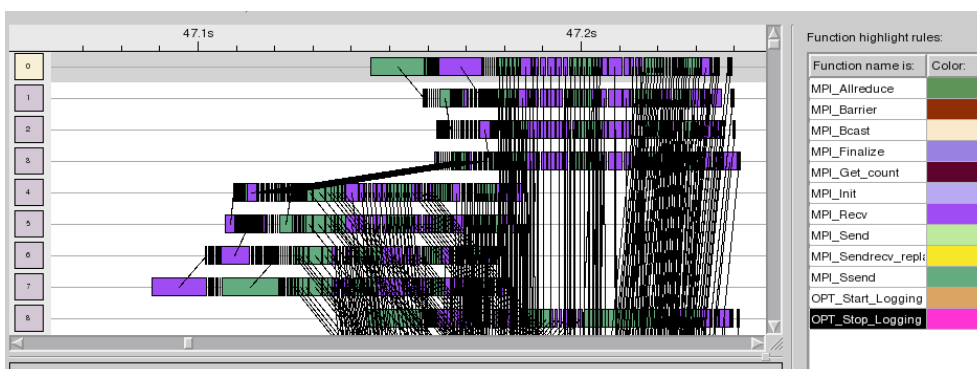


Figure 7: Zoomed timeline view for halo exchange communication pattern on 64 processes.

The timeline view was quite responsive to user instructions (e.g. zoom, toggle on/off communication lines) for 64 process jobs. Load times were somewhat longer when switching views for 256 process jobs, though the GUI remained quite useable.

### MPI Summary View

In many cases summarizing the MPI communications across all processes for the complete job is informative. Figure 8 shows that the message passing in a 256 process run of Senga2 is dominated by calls to MPI\_SSEND, MPI\_RECV and MPI\_GET\_COUNT, with a small amount of time spent in MPI\_ALLREDUCE. The Min/Max/Mean/Variance across the 256 processes is also shown.

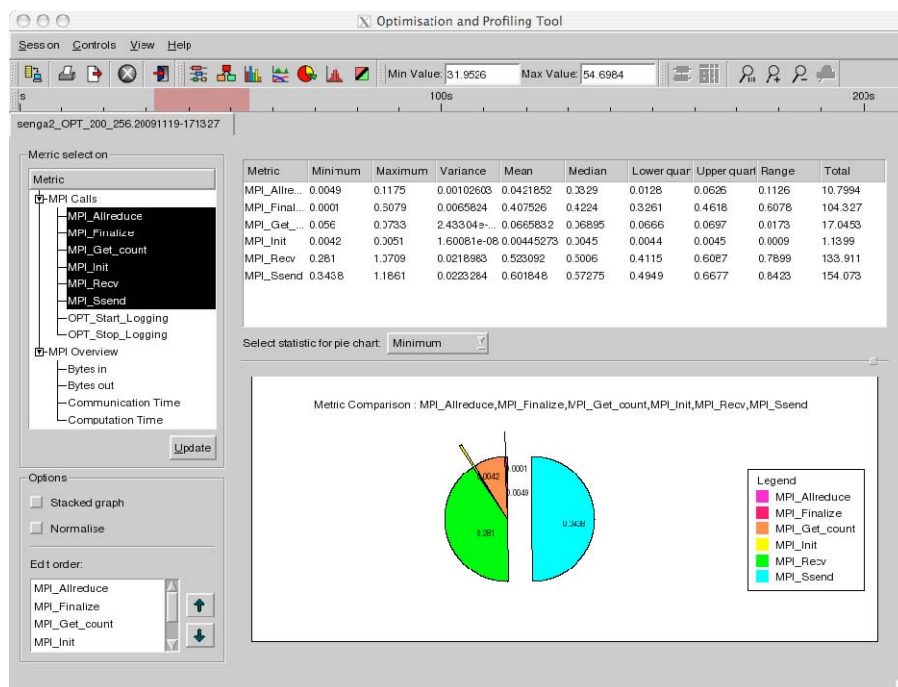


Figure 8: MPI Summary View for 256 process run of Senga2.

### Histogram View

This view arranges metric values from processes into buckets and gives a view of selected measurements by plotting a histogram. By viewing the data in this format users can easily identify load imbalances between selected processors. Figure 9 shows the large variance in time spent in MPI\_ALLREDUCE across 256 processes. Figure 10 displays a stacked histogram of time spent in all MPI calls across 256 processes. Both views are examples of how users can use OPT to easily spot communication load imbalances in their programs and are of particular use on runs involving large process counts.

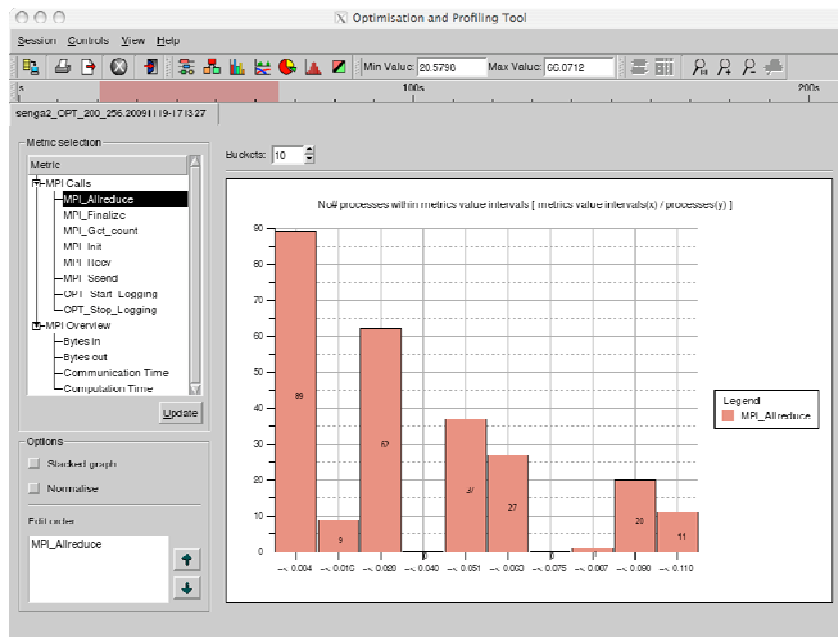


Figure 9: Variance of time spent in MPI\_ALLREDUCE for 256 process run of Senga2.

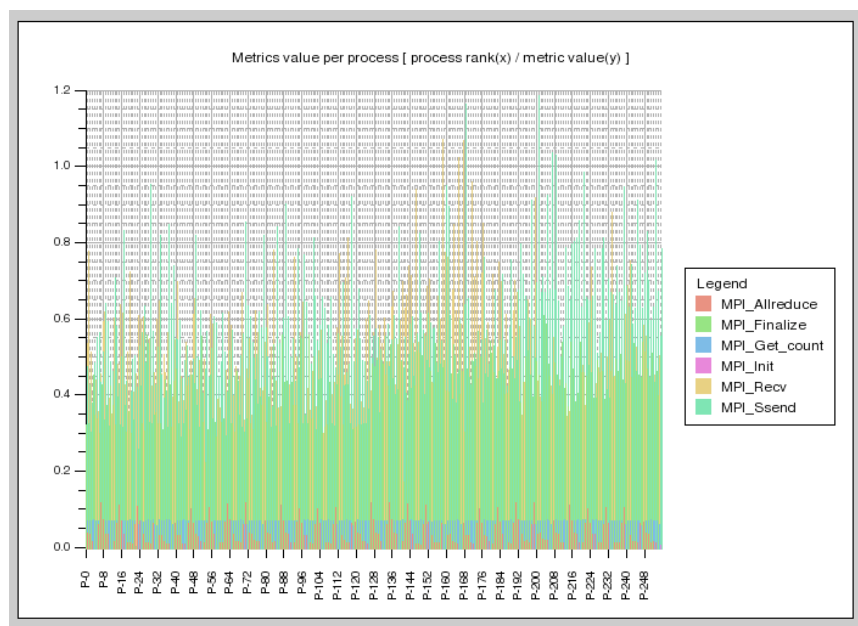


Figure 10: Stacked Histogram of Time Spent in all MPI Calls for 256 process run on Blue Gene/P.

### 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. The banded structure seen in Figure 11 and Figure 12 below is a classic communication pattern for finite-difference based algorithms. Metrics such as bytes transferred, number of MPI calls and time spent in MPI communications can be selected. This view is particularly useful for identifying communication patterns for jobs involving large processor counts. The 256 process message profile view was slow to load (around 10 minutes), but once loaded changing the view's parameters was quite responsive.

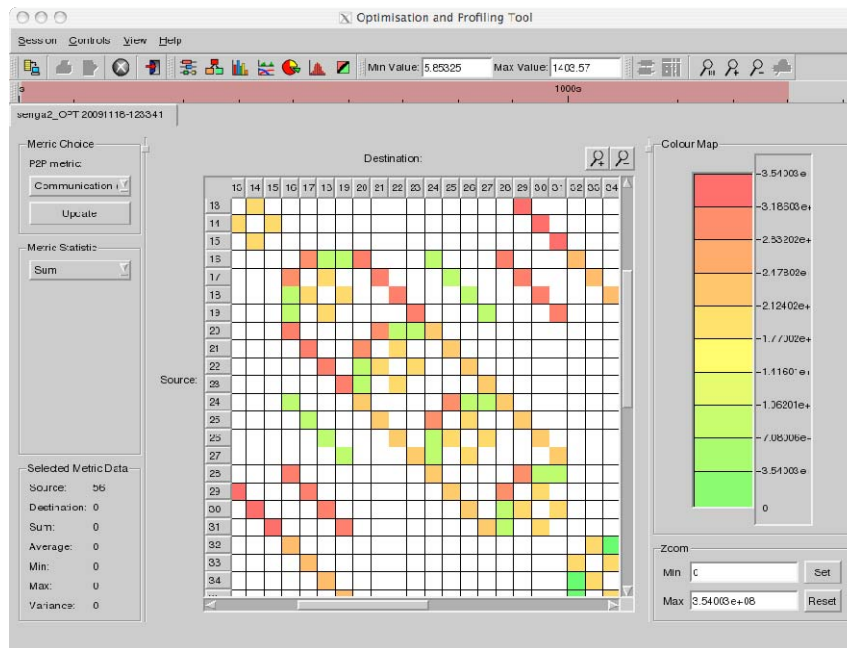


Figure 11: Message Profile View for 64 process run of Senga2.

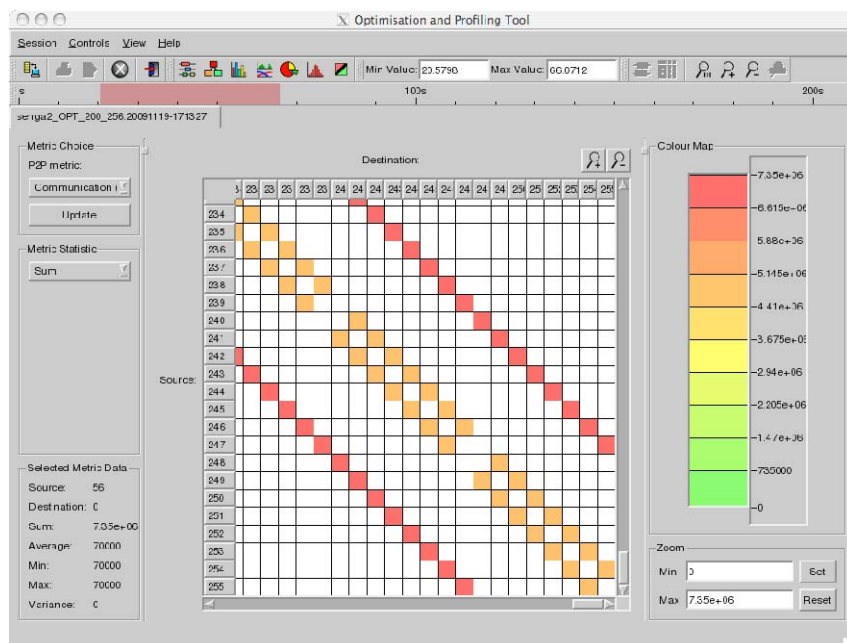


Figure 12: Message Profile View for 256 process run of Senga2.

## 7.2 Scalability Experiments with CrayPat and Apprentice2

Sebastian von Alfthan, CSC, Finland

To assess the functionality and scalability of CrayPat and Apprentice 2, we have analysed the performance of Gromacs running the PABS benchmark dataset. In this section we present step-by-step how the analysis was performed.

We first produced a profile of the Gromacs performance. The first step was to load the CrayPat module (xt-craypat) and to relink Gromacs.

```
module load xt-craypat
make mdrun
```

Here we used the recommended two phase method called Automatic Profiling Analysis (APA). In the first phase the binary was instrumented to collect information on the most time consuming functions:

```
pat_build -O apa mdrun
```

This command produced an instrumented binary (mdrun+apa) that was run using the test case. After the job had finished a performance file or a directory with multiple files had been created. A directory with multiple files is created when running large scale jobs. This is one key aspect which enables CrayPat to support performance measurement at scale. The performance file was thereafter analysed and a text report and an apa file was produced using the pat\_report command:

```
pat_report performance_file.xf
```

The apa file is a text file which defines how to instrument the binary in the second phase. The initial content of this file is automatically generated based on the information gathered in the first stage. The automatic values are most of the time good, but the user can also easily change them, e.g., add user level functions to be traced or change which hardware counters are captured. For Gromacs we noticed that the APA based scheme was not completely successful, we had to activate several important user level functions. In addition to user level functions we also traced MPI and collected some basic HW counter information. Based on the apa file we produced the final instrumented binary:

```
pat_build -O apafire.apa
```

An expert user can also discard the first stage in the APA scheme, and directly produce an instrumented binary. After all, the apa file mostly contains flags for pat\_build. After we run the final instrumented binary we obtained a new performance file. Running pat\_report on it produced a performance report and an ap2 file that can be visualized using Apprentice 2. The report contains information such as a profile (Table 15):

Time %	Time	Imb. Time	Imb. Time %	Calls	Group Function PE='HIDE'
100.0%	148.349789	--	--	177406.9	Total
76.5%	113.532364	--	--	73514.7	USER
54.9%	81.393673	2.780510	3.3%	10001.0	do_nonbonded
10.8%	15.986473	8.292538	34.3%	10001.0	do_force
4.5%	6.629125	3.868527	37.0%	1.0	do_md
2.5%	3.747337	8.112071	68.7%	10001.0	do_force_lowlevel
1.9%	2.881715	0.651005	18.5%	10003.0	csettle

14.0%	20.785209	--	--	6691.3	MPI_SYNC
11.8%	17.472643	13.257841	43.3%	4390.6	MPI_Bcast(sync)
1.9%	2.826379	11.005785	79.9%	1130.5	MPI_Allreduce(sync)
9.5%	14.032195	--	--	97197.9	MPI
9.2%	13.683468	8.110984	37.4%	87094.8	MPI_Sendrecv

Table 15: Profile report for Gromacs.

We also get HW counter information on the whole program, and on individual functions, as shown in Table 16:

USER / do_nonbonded						
-----						
Time%	54.9%					
Time	81.393673 secs					
Imb.Time	2.780510 secs					
Imb.Time%	3.3%					
Calls	122.9 /sec	10001.0	calls			
PAPI_L1_DCM	15.140M/sec	1232454976	misses			
PAPI_TLB_DM	0.019M/sec	1560986	misses			
PAPI_L1_DCA	1872.994M/sec	152465437424	refs			
PAPI_FP_OPS	6320.028M/sec	514462747250	ops			
User time (approx)	81.402 secs	219785329131	cycles	100.0%	Time	
Average Time per Call		0.008139	sec			
CrayPat Overhead : Time	0.0%					
HW FP Ops / User time	6320.028M/sec	514462747250	ops	58.5%	peak(DP)	
HW FP Ops / WCT	6320.028M/sec					
Computational intensity	2.34 ops/cycle	3.37	ops/ref			
MFLOPS (aggregate)	1617927.15M/sec					
TLB utilization	97672.50 refs/miss	190.767	avg uses			
D1 cache hit,miss ratios	99.2% hits	0.8%	misses			
D1 cache utilization (misses)	123.71 refs/miss	15.464	avg hits			
-----						

Table 16: HW counter information for Gromacs.

Load balance is also of outmost importance to improve performance; the report contains a table showing time spent in user functions, and the synchronization and normal MPI time (Table 17):

Time %	Time	MPI Msg Count	MPI Msg Bytes	Avg MPI Msg Size	Group PE[mmm]
100.0%	148.463816	95441.6	1608213464.0	16850.24	Total
76.5%	113.579615	--	--	--	USER
0.3%	119.268743	--	--	--	pe.85
0.3%	113.554316	--	--	--	pe.240
0.3%	111.465198	--	--	--	pe.176
14.0%	20.789510	--	--	--	MPI_SYNC
0.1%	31.694825	--	--	--	pe.45
0.1%	20.931146	--	--	--	pe.166
0.0%	8.090958	--	--	--	pe.201
9.5%	14.094668	95441.6	1608213464.0	16850.24	MPI
0.1%	22.218904	95122.0	1571528263.0	16521.19	pe.237
0.0%	13.053821	95244.0	1671435159.0	17548.98	pe.131
0.0%	10.544807	100104.0	1661084499.0	16593.59	pe.96

Table 17: MPI information for Gromacs.

Finally we also get statistics about message counts and sizes, both for the whole program and for individual MPI calls (Table 18):

```
Totals for program
-----
MPI Msg Bytes          1608213467.0
MPI Msg Count          95441.6 msgs
MsgSz <16B Count      11718.8 msgs
16B<= MsgSz <256B Count 14476.2 msgs
256B<= MsgSz <4KB Count 5952.7 msgs
4KB<= MsgSz <64KB Count 63287.3 msgs
64KB<= MsgSz <1MB Count 4.6 msgs
1MB<= MsgSz <16MB Count 2.0 msgs
=====
MPI_Sendrecv / dd_sendrecv_rvec / dd_move_x / do_force
-----
MPI Msg Bytes          781831261.0
MPI Msg Count          30003.0 msgs
4KB<= MsgSz <64KB Count 30000.7 msgs
64KB<= MsgSz <1MB Count 2.3 msgs
=====
```

**Table 18: Summary of profile information for Gromacs.**

The examples above only show the default reports; there are several options for additional reports and for changing the default ones.

Running Apprentice2 on the ap2 file enables one to visualize the performance data. We will here present a subset of the results one obtains. In Figure 13 the profile is shown as two pie-charts, showing the profile based both on calls and on time. Clicking on a function in the pie-chart shows its load balance, as shown in Figure 14.



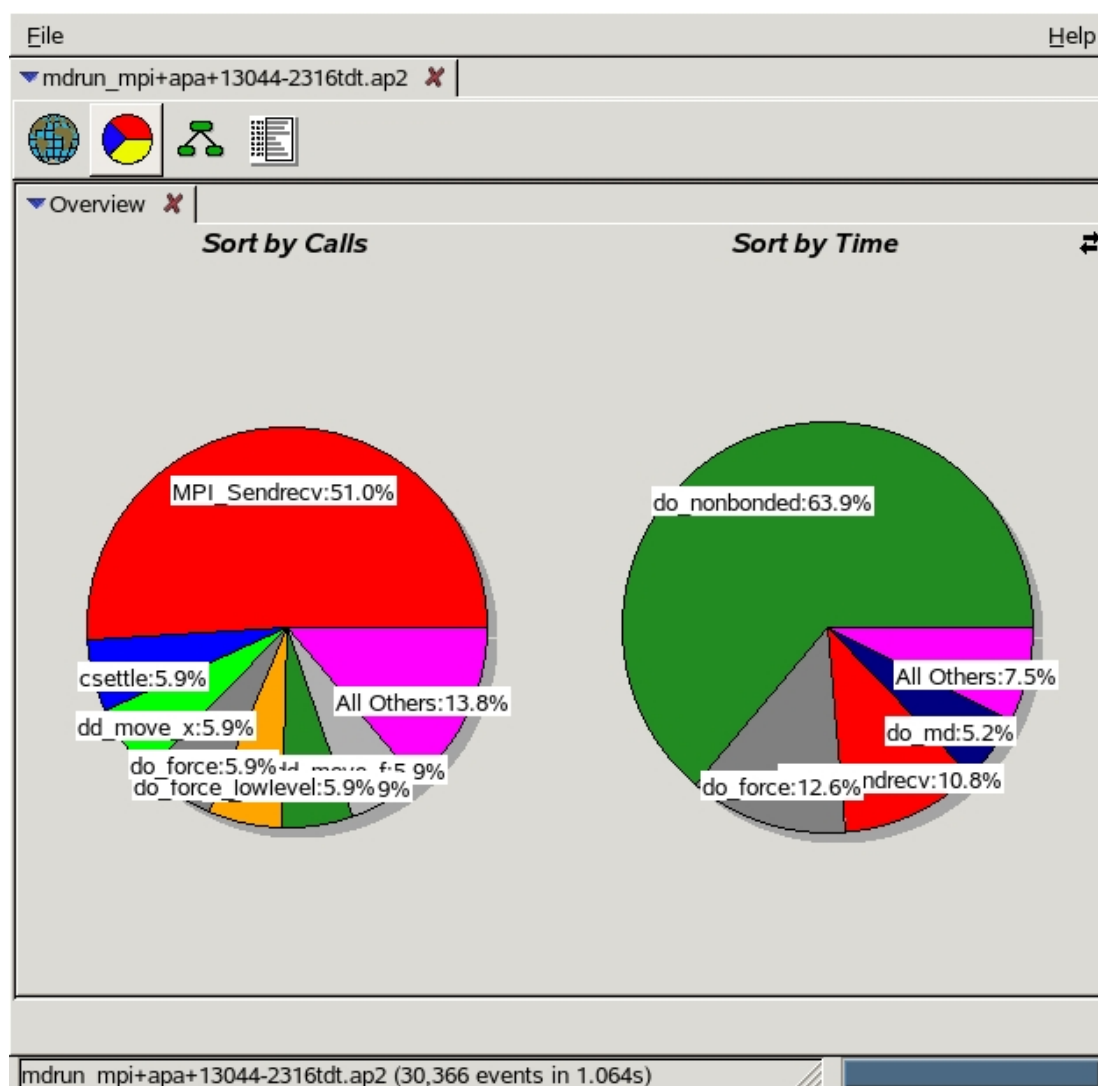


Figure 13: Profile of execution as shown by Apprentice2.

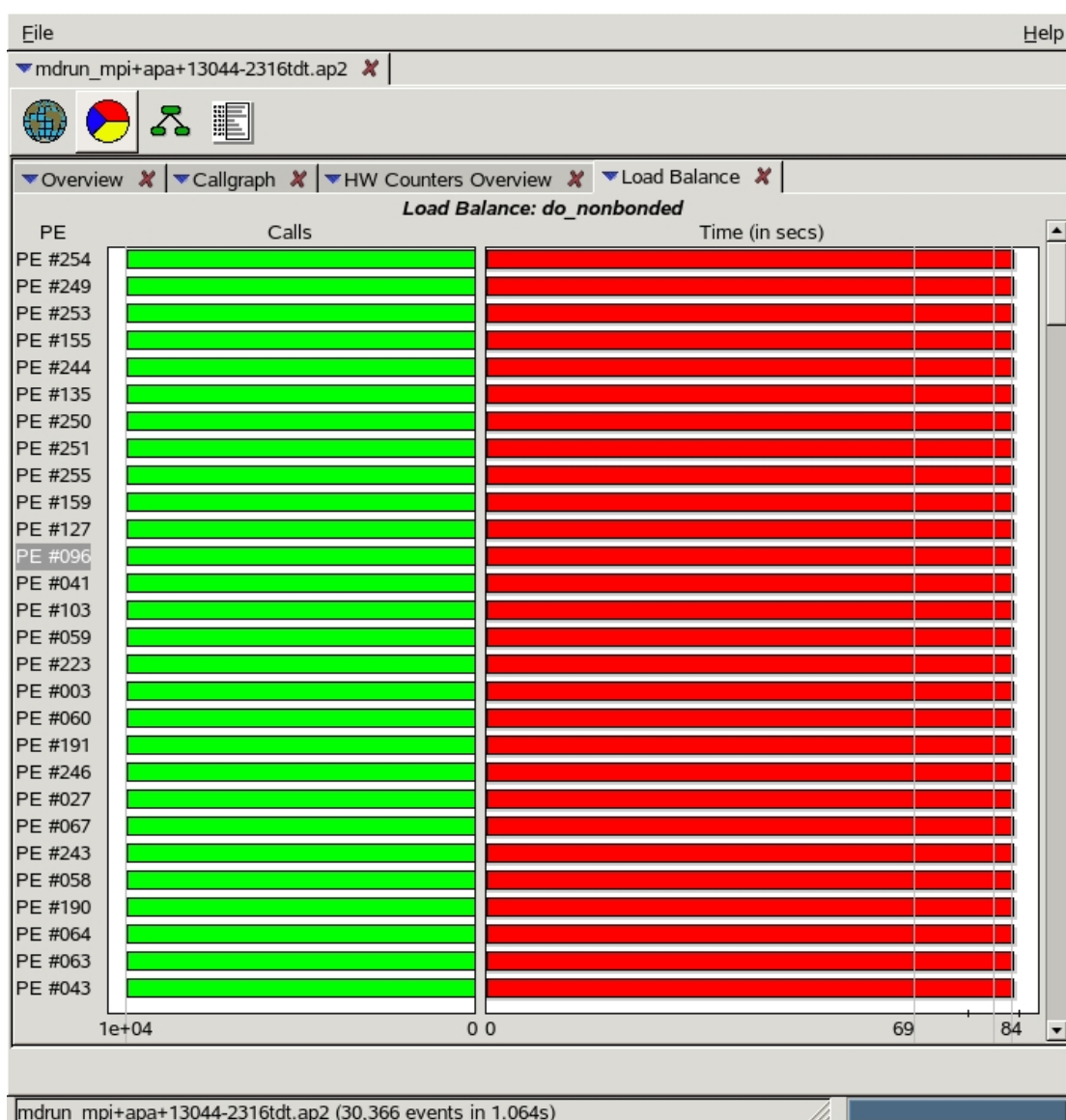


Figure 14: Load balance view in Apprentice 2.

Another very useful view is the call graph (Figure 15), showing a compact presentation about the load balance, how much time is spent in different routines and the relationship between callers and callees. This enables one to get an overview of the program, even if its structure is not familiar. The load balance information is represented by colors, the yellow bar in the background shows the maximum time, the purple bar shows the average time, and the cyan bar shows the minimum time spent in the function. The size of the boxes show the relative time spent in the function, the height represents the time spent in that particular function, while the width corresponds to the cumulative time spent in the children of the function.

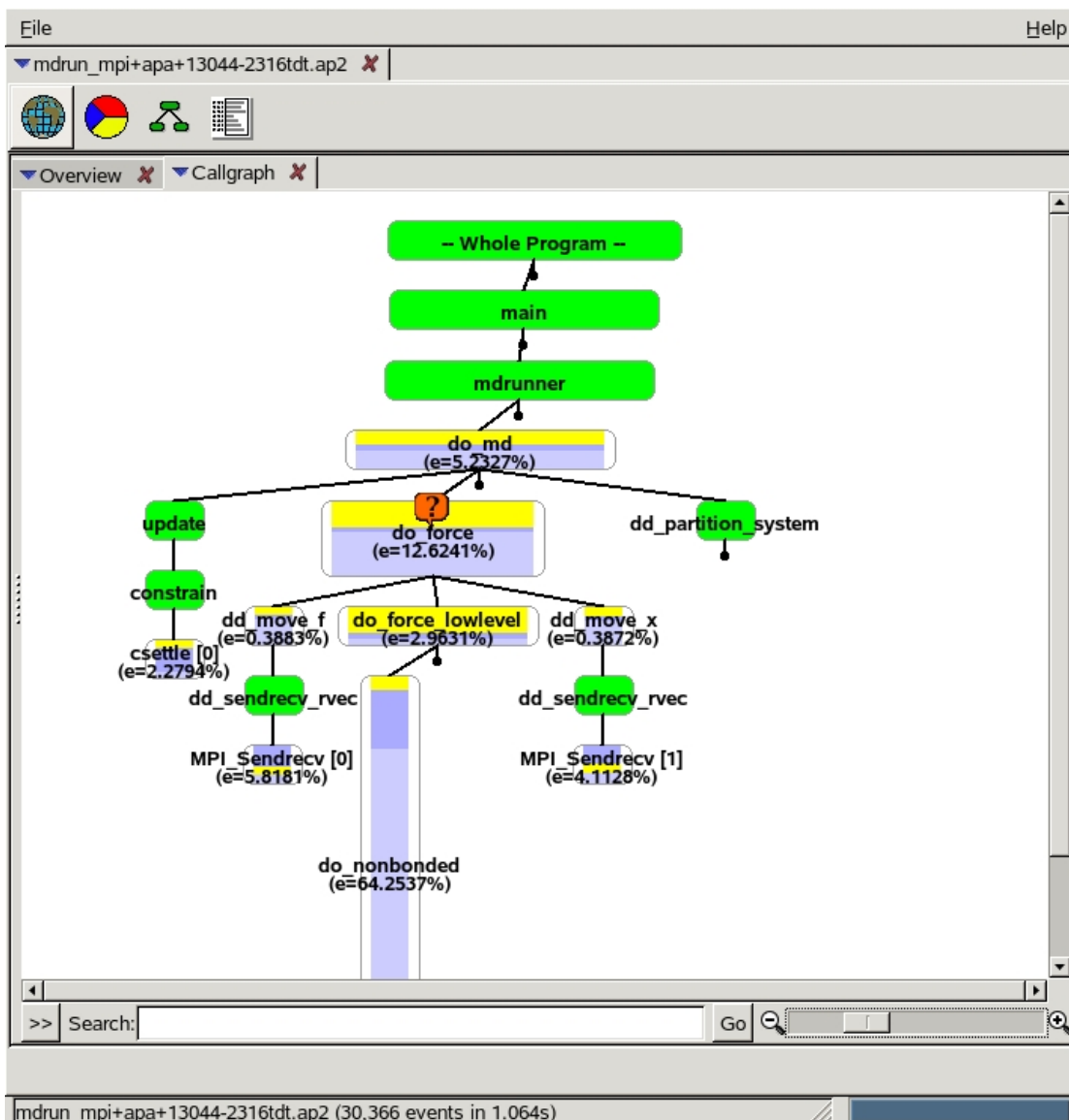


Figure 15: Callgraph view in Apprentice 2.

To enable a time sequence view in Apprentice 2 one simply needs to set an environment variable prior to running the instrumented application:

```
export PAT_RT_SUMMARY=0
```

This works well for short jobs, running on a small number of cores. If this is not the case, the amount of data captured is too large to be analysed with ease. The simple solution for an iterative algorithm, such as the Molecular Dynamics one in Gromacs, is to only turn on tracing for a few iterations. Each iteration is essentially similar to all the other ones, it's thus enough to just sample a few of them. This can be done using the CrayPat API by adding instructions in the source code. We modified Gromacs by turning off tracing right after MPI\_Init in the gmh\_setup function:

```
(void) MPI_Init(argc,&argv);
PAT_tracing_state(PAT_STATE_OFF);
```

In the main iteration loop we then turned on tracing for a few iterations as follows:

```
if(step==910){
    PAT_tracing_state(PAT_STATE_ON);
}
```

```

else if(step==920){
    PAT_tracing_state(PAT_STATE_OFF);
}

```

In Figure 16 a timeline is shown for Gromacs. In this case only MPI routines have been traced, therefore the time between MPI\_Sendrecv's is white. This kind of a view can be very useful in pinpointing the reason for bad load balance. In this case the view is quite simple, as the only communication is MPI\_Sendrecv between neighbors. When looking at another test case for Gromacs in PRACE report D6.4, the view proved extremely useful as it showed that the major bottleneck was MPI\_Alltoall.

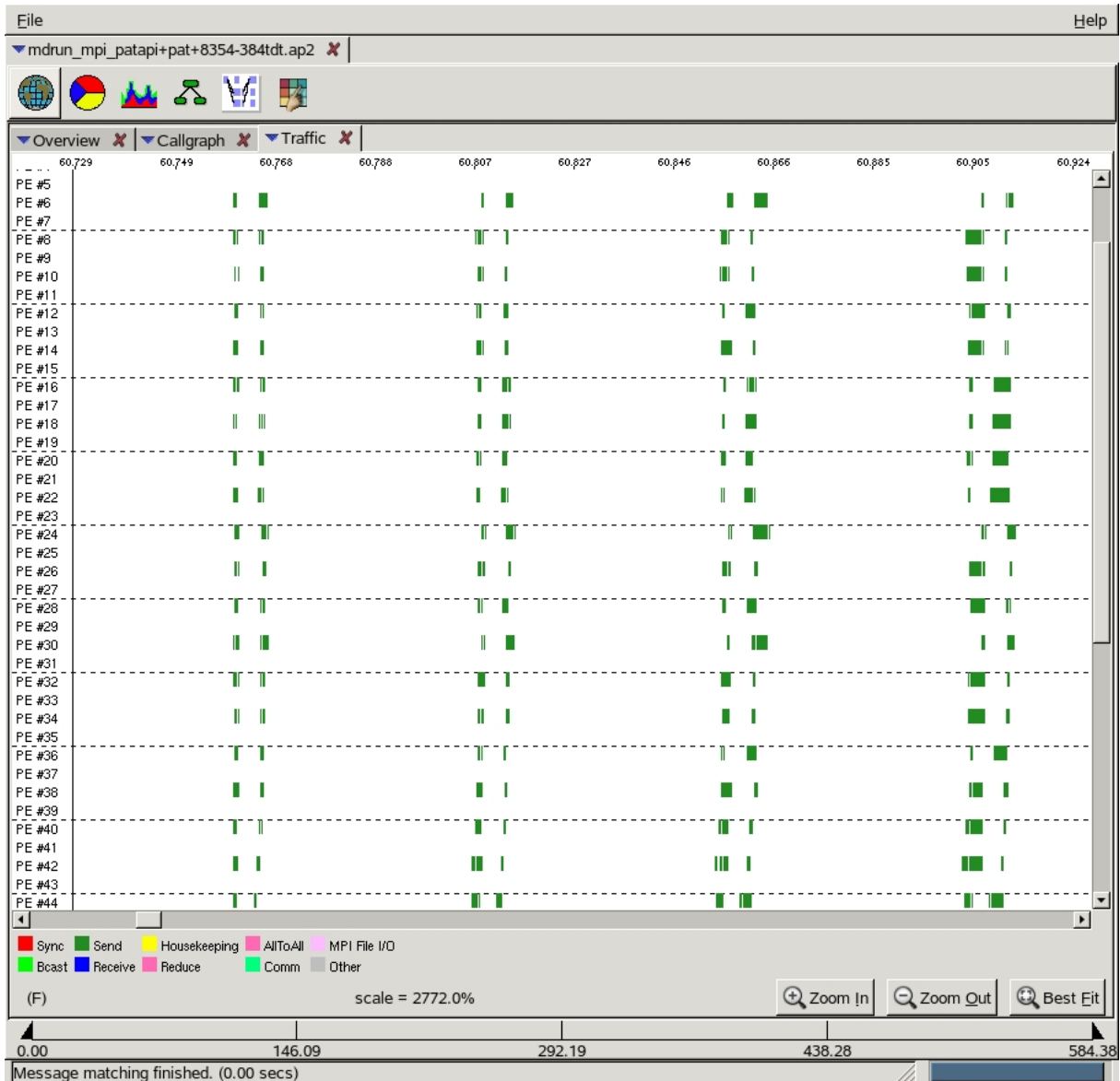


Figure 16: Timeline of Gromacs visualized using Apprentice 2.

## 7.3 Scalability Experiments with IBM HPCT

John Donners, SARA, the Netherlands

### Analyzing the CPU performance

The NEMO executable is instrumented for hardware performance measurements using `hpctInst`:

```
hpctInst -dhpm nemo
```

When running the instrumented binary, the following error message appears:

```
symbol lookup error: /opt/ibmhpc/ppe.hpct/lib64/libshpc.so:
undefined symbol: _vperfctr_open
```

This can be solved by linking the application with the `perfctr` library, but this is not described in the documentation. The instrumentation also creates a hidden file called `'.psigma.hpmhandle'`, which needs to reside in the same directory as the instrumented executable, which is neither documented.

The output is one file with the hardware counters for the first task only. The measurements are analysed using the `peekperf` utility, which shows two windows (cf. Figure 17): one window with a tree with the different source files and subroutines and another window with the source for each file. Once a subroutine is selected in the left window, the selected subroutine is marked in the right window. The tree view can be sorted alphabetically on subroutine and source file name or on the execution time, but it is difficult to find which routines are most cpu-intensive: the execution time includes all callees and the execution time for callees includes the calls from all its callers. The data can also be shown as a large spreadsheet and saved to a file for further analysis. However, this table neither contains the exclusive timing of subroutines. Further details (e.g. the flop rate and the instructions per cycle) are available and are clearly explained in the documentation. The `peekperf` utility allows specifying the group of hardware counters that should be used, but this option seems only available when the program is run from `peekperf`, which is usually not the case for massively parallel applications.

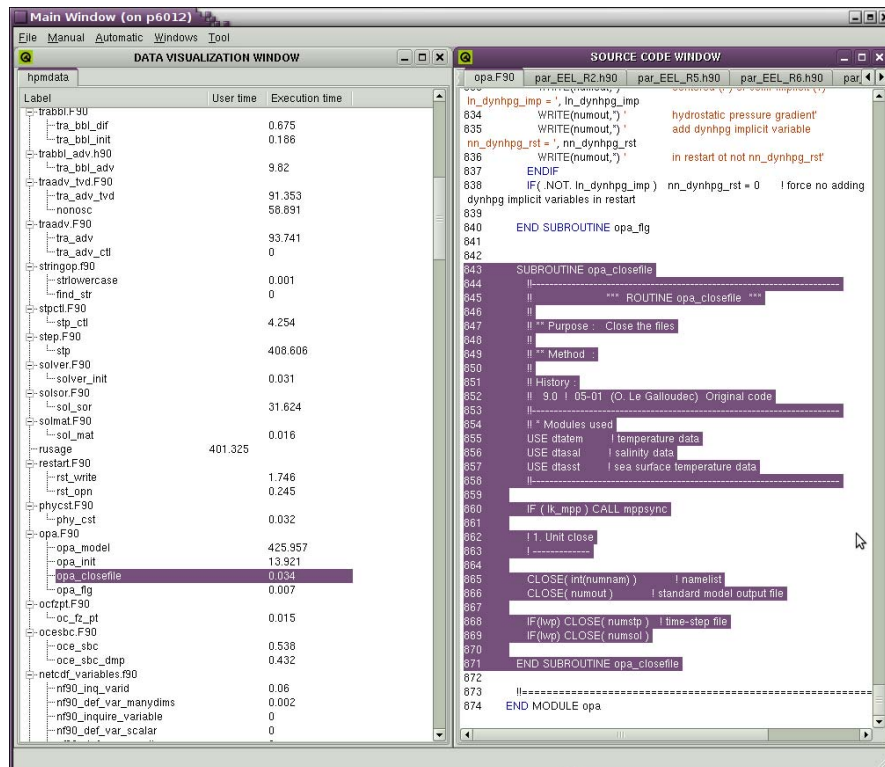


Figure 17: Main window of peekperf.

## Analyzing the MPI performance

The MPI calls in the NEMO application can be instrumented as follows:

```
hpctInst -dmpi nemo
```

which initially results in an error message for NEMO:

```
2750-202 ERROR: cannot find the trampoline for
'mpi_pack_external_size'.
Please add the -emit-stub-syms option while linking
(assume the user uses the XL compiler)!
2750-507 hpctInst: could not do instrumentation
```

For NEMO this problem could be solved by replacing 'include mpif.h' with 'use mpi' in the code. However, other Fortran 90 MPI applications give the same error message, even though the code does say 'use mpi'. It is unclear what is the cause of this ambiguous behavior. The instrumented NEMO binary needs 128 MPI tasks to run, so it is submitted as a batch job and runs from a scratch directory. Unfortunately, running the instrumented binary fails:

```
ERROR: 0032-154 Not a persistent request (0), MPI_Start, task 124
ERROR: 0032-154 Not a persistent request (0), MPI_Start, task 125
ERROR: 0032-154 Not a persistent request (0), MPI_Start, task 126
ERROR: 0032-154 Not a persistent request (0), MPI_Start, task 127
ERROR: 0031-250 task 65: Terminated
```

while the original binary does function properly. As a result, we have been unable to further test the MPI analysis with the HPCT.

## Analyzing the I/O performance

HPCT can also instrument I/O system calls (called 'MIO'), which can be very valuable for benchmarking high-performance computing applications, because I/O is often an important bottleneck for performance and scaling. The NEMO application uses primarily the NetCDF library for output, so the executable was linked with the static NetCDF libraries to enable instrumentation. NEMO was instrumented with the `hpctInst` command-line tool as follows:

```
hpctInst -dmio nemo_world
```

This creates an instrumented executable `nemo_world.inst`, as is the case for the instrumentation of MPI calls and hardware counters. The environment variable `MIO_FILES` was set in the batch job as suggested in the manual:

```
export MIO_FILES="*[trace/xml/events={./mio.evt}]"
```

Unfortunately, no I/O measurements are produced. Interestingly, the I/O instrumentation does change the runtime behavior of the application. After 66 timesteps with many I/O requests, the NetCDF library stops with an error:

```
iom_nf90_rp0123d: should have been an impossible case...
```

The tab window for the MIO instrumentation is empty when trying to instrument a very simple C program with file I/O using `peekperf`. There is no mention that this part of the HPCT is not supported on the Huygens platform (Power-on-Linux), so it is unclear why this doesn't work.

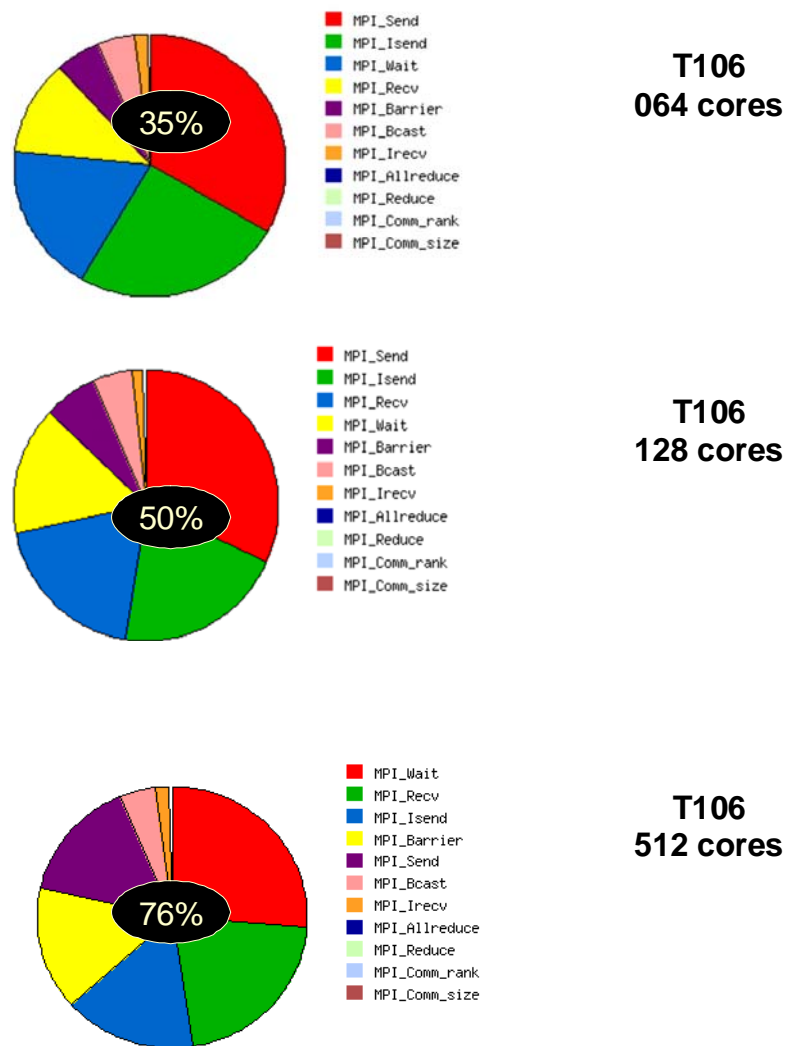
### **X Windows Performance Profiler**

Xprof is a GUI that can be used to graphically display a set of cpu profiles from each task of a parallel code. The call tree is graphically represented: each node is represented by a green box whose width represents the inclusive time (time to execute itself and all its callees) and its height represents the exclusive time (cpu time without callees). In 'averaged view' the boxes are scaled according to the standard deviation of all the profiles. Filter functions can be applied to select functions with certain names (e.g. to filter Fortran modules), the number of calls or is cpu time usage. This last filter is useful to declutter the view for large programs like NEMO. Profiles are available down to the source line level, which is an important advantage over the standard `gprof`. It displays the cpu usage for the source as 'ticks per line' (0.01 seconds), not as a percentage. The boxes are scaled linearly but with a lower limit, which makes insignificant routines appear much larger than they should.

## **7.4 Scalability Experiments with IPM**

Jean-Guillaume Piccinali, CSCS, Switzerland

We have chosen to investigate IPM on the CRAY XT5 using ECHAM5. ECHAM5 was run with a T106L31 configuration, simulating one month of model time. IPM shows a rather poor scalability of the code (Figure 18). For instance, on 64 cores 35% of the execution time is taken by MPI communication routines. For 128 and 512 cores, this is even more.



**Figure 18: Scalability details of IPM.**

As described in D6.3.1, ECHAM5 suffers from high frequency of small messages. Scalability results show (see chapter 3) that ECHAM5 remains under the 10 Tflop/s threshold. It turned out to be impossible to obtain scalability results for that code which pass this threshold. In conclusion, the current problem size of ECHAM5 will not scale to large numbers of cores. As a replacement, we used the LINPACK benchmark to evaluate the performance analysis tool. The issue of the time taken by the tools to generate sufficiently small traces with as much details as possible has been raised in WP6. The following table shows the overhead induced by using either CrayPat or IPM (Table 19):

Cores	No Tool	IPM	IPM Overhead	CrayPat	CrayPat Overhead
2916	1531.3	1459.7	-5%	-	-
3600	1292.2	1232.6	-5%	1348.4	4%
6084	855.3	868.1	2%	911.4	7%
10404	583.2	668.0	15%	672.6	15%
14400	477.0	614.4	29%	562.8	18%
15876	457.2	616.9	35%	547.8	20%



17424	440.3	642.3	46%	530.7	21%
19044	429.4	690.5	61%	531.9	24%
20736	418.4	710.0	70%	517.6	24%

**Table 19: Comparison of overhead introduced by CrayPat and IPM.**

It is interesting to observe that for a few thousands of cores, the run with usage of the IPM tool is faster than without. We have not investigated this further, although it is a rather strange effect. It is unlikely that such tools will scale without modification to hundreds of thousands of processes (although several thousands is not a problem). Based on these results, detailed feedback has been sent to the developers of IPM and the decision has been taken to modify some components of the tool.

IPM wasn't easy to install initially but input from the NERSC staff helped to overcome the issue.

With respect to file handling transparency, IPM can collect individual performance profiles into a database which synthesizes the performance reports via a web interface. Since profiles are stored centrally in an SQL database they provide a performance track record to developers and a means of workload characterization to HPC managers. Analysis can also be done on a local laptop.

## 7.5 Scalability Experiments with Scalasca

Xu Guo and Joachim Hein, EPCC, United Kingdom

### Using Scalasca for HELIUM profiling

The Scalasca toolset was used for profiling HELIUM performance on the SARA POWER6 system, Huygens, and the new Jülich cluster with Nehalem cores, Juropa. The profiling was used to help with the petascaling and optimization for PRACE WP6 task 6.4 and task 6.5. For the HELIUM profiling on Huygens and Juropa, a 1540-block test case was run on 1540 cores with Scalasca used.

Using Scalasca for HELIUM profiling is straightforward. The implementation steps on both prototypes are listed as below.

#### *Huygens:*

- Load modules for Scalasca usage:  
`module load papi scalasca`
- Recompile and link HELIUM with Scalasca:  
`scalasca -instrument /sara/sw/modules/wrappers/sara/mpfort -  
qfree=f90 -O4  
-qgss1 -qarch=auto -qtune=auto -qhot  
-o helium_scalasca helium.f90`
- Run HELIUM with Scalasca analysis:  
`scalasca -analyse poe helium_scalasca`
- The profiling results will be worked out in a separate directory. After checking the execution correctness, the profiling results can be viewed via the Scalasca GUI:  
`scalasca -examine [epik_dir]`

Where `[epik_dir]` is the directory containing all the profiling results.

### *Juropa:*

- Load modules for Intel compiler and Scalasca toolset usage:  

```
module load parastation/intel
module load UNITE
module load scalasca
```
- Recompile and link HELIUM with Scalasca:  

```
scalasca -instrument mpif90 -O3 -ipo \
-o helium_scalasca helium.f90
```
- Run HELIUM with Scalasca analysis:  

```
scalasca -analyse mpiexec -np 1540 ./helium_scalasca
```
- The profiling results will be worked out in a separate directory. After checking the execution correctness, the profiling results can be viewed via the Scalasca GUI:  

```
scalasca -examine [epik_dir]
```

where `[epik_dir]` is the directory containing all the profiling results.

Running HELIUM with Scalasca was successful on both systems. The profiling results were produced as expected. All the profiling data can be viewed via the Scalasca GUI using CUBE. Figure 19 shows an example of using Scalasca GUI to view and analyse the HELIUM profiling results on Huygens. This is very similar to that on Europa system.

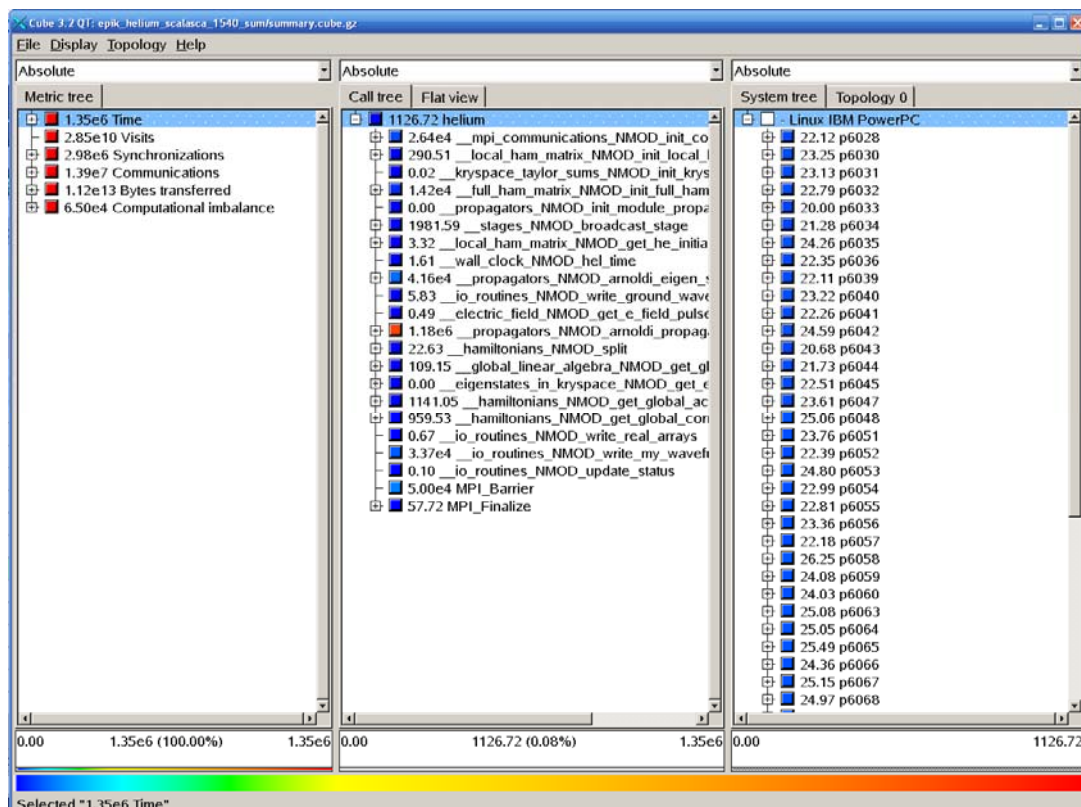


Figure 19: Viewing the HELIUM profile via Scalasca GUI on the Huygens system.

The profiling results with Scalasca were very useful for the HELIUM petascaling and optimization tasks. The main Scalasca functionalities used for HELIUM profiling included:

- **MPI communications and synchronizations profiling:**

This is to identify the most expensive MPI communications and synchronizations for reducing the communication overheads.

- **User functions execution time profiling:**  
This is quite helpful to produce the source code calling tree, identify the most expensive user functions and if possible, locate the bottlenecks in the original source code.
- **Load balance profiling:**  
It helped to analyse the reasons for communications and user functions expense.

Scalasca is portable to multiple systems. It makes easier to compare the behavior of the same application on different architectures. For example, when comparing the Scalasca profiling data on Huygens and Juropa, it can be seen clearly via the CUBE GUI that the key bottlenecks were similar, but the percentage of synchronizations is higher on Juropa.

### Using Scalasca for NAMD profiling

In contrast to HELIUM, which builds with a standard compiler call, building an NAMD executable is more complex. As discussed when instrumenting an executable for Scalasca, the developer has to prepend all compiler/linker calls with a call to the Scalasca executable.

When building an NAMD executable at least the link step is triggered by calling the `charmcc` command, which is the compile command of the Charm++ installation. This command sets up the environment required when using the Charm++ library and invokes the system compiler or linker. To instrument the NAMD executable the call to Scalasca has to be added to Charm++ scripts. For a C++ code, the compiler invoked by `charmcc` appears to be defined by the `CMK_CXX` variable and the linker is defined by the `CMK_LDXX` variable, both are set in the subdirectory of `src/arch` relevant for your system. The file is typically named `conv-mach.sh`, but for some architecture different names are used and some experimentation will be required, when searching for the right place.

During this investigation we did not manage to successfully instrument a NAMD executable with Scalasca. The reason is that the compilation of key NAMD objects failed when “`scalasca -instrument`” was added to the `CMK_CXX` variable. Within the time available, we did not manage to overcome these problems. We can only speculate about the cause.

While not managing to get a full profile on the HECToR system and the JUROPA system, we managed to obtain results for MPI profiling with Scalasca. On the Huygens system even that failed, the instrumented code on execution would not open the required repository to write the results. The Scalasca developers have since traced the cause to be a configuration issue of the Scalasca installation on Huygens, which according to their testing overcomes the problem. We are presently awaiting an updated Scalasca installation to confirm their findings.

We like to note that with Cray’s proprietary tool CrayPat, which uses in many aspects a similar approach to Scalasca, we had no problems obtaining a full NAMD profile for the HECToR and Louhi system. A hardware vendor which only has to cover their own systems and has easier access to undocumented features of the hardware and system software might be at an advantage here. In this context it is also interesting to note that the developers of Charm++ are introducing a profiling interface to their code and are collaborating with tool experts on exploiting this interface [11].

### *MPI profiling on Juropa*

To obtain an MPI profile from Scalasca on the Juropa system one needs to:

- Enable linking with Scalasca by adding “scalasca -instrument” to the CMK\_LDXX variable in the file `cc-mpicxx.sh` in the directory `/src/arch/mpi-linux-x86_64` of the charm++ installation.
- Load modules for Intel compiler and Scalasca toolset usage:
 

```
module load parastation/intel
module load UNITE
module load scalasca
```
- Relink the namd executable
- When executing the instrumented executable the above modules also need to be loaded in the job submission script and the `mpiexec` needs to be prefixed by “`scalasca -analyse`”.
- The results can be visualized by running “`scalasca -examine`” on the epik directory.

In Figure 20 we show a screen dump of the GUI showing the resulting profile from a 512 task run. When compared to Figure 19, it is obvious that the call tree now contains references to MPI calls only. The approach NAMD takes to data exchange makes extensive use of `MPI_Iprobe`, when Charm++ is build onto of MPI. Examination of Figure 20 shows that `MPI_Iprobe` calls do not appear in the profile. The Scalasca developers confirmed that the present version is not profiling `MPI_Iprobe`. The reason behind is that the overheads associated with calls of this nature could be substantial and distort the results. They are presently working on a solution which will be added in a future release.

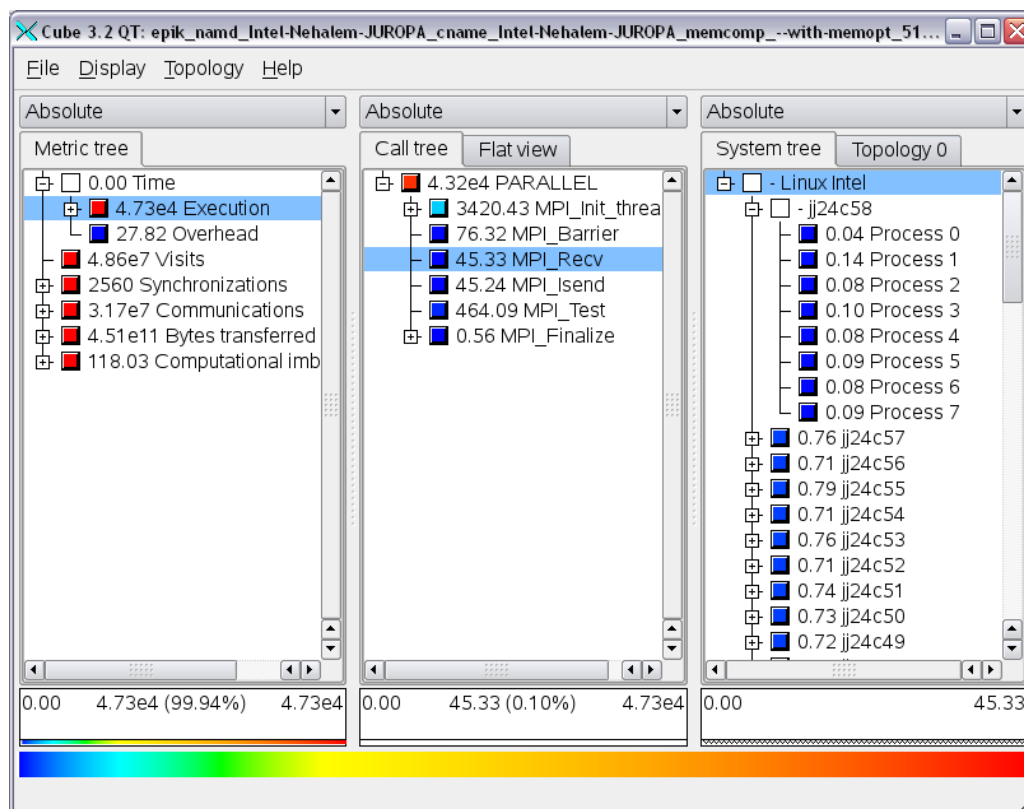


Figure 20: MPI profiling of NAMD on Juropa.

### *MPI profiling on Cray XT*

A similar procedure to Juropa also allowed MPI profiling on the Cray XT. We added “`scalasca -instrument`” to the `conv-mach.sh` file in the Charm++ subdirectory

/src/arch/mpi-crayxt and re-linked the executable after loading the Scalasca module. On execution, the aprun command inside the job submission script needs prepending with **scalasca -analyse**. A screen shot of the MPI profile is shown in Figure 21. This is also from a 512 task run.

Figure 21 gives an example of counting the number of MPI\_Isend commands issued. In the right hand column one can see a break-down by nodes and cores. The green color in the right hand column against process 0 shows that this is issuing significantly fewer MPI\_Isend calls than the other processors, while the red color against process 1 shows that this process is issuing more calls than the other processors. This is a good example of how Scalasca's use of color to guide the analyst quickly to the potential trouble spots.

As noted the screen shot is from a 512 task run. The GUI is well designed for the task of analyzing performance data from experiments with several hundred or thousand tasks. On start-up the GUI gives the analyst an overview of the application quickly and allows getting very detailed information on individual tasks by expanding a few boxes.

As expected from the feedback received from the Scalasca developers, MPI\_Iprobe is also not profiled on the Cray XT.

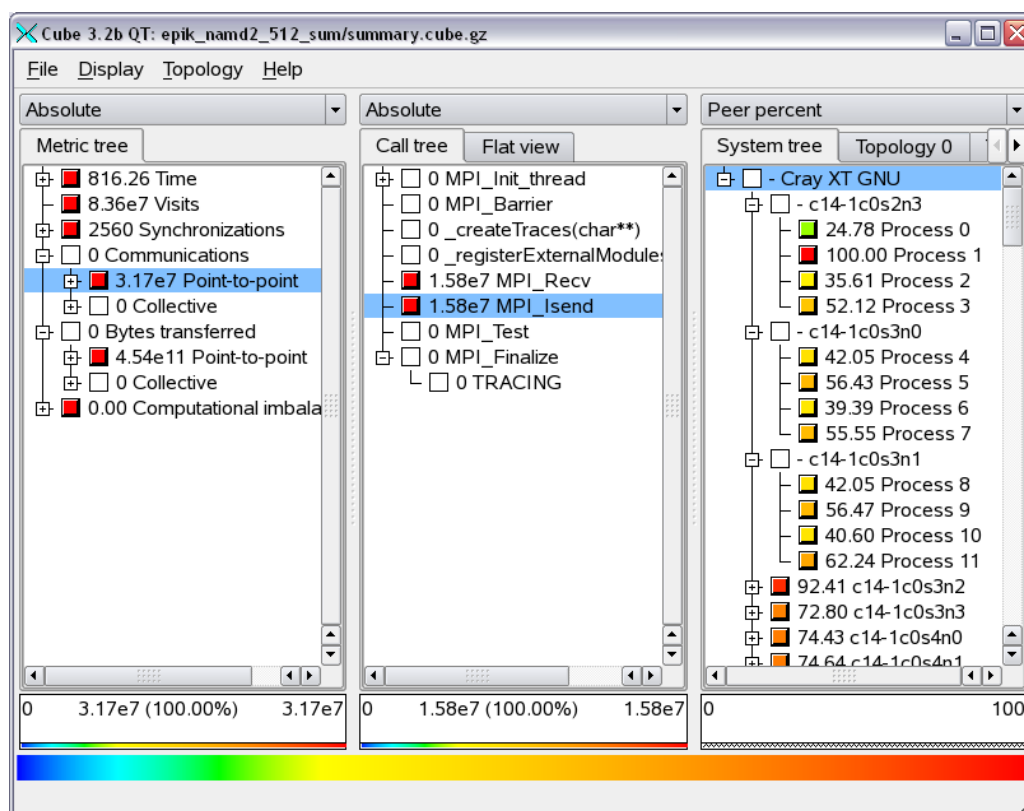


Figure 21: MPI profiling of NAMD on HECToR.

## 7.6 Scalability Experiments with VPA

Maciej Filocha and Maciej Szpindler, PSNC, Poland

We have used VPA to analyse the progress of optimizations works with the SIESTA package on Cell architecture. We have focused on performance issues on a single Cell-based QS22 blade server - two PowerXCell processors (2 PowerPC cores and 16 SPU cores), see Figure 22 for a sample-based profile of the computational kernel.

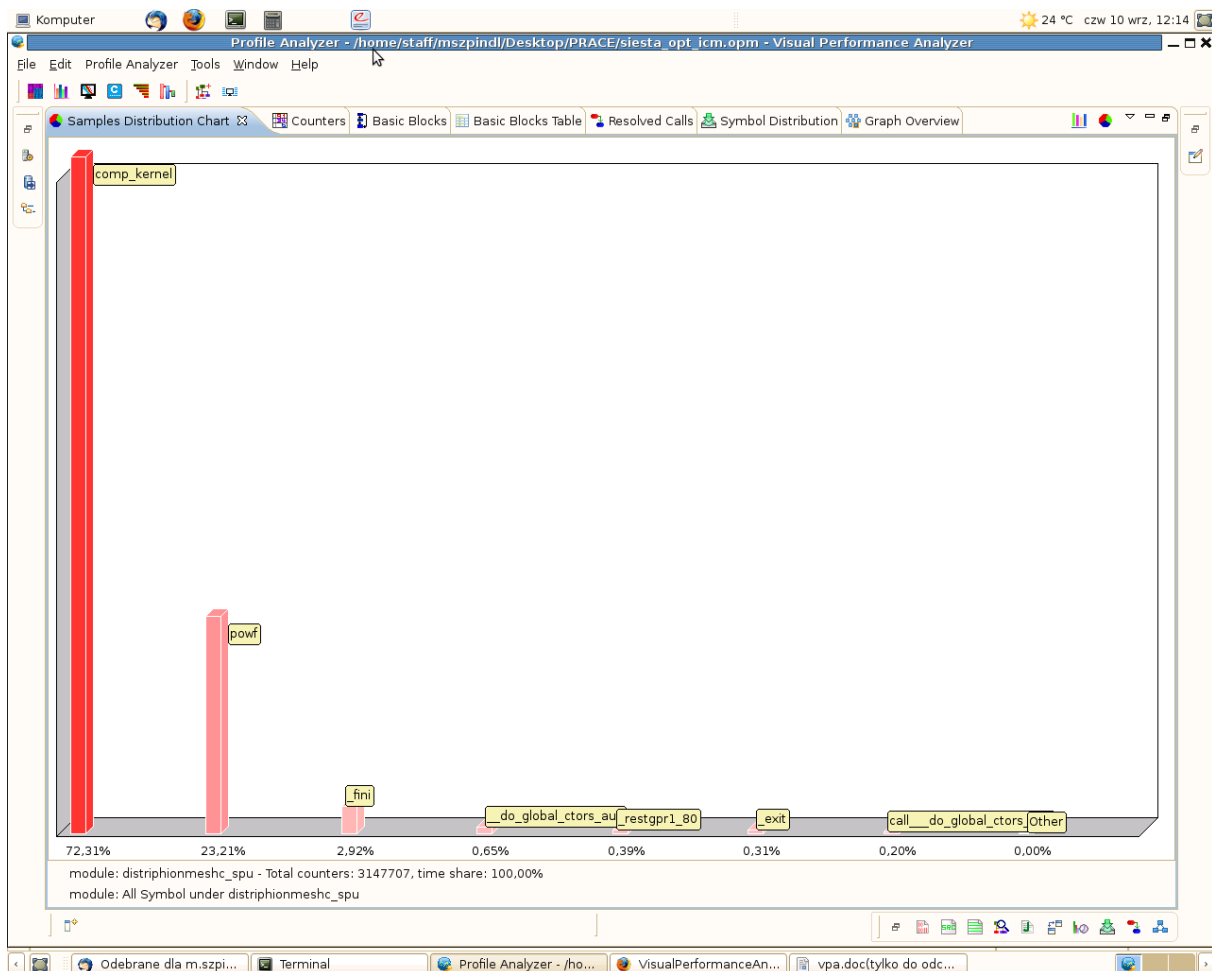


Figure 22: Screenshot: Profile Analyzer with sample-based profile.

In order to generate a sample-based profile one must use the `oprofile` tool (under Linux) to run a program and collect samples. This requires having root permissions which is a little embarrassing. Nevertheless we have found this functionality very useful during single-core performance optimization phase. One can browse the source code annotated with a profile data line-by-line. There is also a functionality of browsing de-assembled instructions mixed together with a sample-based statistics. Another very useful feature is the ability to show profile results per statement/instruction, in one screen. Figure 23 is an example of this.

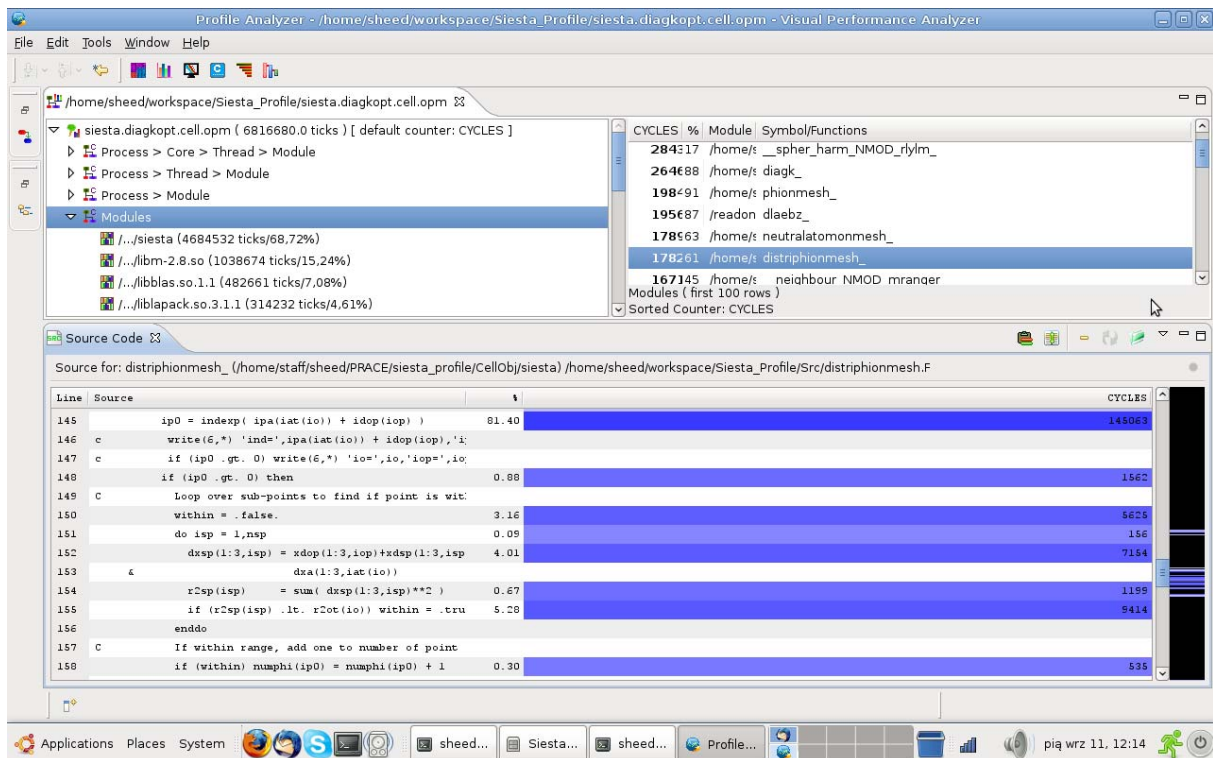


Figure 23: Screenshot: Profile Analyzer showing profile-based annotated source code.

## 7.7 Descriptions of Application Benchmark Codes

This section contains brief descriptions of the benchmark applications from the PABS.

### 7.7.1 QCD

Lukas Arnold, FZJ Germany

The QCD-benchmark contains five kernels. These kernels are executed within a wrapper application (src/qcd-bench.c). This is done by calling one function (**kernel\_x**) for each kernel, which does the initialization, the computation and the finalization of the kernel. In all cases this is (was) the original main function. Each call of the kernel functions includes calls to the wrapper to trigger diagnostics. Some modifications to the code needed to be done, in order to be compatible to the benchmarking environment. The general structure of a kernel separates into three phases: initial, run and finalize. Therefore the diagnostic functions (**jube\_kernel\_\***, the corresponding source code is located in src/qcd-diag.h) are called at the beginning of each phase and at the end of the finalize phase.

The compilation of the qcd benchmark is the following: each kernel compiles it's sources using the parameter set in src/kernel\_x/Makefile.defs, which will be included by the corresponding src/kernel\_x/Makefile. The main make file src/Makefile calls each kernels make files, with an object archive src/kernel\_x.a as target, compiles the wrapper application and links the wrapper and archive objects to one executeable. The whole application is a fortran 90 / C mixture.

The parameter files needed for each kernel (kernel\_x.input\*) have to be located in the execution directory and are used to specify for example the problem size. At the moment the problem size and processor distribution for kernel\_C needs to be specified at the compilation.

#### kernel A

name	BQCD
label	kernel_A
short label	KA
kernel origin	Berlin Quantum ChromoDynamics program (BQCD), DEISA benchmark suite
kernel contact person	Hinnerk Stueben
kernel code status	2008/08/25
problem size parameter	KA_N{X,Y,Z,T}, 4D lattice
problem run time parameter	KA_MAXITER, iteration steps
other needed parameter	KA_P{X,Y,Z,T}, distribution of processes in 4D
	KA_LIBCOMM, see readme section
	KA_LIBCLOVER, see readme section
	KA_LIBD, see readme section
notes	



**kernel B**

name	SU3_AHiggs
label	kernel_B
short label	KB
kernel origin	University of Oulu, Finland, DEISA benchmark suite
kernel contact person	Kari Rummukainen
kernel code status	2008/08/22
problem size parameter	KB_NX, x component of the 3D grid
	KB_NY, y component of the 3D grid
	KB_NZ, z component of the 3D grid
problem run time parameter	KB_MAXITER, iteration steps
other needed parameter	
notes	number of processes needs to be a power of 2

**kernel C**

name	not stated by the authors
label	kernel_C
short label	KC
kernel origin	private communication
kernel contact person	Bjoern Leder
kernel code status	2008/09/22
problem size parameter	KC_N{X,Y,Z,T}, local size of the 4D grid in {x,y,z,t}-direction
problem run time parameter	
other needed parameter	KC_P{X,Y,Z,T}, number of processes in {x,y,z,t}-direction
notes	local grid size must be a multiple of 4 and not smaller than 8

**kernel D**

name	not stated by the authors
label	kernel_D
short label	KD
kernel origin	private communication
kernel contact person	Carsten Urbach
kernel code status	2008/10/23
problem size parameter	KD_NL, size of the 4D grid in {x,y,z}-direction

	KD_NT, size of the 4D grid in t-direction
problem run time parameter	
other needed parameter	KD_P{X,Y,Z}, number of processes in {x,y,z}-direction, the number of processes in t-direction is computed
notes	

**kernel E**

name	Wuppertal Portable Inverter
label	kernel_E
short label	KE
kernel origin	private communication
kernel contact person	Stefan Krieg
kernel code status	2008/11/10
problem size parameter	KE_N{X,Y,Z,T}, size of the 4D grid in {x,y,z,t}-direction
problem run time parameter	KE_MAXITER, number of iteration steps
other needed parameter	KE_N{X,Y,Z,T}, number of processes in {x,y,z,t}-direction
notes	

**7.7.2 Quantum\_Espresso**

Carlo Cavazzoni

CINECA Italy

QUANTUM ESPRESSO is an integrated suite of computer codes for electronic-structure calculations and materials modeling, based on density-functional theory, plane waves, and pseudopotentials (norm-conserving, ultrasoft, and projector-augmented wave). QUANTUM ESPRESSO stands for opEn Source Package for Research in Electronic Structure, Simulation, and Optimization. It is freely available to researchers around the world under the terms of the GNU General Public License. QUANTUM ESPRESSO builds upon newly restructured electronic-structure codes that have been developed and tested by some of the original authors of novel electronic-structure algorithms and applied in the last twenty years by some of the leading materials modeling groups worldwide. Innovation and efficiency are still its main focus, with special attention paid to massively-parallel architectures, and a great effort being devoted to user friendliness. QUANTUM ESPRESSO is evolving towards a distribution of independent and inter-operable codes in the spirit of an open-source project, where researchers active in the field of electronic-structure calculations are encouraged to participate in the project by contributing their own codes or by implementing their own ideas into existing codes.

QUANTUM ESPRESSO implements a variety of methods and algorithms aimed at a chemically realistic modeling of materials from the nanoscale upwards, based on the solution of density-functional theory (DFT) problem, using a plane waves (PW) basis set and pseudopotentials (PP) to represent electron-ion interactions. The codes are constructed around the use of periodic boundary conditions, which allows for a straightforward treatment of infinite crystalline systems, and an efficient convergence to the thermodynamic limit for aperiodic but extended systems, such as liquids or amorphous materials. Finite systems are also treated using supercells; if required, open-boundary conditions can be used through the use of the density-countercharge method. QUANTUM ESPRESSO can thus be used for any crystal structure or supercell, and for metals as well as for insulators. The atomic cores can be described by separable norm-conserving (NC) PPs, ultra-soft (US) PPs, or by projector-augmented wave (PAW) sets. Many different exchange-correlation functionals are available in the framework of the local-density (LDA) or generalized-gradient approximation (GGA), plus advanced functionals like Hubbard U corrections and few meta-GGA and hybrid functionals.

The complete QUANTUM ESPRESSO distribution is relative large: about 240,000 lines of code, excluding copies of the external libraries. With such a sizable code basis, modularization becomes necessary. QUANTUM ESPRESSO is presently divided into several executables, performing different types of calculations, although some of them have overlapping functionalities. Typically there is a single set of functions/ subroutines or a single Fortran 90 module that performs each specific task (e.g. matrix diagonalizations, or potential updates), but there are still important exceptions to this rule, reflecting the different origin and different styles of the original components. QUANTUM ESPRESSO has in fact been built out of the merge and re-engineering of different packages, that were independently developed for several years. In the following, the main components are briefly described.

### **PWscf**

PWscf implements an iterative approach to reach selfconsistency, using at each step iterative diagonalization techniques, in the framework of the plane-wave pseudopotential method. Both separable NC-PPs and US-PPs are implemented; recently, also the projector augmented-wave method has been added. PWscf can use the established LDA and GGA exchange-correlation functionals, including spin-polarization and can treat non-collinear magnetism induced by relativistic effects (spinorbit interactions) or by complex magnetic interactions.

### **CP**

The CP code is the massively-parallel module for Car- Parrinello ab-initio Molecular Dynamics (MD). CP can use both NC PPs and US PPs. In the latter case, the electron density is augmented through a Fourier interpolation scheme in real space ("box grid") that is particular efficient for large scale calculations. CP implements the same functionals as in PWscf, with the exception of hybrid functionals.

### **Parallelization**

Keeping the pace with the evolution of high-end supercomputers is one of the guiding lines in the design of QUANTUM ESPRESSO, with a significant effort being dedicated to porting it to the latest available architectures. This effort is motivated not only by the need to stay at the forefront of architectural innovation for large to very-large scale materials science simulations, but also by the speed at which hardware features specifically designed for supercomputers find their way into commodity computers. The architecture of today's supercomputers is characterized by multiple levels and layers of parallelism: the bottom layer is the one affecting the instruction set of a single core (simultaneous multithreading, hyperthreading); then one has parallel processing at processor level (many CPU cores inside a

single processor sharing caches) and at node level (many processors sharing the same memory inside the node); at the top level, many nodes are finally interconnected with a high-performance network. The main components of the QUANTUM ESPRESSO distribution are designed to exploit this highly structured hardware hierarchy. High performance on massively parallel architectures is achieved by distributing both data and computations in a hierarchical way across available processors, ending up with multiple parallelization levels that can be tuned to the specific application and to the specific architecture. This remarkable characteristic makes it possible for the main codes of the distribution to run in parallel on most or all parallel machines with very good performance in all cases.

More in detail, the various parallelization levels are geared into a hierarchy of processor groups, identified by different MPI communicators. In this hierarchy, groups implementing coarser-grained parallel tasks are split into groups implementing finer-grained parallel tasks. The first level is image parallelization, implemented by dividing processors into image groups, each taking care of one or more images (i.e. a point in the configuration space, used by the NEB method). The second level is pool parallelization, implemented by further dividing each group of processors into npool pools of processors, each taking care of one or more k-points. The third level is plane-wave parallelization, implemented by distributing real- and reciprocal-space grids across the nPW processors of each pool. The final level is task group parallelization, in which processors are divided into ntask task groups of  $nFFT = nPW/ntask$  processors, each one taking care of different groups of electron states to be Fourier-transformed, while each FFT is parallelized inside a task group. A further parallelization level, linear-algebra, coexists side-to-side with plane-wave parallelization, i.e. they take care of different sets of operations, with different data distribution. Linear-algebra parallelization is implemented both with custom algorithms and using ScaLAPACK, which on massively parallel machines yield much superior performances. The table below contains a summary of the five levels currently implemented. With the recent addition of the two last levels, most parallelization bottlenecks have been removed, both computations and data structures are fully distributed, scalability on parallel machines is only limited by the physical sizes of the system being simulated. Scalability is thus guaranteed for large-scale simulations. This being said, it is obvious that the size and specific nature of the specific application sets quite naturally limits to the maximum number of processors up to which the performances of the various codes are expected to scale. For instance, the number of images in a NEB calculation sets a natural limit to the level of image groups, or the number of electronic bands sets a limit for the parallelization of the linear algebra operations. Moreover some numerical algorithms scale better than others. For example, the use of norm-conserving pseudopotentials allows for a better scaling than ultrasoft pseudopotentials for a same system, because a larger plane wave basis set and a larger real- and reciprocal-space grids are required in the former case. On the other hand, using ultrasoft pseudopotentials is generally faster because the use of a smaller basis set is obviously more efficient, even though the overall parallel performance may be not as good.

#### Summary of parallelization levels in QUANTUM ESPRESSO

group	distributed quantities	communications	performance
image	NEB images	very low	linear CPU scaling, fair to good load balancing; does not distribute RAM
pool	k-points	low	almost linear CPU scaling, fair to good load balancing; does not distribute RAM

plane-wave	plane waves, G-vector coefficients, R-space FFT arrays	high	good CPU scaling, good load balancing, distributes most RAM
task	FFT on electron states	high	improves load balancing
linear algebra	subspace Hamiltonians	very high	improves scaling, and constraints matrices distributes more RAM

### 7.7.3 NAMD

Joachim Hein

EPCC United Kingdom

NAMD is a widely used molecular dynamics application designed to simulate bio-molecular systems on a wide variety of compute platforms. NAMD is developed by the “Theoretical and Computational Biophysics Group” at the University of Illinois at Urbana Champaign. In the design of NAMD particular emphasis has been placed on scalability when utilising a large number of processors. The application can read a wide variety of different file formats for e.g. force fields, protein structure, which are commonly used in bio-molecular science.

A NAMD license can be applied for on the developer’s website free of charge. Once the license has been obtained, binaries for a number of platforms and the source can be downloaded from the website. The entire process is very quick.

Deployment areas of NAMD include pharmaceutical research by academic and industrial users. NAMD is particularly suitable when the interaction between a number of proteins or between proteins and other chemical substances is of interest. Typical examples are vaccine research and transport processes through cell membrane proteins.

### 7.7.4 CPMD

Albert Farrés

BSC-CNS Spain

The CPMD code is a plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics. Its first version was developed by Jurg Hutter at IBM Zurich Research Laboratory starting from the original Car-Parrinello codes. During the years many people from diverse organizations contributed to the development of the code and of its pseudopotential library:

Michele Parrinello, Jurg Hutter, D. Marx, P. Focher, M. Tuckerman, W. Andreoni, A. Curioni, E. Fois, U. Roetlisberger, P. Giannozzi, T. Deutsch, A. Alavi, D. Sebastiani, A. Laio, J. VandeVondele, A. Seitsonen, S. Billeter and others.

The current version, 3.13, is copyrighted jointly by IBM Corp and by Max Planck Institute, Stuttgart, and is distributed free of charge to non-profit organizations. CPMD runs on many different computer architectures and it is well parallelized (MPI and Mixed MPI/SMP).

Main characteristics of CPMD:

- works with norm conserving or ultrasoft pseudopotentials
- LDA, LSD and the most popular gradient correction schemes; free energy density functional implementation
- isolated systems and system with periodic boundary conditions; k-points
- molecular and crystal symmetry
- wavefunction optimization: direct minimization and diagonalization
- geometry optimization: local optimization and simulated annealing
- molecular dynamics: constant energy, constant temperature and constant pressure
- path integral MD
- response functions
- excited states
- many electronic properties
- time-dependent DFT (excitations, molecular dynamics in excited states)
- coarse-grained non-Markovian metadynamics

### 7.7.5 Code\_Saturne

Andrew Sunderland, Charles Moulinec  
STFC United Kingdom

*Code\_Saturne*® is a multipurpose Computational Fluid Dynamics (CFD) software, which has been developed by EDF-R&D (France) since 1997. The code was originally designed for industrial applications and research activities in several fields related to energy production; typical examples include nuclear power thermal-hydraulics, gas and coal combustion, turbo-machinery, heating, ventilation, and air conditioning. In 2007, EDF released the code as open-source and this provides both industry and academia to benefit from its extensive pedigree. *Code\_Saturne*®'s open-source status allows for answers to specific needs that cannot easily be made available in commercial “black box” packages. It also makes it possible for industrial users and for their subcontractors to develop and maintain their own independent expertise and to fully control the software they use.

*Code\_Saturne*® is based on a co-located finite volume approach that can handle three-dimensional meshes built with any type of cell (tetrahedral, hexahedral, prismatic, pyramidal, polyhedral) and with any type of grid structure (unstructured, block structured, hybrid). The code is able to simulate either incompressible or compressible flows, with or without heat transfer, and has a variety of models to account for turbulence. Dedicated modules are available for specific physics such as radiative heat transfer, combustion (e.g. with gas, coal and heavy fuel oil), magneto-hydro dynamics, and compressible flows, two-phase flows. There are extensions for specific applications. For example the related code *Mercurie\_Saturne* can be used to model atmospheric environment and flows.

The software comprises of around 500 000 lines of source code, with around 50% written in Fortran90, 40% in C and 10% in Python. The code is portable to Linux-based PCs and

several HEC architectures – e.g. SGI Origin, several PC clusters, Cray XT series and IBM PWR series.

### *Industrial Usage*

*Code\_Saturne*® has been developed specifically for industrial usage by EDF.

The main industrial application examples include:

- The Study of Flows in Pressurized Water Reactors (PWRs), in particular:
  - The modelling of flows around bundles of pipes in nuclear reactors in order to maintain internal structures of reactor vessels. Knowledge of flow conditions in the lower core is especially important to study deformation and fretting of the fuel assemblies.
  - Pressurized thermal shock on a PWR vessel. This involves the injection of cold water in a PWR vessel following the loss of coolant. A coupled local 3D approach, involving *Code\_Saturne*® for the fluid and SYRTHES for heat transfer in the solid, models the temperature evolution at critical locations in the metal of the vessel.
- Pulverized coal furnace slagging
  - Ash deposit on shell plates and exchangers reduces the efficiency of pulverized coal-fired boilers. This ‘slagging’ is investigated with *Code\_Saturne*® in order to compute the temperature and flow field of the carrying phase. A high accuracy Lagrangian approach is invoked to track individual pulverized coal particles.
- Air Quality in Hospital Operating Theatres
  - In hospitals, and particularly in operating theatres, infections can be transmitted via airborne routes and therefore controlling air quality is of high importance. *Code\_Saturne*® enables an accurate representation of the equipment, medical staff, patients and heating, ventilation and air-conditioning systems in order to identify high-risk zones for contamination and to evaluate the efficiency of the ceiling ventilation device.

## 7.7.6 GADGET

Orlando Rivera

LRZ Germany

Gadget is a freely available code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory written by Volker Springel, Max-Planck-Institute for Astrophysics, Garching, Germany. GADGET uses an explicit communication model that is implemented with the standardized MPI communication interface. The code can be run on essentially all supercomputer systems presently in use, including clusters of workstations or individual PCs.

GADGET computes gravitational forces with a hierarchical tree algorithm (optionally in combination with a particle-mesh scheme for long-range gravitational forces) and represents fluids by means of smoothed particle hydrodynamics (SPH). The code can be used for studies of isolated systems, or for simulations that include the cosmological expansion of space, both with or without periodic boundary conditions.

In all these types of simulations, GADGET follows the evolution of a self-gravitating collisionless N-body system, and allows gas dynamics to be optionally included. Both the force computation and the time stepping of GADGET are fully adaptive, with a dynamic range which is, in principle, unlimited.

GADGET can therefore be used to address a wide array of astrophysics interesting problems, ranging from colliding and merging galaxies, to the formation of large-scale structure in the Universe. With the inclusion of additional physical processes such as radiative cooling and heating, GADGET can also be used to study the dynamics of the gaseous intergalactic medium, or to address star formation and its regulation by feedback processes. Gadget is used regularly in several centers and it has a very active user community.

### 7.7.7 EUTERPE (TORB)

Xavier Saez

BSC-CNS Spain

The EUTERPE gyrokinetic code was created at the EPFL in Lausanne as a global linear particle in cell code for studying electrostatic plasma instabilities. It allows three-dimensional turbulence simulations using a plasma equilibrium calculated with the VMEC code as a starting point. EUTERPE was further developed at the Max Planck IPP and several linear calculations of ion temperature gradient (ITG) driven turbulence in tokamak and stellarator geometry have been carried out using it. The code has been afterwards heavily optimized and improved and non-linear dynamics have been included.

The EUTERPE code solves the linear electrostatic gyrokinetic equation for ions (the electrons are assumed to be adiabatic) in the whole plasma domain (full radius) for a three-dimensional realistic stellarator geometry. Numerically the code uses a Monte Carlo method, the so-called particle-in-cell (PIC) method, to follow the characteristics of the gyrokinetic partial differential equation. In order to decrease the statistical noise a  $\delta f$ -approach, which is equivalent to a control variates method, is used. This method serves to reduce the noise by orders of magnitude. The charge assignment process and the discretization of the three-dimensional Helmholtz equation have been unified by employing tensor product B-splines as finite elements. The resulting very large system of equations is solved using the PETSc library which simplifies parallelization of the sparse matrix operations and provides various parallel iterative solvers and preconditioners. The parallelization strategy consists of a domain decomposition concept in the toroidal direction. It has been implemented with the Message-Passing Interface (MPI library).

### 7.7.8 WRF

Andrew Porter

STFC United Kingdom



The Weather Research & Forecasting (WRF) model was developed at the National Center for Atmospheric Research (NCAR) in the United States as a regional- to global-scale model for both research applications and operational weather-forecast systems.

WRF is now used as the National Oceanic and Atmospheric Administration's primary forecast model, for forecasts of 1-3 days ahead, and is used by weather agencies all over the world (including weather agencies in Indo-China and Asia). As of June 2008 there are in excess of 6000 registered WRF users.

The WRF system incorporates two different dynamics solvers; the Advanced Research WRF (ARW) solver (developed by Mesoscale and Microscale Meteorology Division of NCAR) and the Nonhydrostatic Mesoscale Model solver (developed by the National Centers for Environmental Prediction, US). In this document we will discuss only the ARW version of the WRF modeling system.

The ARW solves the fully-compressible, non-hydrostatic Euler equations on an Arakawa C-grid staggering in the horizontal plane and a terrain-following, dry hydrostatic pressure vertical coordinate. There are 2<sup>nd</sup>- to 6<sup>th</sup>-order advection options for spatial discretization in both horizontal and vertical directions. Integration in time is performed using a time-split method with a 2<sup>nd</sup>- or 3<sup>rd</sup>-order Runge-Kutta scheme with a smaller time step for acoustic- and gravity-wave modes. The model supports periodic, open, symmetric and specified lateral boundary conditions and is capable of whole-globe simulations using polar Fourier filtering and periodic east-west boundary conditions.

The WRF model has the ability to incorporate a one- or two-way nested sub-structure (including moving nests) that enables it to be used for high-resolution case studies.

The WRF model has, from the outset, been designed and written to perform well on massively-parallel computers. It is written in Fortran90 and can be built in serial, parallel (MPI) and mixed-mode (OpenMP and MPI) forms, simply by choosing the appropriate option during the configure process.

The code has been ported to a range of high-end computing architectures including the IBM Blue Gene, IBM Power series and the Cray XT series.

The WRF code is freely downloadable following on-line registration and is explicitly declared to be in the public domain (see <http://www.mmm.ucar.edu/wrf/users/public.html>).

### 7.7.9 NEMO

John Donners

SARA the Netherlands

NEMO (Nucleus for European Modelling of the Ocean) is a state-of-the-art modeling framework for oceanographic research, operational oceanography seasonal forecast and climate studies.

NEMO includes:

- 4 major components
  - the blue ocean (ocean dynamics, NEMO-OPA)
  - the white ocean (sea-ice, NEMO-LIM)
  - the green ocean (biogeochemistry, NEMO-TOP) ;
  - the adaptative mesh refinement software (AGRIF) ;

- some reference configurations allowing to set-up and validate the applications;
- a set of scripts and tools (including pre- and post-processing) to use the system.

NEMO is used by a large community: 240 projects in 27 countries (14 in Europe, 13 elsewhere), 350 registered users (numbers for year 2008). NEMO is available under CeCILL license (public license).

#### 7.7.10 CP2K

Pekka Manninen

CSC Finland

CP2K is a freely available (GPL) program to perform atomistic and molecular simulations of solid state, liquid, molecular and biological systems. It provides a general framework for different methods such as e.g. density functional theory (DFT) using a mixed Gaussian and plane waves approach (GPW), and classical pair and many-body potentials. It is very well and consistently written, standards-conforming Fortran 95, parallelized with MPI and in some parts with hybrid OpenMP+MPI as an option.

CP2K provides state-of-the-art methods for efficient and accurate atomistic simulations, sources are freely available and actively improved. It is therefore easy to give the code a try, and to make modifications as needed. However, The CP2K code comes with little documentation and without any warranty. No official release has been made. Substantial changes, improvements and bug fixes will be made at irregular intervals. Using the code for production quality simulations is possible but requires detailed knowledge about the active development. It has an active international development team, with the unofficial headquarters in the University of Zürich.

#### 7.7.11 GROMACS

Sebastian von Alfthan

CSC Finland

The GROMACS website contains the following brief description of the GROMACS code:  
[http://www.gromacs.org/About\\_Gromacs](http://www.gromacs.org/About_Gromacs).

GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles.

It is primarily designed for biochemical molecules like proteins and lipids that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the nonbonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers.

GROMACS supports all the usual algorithms you expect from a modern molecular dynamics implementation, (check the online reference or manual for details), but there are also quite a few features that make it stand out from the competition:

- GROMACS provides *extremely high performance* compared to all other programs. A lot of algorithmic optimizations have been introduced in the code; we have for

instance extracted the calculation of the virial from the innermost loops over pairwise interactions, and we use our own software routines to calculate the inverse square root. The innermost loops are generated automatically in either C or Fortran at compile time, with optimizations adopted to your architecture. Assembly loops using SSE and 3DNow! multimedia instructions are provided for i386 processors, separate versions using all x86-64 registers are used on Opteron x86-64 and Xeon EM64t machines. This results in exceptional performance on inexpensive PC workstations, and for Pentium IV/Opteron processors there are also SSE2 double precision assembly loops. There are new manually tuned assembly loops for ia64 (both single and double precision), and last but certainly not least we have written AltiVec assembly loops both for Linux and Mac OS X. Gromacs is normally 3-10 times faster than any other program; check the article in Journal of Molecular Modeling (reference can be found under resources) for a comparison benchmark.

- GROMACS is user-friendly, with topologies and parameter files written in clear text format. There is a lot of consistency checking, and clear error messages are issued when something is wrong. Since the C preprocessor is used, you can have conditional parts in your topologies and include other files. You can even compress most files and GROMACS will automatically pipe them through gzip upon reading.
- There is no scripting language - all programs use a simple interface with command line options for input and output files. You can always get help on the options by using the **-h** option, or use the extensive manuals provided free of charge in electronic or paper format. There is also an integrated graphical user interface available for all programs.
- As the simulation is proceeding, GROMACS will continuously tell you how far it has come, and what time and date it expects to be finished.
- Both run input files and trajectories are independent of hardware endian and can thus be read by any version GROMACS, even if it was compiled using a different floating-point precision. All files from GROMACS 2.0 can further be used in the new version 3!
- GROMACS can write coordinates using lossy compression, which provides a very compact way of storing trajectory data. The accuracy can be selected by the user.
- GROMACS comes with a large selection of flexible tools for trajectory analysis - you won't have to write any code to perform routine analyses. The output is further provided in the form of finished Xmgr/Grace graphs, with axis labels, legends, etc. already in place!
- A basic trajectory viewer that only requires standard X libraries is included, and several external visualization tools can read the GROMACS file formats.
- GROMACS can be run in parallel, using standard MPI communication.
- GROMACS contains several state-of-the-art algorithms that make it possible to extend the time steps in simulations significantly, and thereby further enhance performance without sacrificing accuracy or detail.
- The package includes a fully automated topology builder for proteins, even multimeric structures. Building blocks are available for the 20 standard aminoacid residues as well as some modified ones, the 4 nucleotide and 4 deoxynucleotide residues, several sugars and lipids, and some special groups like hemes and several small molecules.
- There is ongoing development to extend GROMACS with interfaces both to Quantum Chemistry and Bioinformatics/databases.
- GROMACS is *Free Software*, available under the [GNU General Public License](#).

### 7.7.12 NS3D

Harald Klimach

HLRS Germany

The code NS3D has been developed for direct numerical simulation (DNS) of the compressible Navier-Stokes equations. Spatial discretization in streamwise and normal directions is done by 6th-order compact finite differences. Since periodicity is assumed in spanwise direction, a spectral ansatz is used here. Instead of evaluating some sort of viscous fluxes, second derivatives are computed directly, which better resolves viscous terms. Time integration is done using the standard 4th-order Runge-Kutta scheme. Execution on multiple processors is implemented by domain decomposition (MPI) and shared-memory parallelization. With the combination of grid transformation and domain decomposition, the code can be applied to a wide range of geometric configurations.

Applications range from sub- to supersonic flows with emphasis on laminar-turbulent transition, aeroacoustics and flow control. The code has been developed at the Institute for Aerodynamics and Gasdynamics (IAG) at the Universität Stuttgart.

Right now, it is applied only at this institution but usage includes also industrial cooperations. For Airbus, new high-lift configurations are simulated where active flow control is intended to avoid separation of the flow. Another example is the cooperation with iTronic GmbH where a new acoustic sensor of surface roughness was investigated.

### 7.7.13 AVBP

Bertrand Cirou

CINES France

AVBP is one of the very few codes that can simulate turbulent combustion taking place in turbulent flows within complex geometries. It has been jointly developed in France by CERFACS and IFP to perform Large Eddy Simulation (LES) of reacting flows, in gas turbines, piston engines or industrial furnaces. This compressible LES solver on unstructured and hybrid grids is employed in multiple configurations for industrial gas turbines (Alstom, Siemens, Turbomeca), aero gas turbines (SNECMA, Turbomeca), rocket engines (SNECMA DMF Vernon), laboratory burners used to study unsteady combustion (Cambridge, École Centrale Paris, Coria Rouen, DLR, Karlsruhe University, Munich University).

In 2008, the DoE (U.S. Department of Energy) for research in massively parallel applications has allocated 4,000,000 hours of CPU to a project using AVBP, also in the scope of the INCITE project. Further, AVBP has been used in many EC projects (in FP4, FP5, FP6 and FP7).

#### 7.7.14 HELIUM

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

EPCC United Kingdom

The application HELIUM simulates the interaction between an intense linearly-polarized laser pulse and a Helium atom. It does so by numerically solving a time-dependent Schrodinger equation for the full system.

The source code was developed by the Queen's University Belfast and has access restrictions: permission for using HELIUM must be obtained from Ken Taylor <k.taylor@qub.ac.uk>. For the PRACE project, all the project members outside UK are required to sign the HELIUM NDA before usage.

#### 7.7.15 TRIPOLI-4

Jacques David

CEA France

TRIPOLI-4 is a computer code simulating the 3D transport of neutrons, photons, electrons and positrons with the Monte Carlo method. It uses full pointwise as well as multigroup cross-sections. It addresses radiation shielding and neutronic (subcritical and critical) problems. The code has been validated through several hundred benchmarks as well as measurement campaigns, and is used by the french nuclear industry. It is available from the OECD/NEA databank.

TRIPOLI-4 is directly compatible with pointwise cross-sections produced by the NJOY processing code system. It may also be run with homogenized multigroup cross-sections, and multigroup cross-sections with probability tables. It computes the following quantities: flux, current, reaction rates, dose equivalent rates, deposit of energy, recoil energy and multiplication factor (in criticality mode). The associated types of estimator are collision, tracklength, surface and point detectors. The geometry may be described by predefined shapes combination and/or surface' equations. Complex lattices and lattices of lattices are available. The source description is factorized in space, energy, direction and time, providing the user with an extended choice through tabulated or analytical laws. TRIPOLI-4 makes use of several variance reduction techniques, which are essential in shielding calculations. The code has perturbation estimation capabilities (concentration, density), using the correlated sampling technique. TRIPOLI-4 can be executed in a parallel mode on workstation networks as well as massively parallel machines. The communication graph is very simple and fault-tolerant: should some processor be stopped for any reason, the whole simulation would keep on until the right number of batches is obtained from the remaining processors.

### 7.7.16 PEPC

Lukas Arnold

FZJ Germany

PEPC - Pretty Efficient Parallel Coulomb-solver - is a parallel tree-code for rapid computation of long-range Coulomb forces in N-body particle systems. Based on the original Barnes-Hut algorithm, the code uses successively larger multipole-groupings of distant particles to reduce the computational effort in the force calculation from the generally unaffordable  $O(N^2)$  operations needed for brute-force summation, to a more amenable  $O(N \log N)$  complexity. The parallel version is a pure MPI implementation of the Warren-Salmon 'Hashed Oct Tree' scheme, including several different variations of the tree traversal routine - the most challenging component in terms of scalability.

The public version is divided into kernel routines and 'front-end' applications, which currently include a skeleton molecular dynamics program (PEPC-E), and a code for simulating laser- or beam-plasma interactions (PEPC-B), developed at the Forschungszentrum Juelich. There is also a (non-public) gravitational version (PEGS) for modelling astrophysical discs, developed in collaboration with the University of Cologne.

The code currently runs on IBM Regatta, BlueGene/L/P and standard Linux clusters, but should be portable to any Unix-based parallel architecture. The User Guide provides an introduction to compiling and running the code.

More information, downloads and references are available at <http://www.fz-juelich.de/jsc/pepc/>.

### 7.7.17 GPAW

Jussi Enkovaara

CSC Finland

GPAW is an efficient program package for electronic structure calculations based on the density functional theory (DFT) and the time-dependent density functional theory (TD-DFT). The density-functional theory allows studies of ground state properties such as energetics and equilibrium geometries, while the time-dependent density functional theory can be used for calculating excited state properties like optical spectra. The program package includes two complementary implementations of time-dependent density functional theory, a linear response formalism and a time-propagation in real time.

The program uses the projector augmented wave (PAW) method which allows one to get rid of the core electrons and work with soft pseudo valence wave functions. The PAW method can be applied on the same footing to all elements, for example, it provides a reliable description of the transition metal elements and the first row elements with open p-shells which are often problematic for standard pseudopotentials. A further advantage of the PAW method is that it is an all-electron method (frozen core approximation) and there is a one to one transformation between the pseudo and all-electron quantities.

The equations of the (time-dependent) density functional theory within the PAW method are discretized using finite-differences and uniform real-space grids. The real-space

representation allows flexible boundary conditions as the system can be finite or periodic in one, two or three dimensions (e.g. cluster, slab, bulk). The accuracy of the discretization is controlled basically by single parameter, the grid spacing. The real-space representation allows also efficient parallelization with domain decomposition.

The program offers several parallelization levels. The most basic parallelization strategy is domain decomposition over the real-space grid. In magnetic systems it is possible to parallelize over spin, and in systems which have k-points (surfaces or bulk systems) parallelization over k-points is also possible. Furthermore, parallelization over electronic states is possible in DFT and in real-time TD-DFT calculations.

### 7.7.18 ALYA

Guillaume Houzeaux and Raúl de la Cruz

BSC-CNS Spain

The Alya System is a Computational Mechanics (CM) code with two main features. Firstly, it is *specially designed* for running with the highest efficiency standards in large scale supercomputing facilities. Secondly, it is capable of solving different physics, each one with its own modelization characteristics, in a coupled way. Both main features are intimately related, meaning that all complex coupled problems solved by Alya must retain the efficiency. Among the problems it solves are: Convection-Diffusion-Reaction, Incompressible Flows, Compressible Flows, Turbulence, Bi-Phasic Flows and free surface, Excitable Media, Acoustics, Thermal Flow, Quantum Mechanics (TDFT) and Solid Mechanics (Large strain). By *specially designed* we mean that Alya is designed from scratch to program in a flexible yet clear way every kind of CM model to run in parallel computers. That is to say that Alya is not an original sequential code parallelized afterwards, but a code so designed from scratch.

For more information: Alya home page: [http://www.bsc.es/plantillaA.php?cat\\_id=552](http://www.bsc.es/plantillaA.php?cat_id=552)

### 7.7.19 OCTOPUS

Fernando Nogueira

UC-LA Portugal

Octopus is a code that aims to simulate with great accuracy some complex electronic processes in medium to large systems. To achieve this, octopus relies on Density-Functional Theory (DFT) and in particular on its time-dependent formulation (TDDFT). The use of DFT allows Octopus to deal with systems larger than those typically studied with traditional Quantum Chemistry or Quantum Monte-Carlo techniques (e.g., molecular systems of biological interest). Although DFT is not as accurate as these techniques, its scaling with the number of electrons of the system is much more favourable. Octopus differs from most of DFT codes in several aspects:

Target problems:

- (i) Response of molecules or clusters to external perturbations:
  - a. Linear optical (i.e. electronic) response of molecules or clusters;

- b. Non-linear response to classical high-intensity electromagnetic fields, taking into account both the ionic and electronic degrees of freedom.
- (ii) Ground-state and excited state electronic properties of 1- and 2-dimensional systems, such as quantum dots.
- (iii) Photo-induced reactions of molecules (e.g., photo-dissociation, photo-isomerization, etc.).

Two different approaches to TDDFT:

- (i) “Standard” TDDFT-based linear-response theory, which provides excitation energies and oscillator strengths for ground-state to excited-state transitions.
- (ii) Explicit time-propagation of TDDFT equations, allowing the use of large external potentials, well beyond the range of validity of perturbation theory.

Methodology:

Instead of using a basis set expansion of the (Kohn-Sham) wavefunctions, octopus uses a numerical mesh. Auxiliary basis sets (plane waves, atomic orbitals) are used only when necessary. Grids can be non-uniform, adapting to the inhomogeneity of the problem. Multigrid techniques can be used to accelerate the calculations.

For most calculations, the code relies on the use of several types of pseudopotentials: Troullier–Martins, Hamann, and Hartwigsen–Goedecker–Hutter.

Technical aspects:

- (i) The code has been designed with emphasis on parallel scalability. As a consequence, it allows for multiple task divisions: k-points, Kohn-Sham states, and real space regions.
- (ii) The language of most of the code is Fortran 90. Other languages, such as C or Perl, are also used.
- (iii) It only uses standard and portable tools. The resulting code may thus run on virtually any Unix-like platform.
- (iv) The package is licensed under the GNU General Public License (GPL).

Users:

The code is used by researchers in electronic structure (condensed matter physicists, chemists, astrophysicists) studying the interaction of matter with light. It is one of the base tools of the European Theoretical Spectroscopy Facility (ETSF). octopus mailing list has more than 230 active users, and the code is downloaded from the website on average 143.3 times per month since 2004.

## 7.7.20 BSIT

Mauricio Araya

BSC-CNS

BSIT stands for Barcelona Seismic Imaging Tools, which is a tool set for computational geophysics composed by:

- Reverse Time Migration (RTM), this one is actually the code been mapped to Cell/B.E.
- Forward modeling



- Tomography
- Seismic data-base management
- Seismic data processing tools

### 7.7.21 ELMER

Mikko Lyly

CSC Finland

Elmer is a finite element software package for the solution of partial differential equations. Elmer can deal with a great number of different equations, which may be coupled in a generic manner making Elmer a versatile tool for multiphysical simulations. As an open source software, Elmer also gives the user the means to modify the existing solution procedures and to develop new solvers for equations of interest to the user.

The development of Elmer was started in 1995 as part of a national CFD technology program funded by the Finnish funding agency for technology and innovation, Tekes. The original development consortia included partners from CSC – IT Center for Science (formerly known as CSC – Scientific Computing), Helsinki University of Technology TKK, VTT Technical Research Centre of Finland, University of Jyväskylä, and Okmetic Ltd. After the five years initial project ended the development has been continued by CSC in different application fields.

In September 2005 Elmer was released under GNU General Public License (GPL). This has widened the user community, particularly the number of international users has grown. However, as the sole owner of the copyright to Elmer source code, CSC may distribute Elmer also under other licensing terms. Therefore, if GPL does not suit your purposes, you may contact the Elmer team for other licensing options.

Elmer is distributed only through the Internet. The actual distribution site may vary but the pointer to the location may always be found at <http://www.csc.fi/elmer>. The distribution of Elmer comes in three different parts: sources, binaries, and documentation. Unix users are encouraged to compile the software themselves. The compilation instructions are given at the www-page. For Windows and Macintosh a precompiled binary version of the code is also provided. The documentation of the software is already quite extensive, but unfortunately still not complete.

Everybody is welcome to contribute to the Elmer project. Often the bottle-neck is in case specification, testing and verification which may be done without in-depth knowhow of the code. Also contributions to the code are welcome. However, before granting a permission to commit to the main source file archive a Elmer Contributor Agreement has to be signed. This gives CSC the right to use contributions to Elmer under the current free software license, and also under other licenses we may use. However, this does not limit the contributors right to use the contributed code in any way.

Elmer offers a wide range of methods and techniques for the computational modeling of physical phenomena described by partial differential equations. In the following some of the most essential ones are summarized.

The Elmer package contains solvers for a variety of mathematical models. The following list summarizes the capabilities of Elmer in specialized fields:

- Heat transfer: models for conduction, radiation and phase change

- Fluid flow: the Navier-Stokes, Stokes and Reynolds equations, k- $\epsilon$  model
- Species transport: generic convection-diffusion equation
- Elasticity: general elasticity equations, dimensionally reduced models for plates and shells
- Acoustics: the Helmholtz equation
- Electromagnetism: electrostatics, magnetostatics, induction
- Microfluidics: slip conditions, the Poisson-Boltzmann equation
- Levelset method: Eulerian free boundary problems
- Quantum Mechanics: density functional theory (Kohn-Sham)

For approximation and linear system solution Elmer offers a great number of possibilities. The following list summarizes some of the most essential ones.

- All basic element shapes in 1D, 2D and 3D with the Lagrange shape functions of degree  $k-2$
- Higher degree approximation using p-elements
- Time integration schemes for the first and second order equations
- Solution methods for eigenvalue problems
- Direct linear system solvers (Lapack & Umfpack)
- Iterative Krylov subspace solvers for linear systems
- Multigrid solvers (GMG and AMG) for some basic equations
- ILU preconditioning of linear systems
- Parallelization of iterative methods
- The discontinuous Galerkin method
- Stabilized finite element formulations, including the methods of residual free bubbles and SUPG
- Adaptivity, particularly in 2D
- BEM solvers (without multipole acceleration)

### 7.7.22 SPECFEM3D

Eric Boyer

CINES France

The software package SPECFEM3D simulates southern California seismic wave propagation based upon the spectral-element method (SEM). Effects due to lateral variations in compressional-wave speed, shear-wave speed, density, a 3D crustal model, topography and bathymetry are included. For a detailed introduction to the SEM as applied to regional seismic wave propagation. The Moho map was determined by Zhu and Kanamori. The 1D soCal model was developed by Dreger and Helmberger. The package can accommodate full 21-parameter anisotropy as well as lateral variations in attenuation. Adjoint capabilities and finite-frequency kernel simulations are included.

All SPECFEM3D software is written in Fortran90, and conforms strictly to the Fortran95 standard. It uses no obsolete or obsolescent features of Fortran77. The package uses parallel programming based upon the Message Passing Interface (MPI).