SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

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

PRACE-2IP

PRACE Second Implementation Project

Grant Agreement Number: RI-283493

D12.3
Development Environment and Tools

Final

Version: 1.0
Author(s): Jose Carlos Sancho, Claudia Rosas, Vladimir Subotic, Jesus Labarta (BSC)
Date: 20.02.2013

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-283493**	
	Project Title: PRACE Second Implementation Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: <D12.3>	
	Deliverable Nature: < Report >	
	Deliverable Level: PU	**Contractual Date of Delivery:** 28 / February / 2013
		Actual Date of Delivery: 28 / February / 2013
	EC Project Officer: Leonardo Flores Añover	

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

Document Control Sheet

Document	**Title: Development Environment and Tools**	
	ID: D12.3	
	Version: <1.0 >	**Status: Final**
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D12.3.docx	
Authorship	**Written by:**	Jose Carlos Sancho, Claudia Rosas, Vladimir Subotic, Jesus Labarta (BSC)
	Contributors:	
	Reviewed by:	Miroslaw Kupczyk, PSNC; Dietmar Erwin, Juelich
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	28/01/2013	Draft	First Draft
1.0	08/02/2013	Final version	

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] J.Dongarra et al., “The international exascale software project roadmap”, International Journal of High Performance Computing Applications, 2011, 25(I), 3-60
[2] J.Abeles et al., “Performance Guide for HPC Applications on IBM Power 775 Systems”, Release 1.0, April 15, 2012, IBM Systems and Technology Group
[3] A.Duran, X.Teruel, R.Ferrer, X.Martorell, E.Ayguade, “Barcelona OpenMP Tasks Suite: A Set of Benchmarks Targeting the Exploitation of Task Parallelism in OpenMP”, Proceeding ICPP '09 Proceedings of the 2009 International Conference on Parallel Processing, p.124-131, IEEE Computer Society Washington, DC, USA
[4] J. Labarta, S. Girona, V. Pillet, L. Gregoris and T. Cortes, "Dip: A parallel program development environment.," in Proceedings of the Second International Euro-Par Conference on Parallel Processing, London, UK, UK, 1996, pp. 665--674.
[5] V. Pillet, J. Labarta, T. Cortes and S. Girona, "PARAVER: A Tool to Visualize and Analyze Parallel Code," in Proceedings of WoTUG-18: Transputer and occam Developments, 1995, pp. 17--31.
[6] S. Girona, J. Labarta and R. M. Badia, "Validation of Dimemas Communication Model for MPI Collective Communications," in Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, London, UK, UK, 2000, pp. 39--46.
[7] V. Subotic, R. Ferrer, J. C. Sancho and J. Labarta, "Quantifying the potential task-based dataflow parallelism in MPI applications," in Proceedings of the 17th International Conference on Parallel Processing, Bordeaux, 2011, pp. 39--51.
[8] http://www.prace-ri.eu/white-papers

List of Acronyms and Abbreviations

CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CUDA	Compute Unified Device Architecture (NVIDIA)
DP	Double Precision, usually 64-bit floating point numbers
DRAM	Dynamic Random Access memory
EC	European Community
EPCC	Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
EPSRC	The Engineering and Physical Sciences Research Council (United Kingdom)
ETHZ	Eidgenössische Technische Hochschule Zuerich, ETH Zurich (Switzerland)

FP	Floating-Point
FPGA	Field Programmable Gate Array
FPU	FZJ Forschungszentrum Jülich (Germany)
GB	Giga (= 2^{30} ~ 10^9) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^9) bits per second, also Gbit/s
GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GCS	Gauss Centre for Supercomputing (Germany)
GÉANT	Collaboration between National Research and Education Networks to build a multi-gigabit pan-European network, managed by DANTE. GÉANT2 is the follow-up as of 2004.
GENCI	Grand Equipement National de Calcul Intensif (France)
GFlop/s	Giga (= 10^9) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GPGPU	General Purpose GPU
GPU	Graphic Processing Unit
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
IDRIS	Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
ISC	International Supercomputing Conference; European equivalent to the US based SC0x conference. Held annually in Germany.
JSC	Jülich Supercomputing Centre (FZJ, Germany)
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
KTH	Kungliga Tekniska Högskolan (represented in PRACE by SNIC, Sweden)
LINPACK	Software library for Linear Algebra
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
Mop/s	Mega (= 10^6) operations per second (usually integer or logic operations)
MoU	Memorandum of Understanding.
MPI	Message Passing Interface
NCF	Netherlands Computing Facilities (Netherlands)
NDA	Non-Disclosure Agreement. Typically signed between vendors and customers working together on products prior to their general availability or announcement.
OpenCL	Open Computing Language
OpenGL	Open Graphic Library
Open MP	Open Multi-Processing

PGAS	Partitioned Global Address Space
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PSNC	Poznan Supercomputing and Networking Centre (Poland)
SGI	Silicon Graphics, Inc.
SHMEM	Share Memory access library (Cray)
SIMD	Single Instruction Multiple Data
SMP	Symmetric MultiProcessing
SNIC	Swedish National Infrastructure for Computing (Sweden)
SP	Single Precision, usually 32-bit floating point numbers
SPE	Synergistic Processing Element (core of Cell processor)
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
TB	Tera (= 240 ~ 1012) Bytes (= 8 bits), also TByte
TFlop/s	Tera (= 1012) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
UPC	Unified Parallel C

Executive Summary

Work Package 12 (WP12) "Novel Programming Techniques" performs research and development in four key areas for future multi-petascale and exascale systems, auto-tuned runtimes (Task 12.1), scalable numerical algorithms (Task 12.2), development environment and tools (Task 12.3) and file system optimization (Task 12.4).

Specifically, in this deliverable we are reporting the progress made on development environments and tools. This task effort has been distributed among four different projects each of which is focused on a different programming model or programming technique.

- Task-based programming model, OmpSs
- Simultaneous multithreading technique, SMT
- Programming standard for accelerators, OpenACC
- Partitioned global address space, PGAS

These projects address or suggest different tools specifically to optimize codes in each particular programming model or technique. This deliverable provides a summary of the results and findings achieved by the different projects. It is covering work from the start of WP12 up to month eighteen. Along with the deliverables the researchers also produced whitepapers for each of the projects. More detailed results and descriptions for each project can be found in these whitepapers. They would be useful for readers that are interested in more detailed information.

1 Introduction

The objective of this task is to develop technologies and tools that will help application programmers to accelerate their codes on multicore processors and heterogeneous architectures.

In the last years, we have been witnesses of a new paradigm shift in computer systems driven by the emergence of multicore processors and heterogeneous architectures. This hardware paradigm shift is disrupting any traditional software development tools which were basically conceived to optimize traditional single processor architectures. As a consequence, there is currently an urgent need to adapt software design to these new architectures. New parallel programming languages, libraries, tools and compilers became necessary to design and implement applications able to efficiently harness the computational potential of these architectures.

The large variety of codes that are used today in different areas of science and industry, where each of them is behaving differently, obviously indicates that there will not be a single solution that optimizes all the codes at once. For this reason, we are exploring and investigating the efficiency of current new proposed techniques and programming languages to optimize these codes.

The research activities conducted are being focused on four different techniques to refactoring codes. The first one is focused on refactoring codes to task-based parallel programming languages such as OmpSs. The second one uses the technique of simultaneous multithreading technique on current multicores processors. The third one is dedicated to the new accelerator programming standard OpenACC and optimizes codes on heterogeneous architectures based

on GPUs. And the last one, it is focused on PGAS programming languages that exploits the locality of codes.

A brief summary of the key research accomplishments on these four tasks are described below.

- Task-based programming model, OmpSs. It is shown how the performance analysis and visualization tools developed at Barcelona Supercomputing Center can help programmers to optimize existent task-based applications. Results reported on the Jacobi method solver show a substantial improvement of 80% in comparison with its corresponding serial version on multicore processors.
- Simultaneous multithreading technique, SMT. This project is investigating the optimization of codes using different simultaneous multithreading (SMT) modes currently available on most multicore processors. Predicting which mode achieves the best performance for a particular application is the ultimate goal. Results show a big potential of this technique achieving up to 2X performance improvement on some codes.
- Partitioned global address space, PGAS. This project is experimenting with the PGAS parallel programming language X10. The HYDRO code and the TRITON software are being ported to X10. Results shows that measured compute times show fair scalability. However, there is still a need to develop efficient tools to better port these codes to X10.
- Programming standard for accelerators, OpenACC. In this project, OpenACC's loop scheduling was varied to explore whether automatic compiler behaviour could be improved through the use of manual scheduling clauses. Results showed a factor of 3.5X performance improvement over the traditional automatic scheduling.

The deliverable itself is quite concise in order to allow readers to easily identify the projects that are of particular interest for them and to encourage further reading in the accompanying white papers. The white papers can be found on the PRACE web [8].

In this work package we have not yet established a collaboration with WP11 dealing with prototyping. It might be due to that our work was mostly focused on new programming languages on existing computers rather than exploring these developments on new machines as in WP11. On the other hand, we have established a successful collaboration with WP8 Community Code Scaling. There is a common agreement that it would be beneficial to industry to show the last developments in the area of software such as the ones explored in this work package on development environments and tools.

The rest of the deliverable is organized as follows. Chapter 2 provides a description of the organization of Task 12.3. Brief discussions on the main research result achieved as well as brief overviews of the projects are given in Chapter 3. And finally, Chapter 4 presents the conclusions.

2 Task Organization

Figure 1 shows the different projects and their main targeted programming language or technique. These four projects are being carried out by four different partners located in four different countries in Europe as can be seen in Table 1. BSC was the leader of this task and was fully involved in organization the work and monitoring the progress of the different partners over the duration of this task.

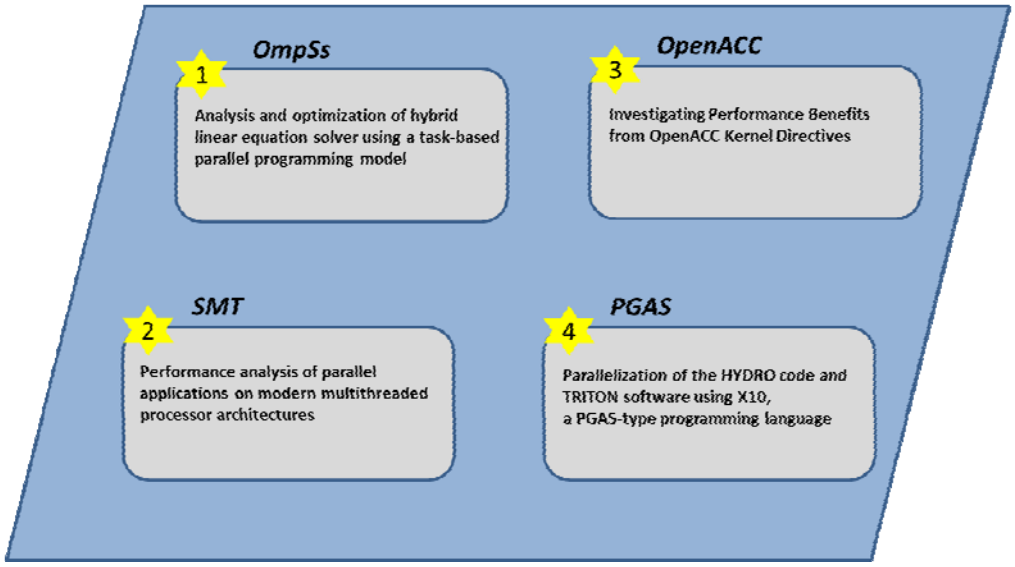

Figure 1: Overview of the target programming model for the different tasks

Project	Partner	Country	PMs
OmpSs	BSC	Spain	22
SMT	PSNC	Poland	5
PGAs	GENCI	France	10
OpenACC	ICHEC	Ireland	5
Total			42

Table 1: Overview of efforts per partner

Furthermore, a wiki page has been used for all partners to facilitate the organization and monitoring of the activities carried out in this task.

3 Project Overviews

An overview and some relevant results of the four projects that are being carried on task 12.3 are giving in the following four sections.

3.1 Analysis and optimization of hybrid linear equation solver using a task-based parallel programming model

Supported by: C. Rosas, V. Subotic, J.C. Sancho (BSC)

Whitepaper: C. Rosas, V. Subotic, J.C. Sancho. "Analysis and optimization of hybrid linear equation solver using a task-based parallel programming model", PRACE technical white paper.

In the last years, the emergence of multicore processors led to an urgent need to adapt software design. Applications must efficiently exploit the availability of parallelism that large amount of cores are providing. In consequence, new parallel programming languages, libraries, tools and compilers became necessary to design and implement applications able to take advantage of the potential of parallelism. Dataflow programming models based on tasks have been developed to harness the potential of multicore architecture, e.g. MPI/OpenMP and MPI/OmpSs[1]. They combine distributed and shared memory concepts into hybrid codes.

Presently, the user/programmer is fully involved in the process of exploiting manually the potential parallelism of the application. This process is highly inefficient and time consuming and might prevent the programmer to port existent applications to a task-based parallel programming model. There is no guarantee if the resulting performance improvement is worth the programming effort.

In this project, it is shown how the performance analysis and visualization tools developed at Barcelona Supercomputing Center can help programmers to optimize existent task-based applications. To show the usability of these tools for the optimization process an easy methodology has been designed and applied in a hybrid implementation of a linear equation solver based on the Jacobi's method. The methodology consists of three major steps: (i) performance analysis; (ii) prediction; and (iii) implementation, which goes through a loop to reach a more efficient implementation in each step. First, an analysis of the application must be done in order to find possible performance issues and get some insights about the implementation and its overall performance. Second, main proposal of performance improvements are evaluated before modifying the original code. Finally, once the main issues are reported and potential code optimizations are identified, e.g. defining an optimal taskification[2], users may proceed to implement and evaluate proposed solutions in a real production machine.

The hybrid implementation of Jacobi's method provided by EPCC combines MPI and OpenMP programming models. Here, asynchronous communication and computation are used to reduce undesirable synchronizations that might not scale as expected on the upcoming exascale machines.

To analyze these applications, tools such as Extrae [4], Paraver [5], Dimemas [6], and Tareador [7] were used, and the OmpSs programming model enabled some optimizations in the existent code. Extrae is based on mpitrace tracing library to instrument parallel executions. It intercepts calls to certain functions and records the events that mark these occurrences. Paraver is a parallel program visualization and analysis tool. It provides a qualitative perception of the application's time-behaviour by visual inspection and a

[1] OmpSs is an effort to extend OpenMP to support asynchronous parallelism and heterogeneity.
[2] Divide the original code into smaller and different tasks, which can be executed in parallel

quantitative analysis of the run. Dimemas is an open-source trace file based simulator for analysis of message-passing applications on a configurable parallel platform. It reconstructs the time behaviour of a parallel application on a machine modelled by a set of performance parameters. Tareador allows the programmer to start from a sequential application, and using a set of simple constructs proposes some decomposition of the sequential code into tasks. It dynamically instruments the annotated code and at run-time detects actual data-dependencies among the proposed tasks. The OmpSs is based on the OpenMP programming model, with some modifications to its execution and memory model in order to support asynchronous parallelism and heterogeneous devices. OmpSs execution model is a thread-pool model instead of the traditional OpenMP fork-join. The master thread starts the execution and other threads cooperate executing the work it creates, therefore, there is no need for a parallel region.

The test bed where the Jacobi implementations were running is called MinoTauro and is hosted by BSC. It consists of 126 compute nodes and 2 login nodes. Every node has two processors Intel Xeon E5649 6-Core at 2, 53 GHz running Linux operating system with 24 GB of RAM memory, 12MB of cache memory and 250 GB local disk storage.

Results from executing the original synchronous version of Jacobi annotated with OmpSs directives are reported below. The number of cores was increased from 1 to 12. In particular, Figure 2 shows the Paraver views for one thread and two threads. As you can see by doubling the number of threads a reduction in total execution time is achieved. With 12 threads, it greatly reduces total execution time in up to an 80%, in comparison with serial version (shown in Figure 3). Extensive experimentation using MPI/OmpSs, and final comparisons with original MPI/OpenMP implementations, are still in development and have to be reported soon.

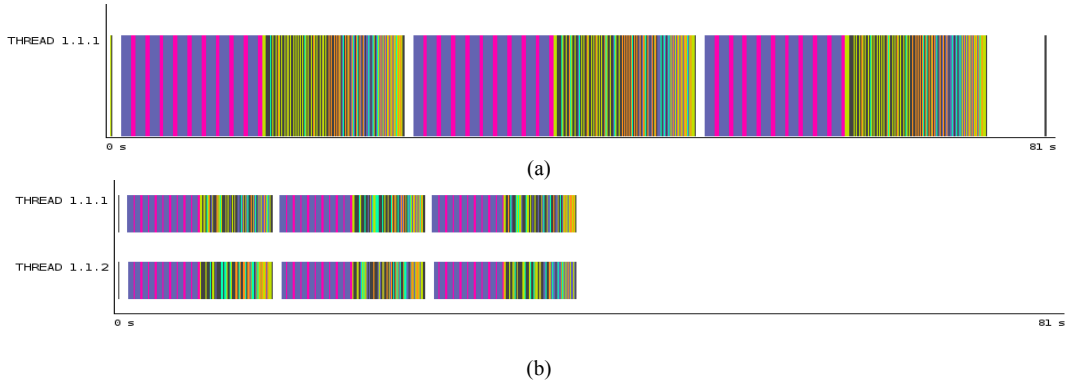

Figure 2: Traces for user functions in the implementation of the Jacobi method using OmpSs directives, problem size equal to 50, and number of tasks inside each thread equal to 4x4x4. (a) 1 thread; and (b) 2 threads

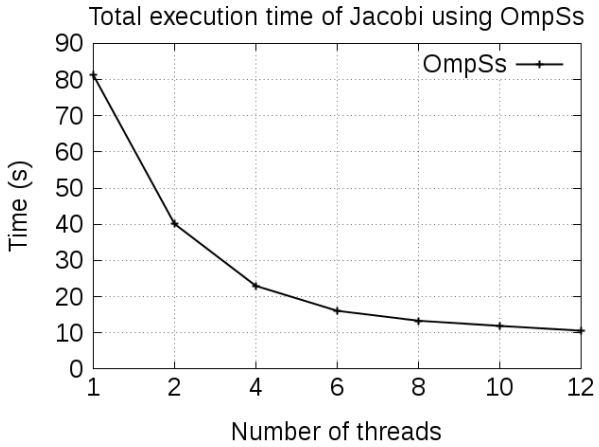

Figure 3: Total execution time in seconds when increasing the number of thread. Implementation of the Jacobi method using OmpSs directives, problem size equal to 50, and number of tasks inside each thread equal to 4x4x4

3.2 Performance analysis of parallel applications on modern multithreaded processor architectures

Supported by: M.Cytowski, M.Filocha, J.Katarzyński, M.Szpindler (PSNC)

Whitepaper: M.Cytowski, M.Filocha, J.Katarzyński, M.Szpindler. "Performance analysis of parallel applications on modern multithreaded processor architectures", PRACE technical white paper.

Introduction

Performance of today's general purpose processor architectures is driven by three main components: clock speed, number of computational cores and number of double precision operations per cycle. The combination of those three is widely used as a basic measure of processors performance known as FLOPs – number of floating point operations per second. Since further increasing clock speed and core count is technologically still very difficult, hardware vendors continue to develop different ways to increasing single core applications performance i.e. vector processing units, support for fused multiply and add operations and hardware support for simultaneous processing of multiple threads, so-called multithreading. One of the most appropriate ways to measure real performance of a given processor architecture is to measure its efficiency when used for selected classes of scientific algorithms and applications.

We decided to measure the performance of applications and synthetic benchmarks using different simultaneous multithreading (SMT) modes. This specific processor architecture feature is currently available in many petascale HPC systems. Both IBM Power7 processors available in Power775 (IH) and IBM Power A2 processors available in Blue Gene/Q are built upon 4-way simultaneous multithreaded cores. It should be also mentioned that multithreading is predicted to be one of the leading features of future exascale systems available by the end of the next decade [1].

This work was motivated by results presented in [2] which show that the performance gain from SMT varies depending on the program execution and its execution model, the threading mode used on the processor, and the resource utilization of the program. The gains from using SMT modes with selected algorithms where measured using well known benchmarks: SPEC CFP2006, NAS Parallel Benchmark Class B (OpenMP), and NAS Parallel Benchmark Class C (MPI). One of the conclusions of the study presented in [2] was that throughput type

workloads are best suited to see gains from using higher SMT modes. On the other hand, codes with high memory traffic will most likely not perform well when executed in SMT2 or SMT4 mode.

Parallel applications can use the SMT modes executing with numbers of processes and/or threads that exceed the physical number of cores available in the system. This may be achieved by executing an application with 2x (SMT2) or 4x (SMT4) more MPI processes or by executing an OpenMP/Pthreads code with 2x or 4x more threads or by mixing those two MPI and multithread execution modes (e.g. in the case of hybrid MPI/OpenMP codes).

Our results show that the SMT mechanism available in modern processor chips can be efficiently used to increase performance of applications and algorithms. On the other hand there exists a class of algorithms and applications that does not benefit from multithreading. In-depth investigation of the reasons of such divergence is planned as future work.

Benchmarks description and computational environment

We have measured the performance of selected scientific applications with different SMT modes (GADGET2, WRF, CPMD, GROMACS, GPAW). The benchmarks where executed on a Power750 system with two Power7 processors (each with 8 cores) operating at 3.5 GHz with total of 128 GB RAM. We have also analyzed in detail the performance of one application – GADGET2. We have measured the performance of different algorithms used in GADGET2 code. Moreover we have decided to measure the performance of scientific algorithms available in the BOTS benchmark [3]. The BOTS benchmark was executed on computational nodes of the PRACE Tier-1 Boreasz system available at ICM, University of Warsaw. Boreasz is IBM Power775 (IH) supercomputer whose computational nodes (called octants) are composed of four Power7 processors (with 8 cores each) operating at 3.8 GHz with total of 128 GB RAM. This gave us additional information on the type of algorithms whose performance might benefit from SMT mechanism.

Results

All results of our work are described in the whitepaper „ Performance analysis of parallel applications on modern multithreaded processor architectures" available on-line at www.prace-ri.eu. Here we present the most interesting case of the GADGET-2 application. Figure 4 shows the performance of the GADGET2 code measured against different SMT modes. The test case of GADGET2 consisted of almost 28 million particles. As we can see GADGET2 is an example of an application which benefits from using the SMT mechanism. The best wall clock time results are always achieved for SMT4 mode regardless of number of cores in use.

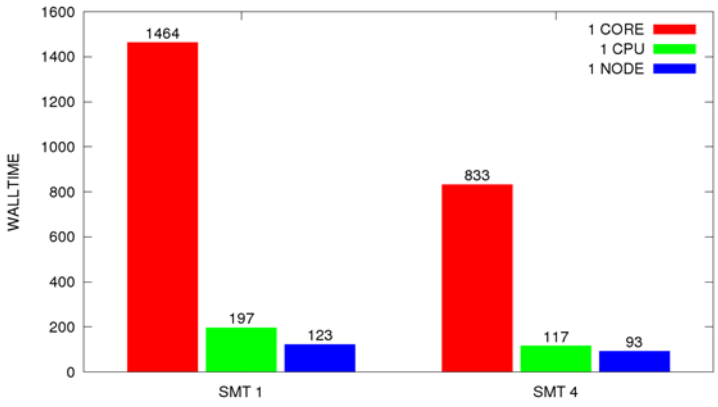

Figure 4: Performance of GADGET2

We decided to have a closer look at the performance of the GADGET2 code. This work was motivated by the fact that GADGET2 benefits from using higher SMT modes but also by the fact that it implements and uses algorithms of different computational nature: tree code, FFT, particle computations.

	Single cpu			Single node (2 cpus)		
Algorithm	**SMT 1**	**SMT 2**	**SMT 4**	**SMT 1**	**SMT 2**	**SMT 4**
Tree walk	51,64	33,57	25,23	26,73	18,49	13,07
SPH	81,28	56,26	42,79	42,2	31,4	22,42
Particle-Mesh	40,39	34,15	33,53	25,94	31,84	26,66

Table 2: Performance of codes on different SMT configurations [sec.]

Measurements obtained for different algorithms used in the GADGET2 code (presented in the table above) demonstrate that the SMT mechanism is not always the key for better performance. Especially the Particle-Mesh algorithm which extensively uses FFT computations does not benefit from SMT 2 and SMT 4 at all. However, both Tree walk and Smoothed Particle Hydrodynamics steps present very good performance when executed in higher SMT modes.

Future work

The results we obtained motivated us to plan future work related to a performance study of algorithms and applications on multithreaded and multicore architectures.

Firstly, we are very keen to know why the performance of the chosen algorithms can benefit from using higher SMT modes while others do not. We believe that this problem might be addressed by a detailed analysis of hardware performance counters. Currently we are analyzing the hardware performance counters for the chosen applications from the BOTS benchmark. We are looking for a correlation between the ability to efficiently use the SMT mechanism and the performance profile of the given application. Such a result would lead us to better understanding of the computational nature of different algorithms, but it could also be used to propose an automatic heuristic algorithm (e.g. based on decision trees) to decide which multithreaded code fragments should be using SMT2 or SMT4 mode.

Secondly, many modern HPC applications use both MPI and thread parallel model (e.g. mixed MPI + OpenMP). Parallel processes executed on different computational nodes include many thread parallel regions which are executed on the available computational cores. Very often the number of threads in thread based parallelization is controlled by a single switch (e.g. the OMP_NUM_THREADS environment variable). Since different algorithms and code fragments may present different scalability on a given HPC platform then it would be appropriate to choose the number of threads for execution to each parallel region individually. Moreover such a decision could be made automatically only with a minor information gathered from user (mainly the information about the preferred execution model of the code).

The tool that we are currently developing within PRACE-2IP project will address both above mentioned challenges.

3.3 Parallelization of the HYDRO code and TRITON software using X10, a PGAS-type programming language

Supported by: Marc Tajchman (CEA-DEN)

Whitepaper: Marc Tajchman, "Parallelization Using a PGAS Language such as X10 in HYDRO and TRITON Software", PRACE technical white paper.

The HYDRO code and TRITON software are intended for 2D (Hydro) and 3D (Triton) simulations in hydro-dynamics. They both operate on 2D or 3D meshes.

In several other contributions, implementations of Hydro have been presented and evaluated using standard programming languages (C/C++/Fortran) and parallel paradigms (MPI, OpenMP), as well as less-standard ones (CUDA, UPC, Chapel). The sequential version of Hydro we used consisted of 1000+ lines of C code.

Triton is an in-house software platform developed at CEA-DEN, with a sequential implementation in C, and parallel implementations using MPI or CUDA. Triton can be used to test several fluid models and versions of the numerical flux computation. The sequential version of Triton has 10000+ lines of C code.

X10 is a parallel programming language of PGAS type. The X10 execution model is different from the MPI model (SPMD model). A code is launched on several processes (Places in X10), but only the main process (Place 0) starts. The main process can then start activities (execution threads) locally (i.e. on the same place) or remotely (on different places). So, a multi-threads code (in the common sense) is defined by the developer as a single place that launches several (local) activities. A multi-processes code (in the common sense) is simulated by starting activities from place 0 to wake up other places. Any combination of these examples can be implemented to model hybrid programming, in a very flexible way. Activities are synchronous or asynchronous, and X10 defines a set of synchronization to monitor these activities.

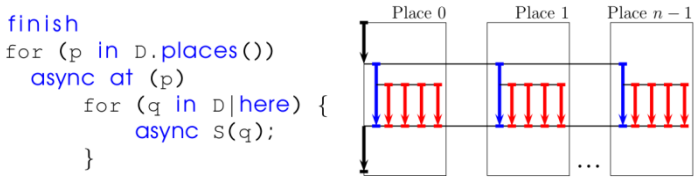

Figure 5: Hybrid programming: on each place, one activity is (remotely) launched and starts a set of local sub-activities

The second main feature of X10 is the possibility to instantiate distributed arrays globally. The developer first defines the global domain, specifies a distribution scheme (which part of the array will be located on each place), and finally, reserves the coefficients. Read/write access to coefficients is carried out in a standard way if the coefficient is located on the place where access is requested. Access to a remotely located coefficient is done by launching a remote activity. Vectorized access to several coefficients is possible if the requested coefficients are located on the same place, and is recommended for efficiency reasons.

X10 is object-oriented, with a grammar similar to Java or C++, this characterization was used particularly during development of X10 version of Triton.

All developments were carried out starting from the sequential C versions. We made this decision to be as little as possible influenced by parallel paradigms (e.g. MPI or OpenMP) used in existing versions.

For Hydro, we used the particular properties of the numerical scheme: at each time-iteration, the scheme is divided in 3 groups of independent 1D computations. Each group is separated by a redistribution of the coefficients between the places. This implementation works well on a small set of places (processes), but we do not expect this procedure to be scalable.

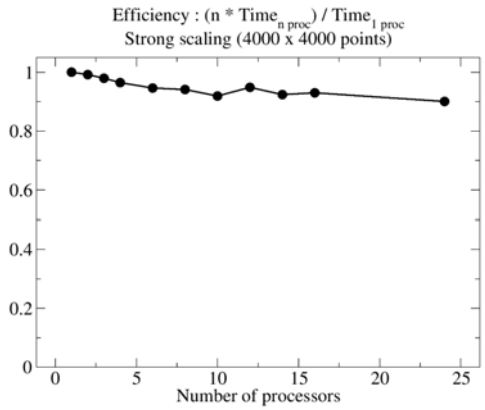

Figure 6: Strong scaling behavior with Hydro

For Triton, we implemented a sub-domains division, adding a halo of supplementary cells around each sub-domain (this is a standard implementation for this kind of codes). The halo allows to group transfers between places. Object-orientation of X10 is used to simplify the scheme implementation. A working implementation has been written and checked against sequential versions.

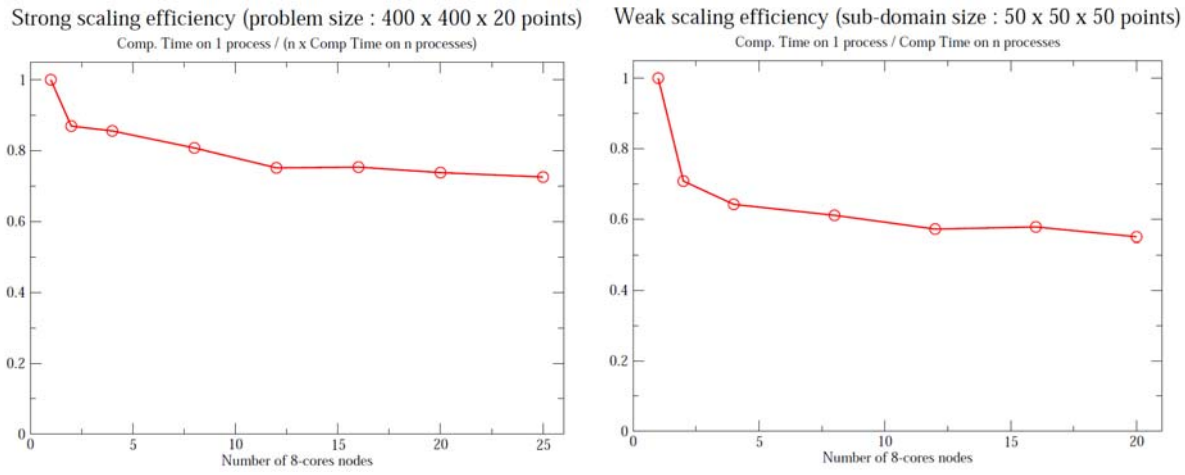

Figure 7: Strong and Weak scaling tests with Triton

The efficiency gap between 1 and 2-nodes runs suggests that our implementation needs to be better optimized.

We list here what we have learned during the development of X10 code versions:

- The easiest and fastest way to build a code using a PGAS language is to start from a sequential version. The structure of the PGAS code versions is very similar to the original sequential version, especially if we use “transparent” accesses to remote data. But the last feature may lead to a very inefficient code, with many of hidden movements of small packets of data between processors. See next item of this list.
- To obtain more performance, we must pack as much as possible the data exchanges between two places (processes). In practice, the developer must maintain buffers for data exchanges so that the code will not be as simple as we can expect.
- The X10 language (the same characteristics apply to Chapel, another PGAS language) offers a very fine control on the parallel tasks and data distributions. The ability to start threads in the same process or in a remote process facilitates the conception of the control flow during the execution. Other languages, Co-Array Fortran, UPC, or XcalableMP are less versatile, but also easier to apprehend for “PGAS programming style” beginners.
- A very interesting aspect of PGAS languages is that there is much less possibilities of “dead locks”. The PGAS runtime will take in charge the “details” of organizing the communications.
- Still, the developer has to verify that there are no data races, e.g. by correctly using synchronization mechanisms provided by the language. In the codes we consider here, control flow is simple; we encounter no difficulties with data races. This may be more challenging for more complex codes.

The measured compute times shows fair scalability behaviour (on the limited sets of nodes) but remains far behind MPI/OpenMP versions. There are probably multiple reasons for that, but, at least, our X10 source needs to be optimized and compiled properly (without assertions and checks). Also, future versions of the X10 compiler must and will probably generate more efficient intermediate C++ code.

The main advantages of PGAS languages for developers are the ease to implement task and data parallelism paradigms, the flexibility of the remote data accesses and computations. The main drawback is the performances of the binaries generated by current PGAS tools. It remains us to check the scalability of our developments on larger sets of nodes.

3.4 Investigating Performance Benefits from OpenACC Kernel Directives

Supported by: Gilles Civarioa, Benjamin Eagan (ICHEC)

Whitepaper: Gilles Civarioa, Benjamin Eagan. "Investigating Performance Benefits from OpenACC Kernel Directives", PRACE technical white paper.

OpenACC is a standardized programming language extension for C and Fortran with support from PGI, CAPS, NVIDIA, and Cray. This is used to program heterogeneous systems by placing directives to indicate which regions to offload to the acceleration device. Code is then generated automatically for these regions. In this report, OpenACC's loop scheduling was varied to explore if automatic compiler behaviour could be improved through the use of manual scheduling clauses.

Manual scheduling was achieved by providing gang and vector clauses to each loop directive, along with the corresponding size for each value. The gang value determines the course-grained parallelism for each loop, similar to the thread blocks in CUDA terminology, and the vector clause dictates the fine-grained parallelism, specifying the number of threads with access to shared memory within a gang.

Experiments were carried out using a naïve matrix-matrix multiplication code, as well as a Classical Gram-Schmidt orthonormalisation. The PGI and CAPS implementations of OpenACC were both explored for these examples, with Intel's compiler handling the CPU regions for the CAPS compilations. Experiments were run using a 2.8 Xeon GHz X5560 and an NVIDIA Tesla M2090. Further testing was carried out using an Intel Xeon 3.30 GHz E5-2643 with an NVIDIA K20 GPU.

On the M2090 test platform, a systematic sweep of gang and vector values was performed to identify the performance properties of various scheduling combinations for the matrix-matrix multiply. It was seen that the use of manual scheduling improved the runtime over the automatic scheduling by a factor of 1.7 and 3.1 for the PGI and CAPS compilers respectively. The results from these tests can be seen in Figure 8 and Figure 9.

The matrix-matrix multiply was then run on an Intel Xeon 3.30 GHz E5-2643 with an NVIDIA K20 GPU. It was seen that the ideal gang and vector values from the NVIDIA M2090 were not the ideal values for the K20. The optimal runtime was achieved by doubling the vector size to match the architecture.

Finally, experiments were carried out using the PGI and CAPS OpenACC implementations to parallelise a Classical Gram-Schmidt orthonormalisation code. This program involved multiple loops of varying complexities instead of a single loop as seen with the matrix-matrix multiply. In this case, the PGI implementation was seen to have the ideal runtime with automatic scheduling. The CAPS implementation however benefited from manual scheduling, with a factor of 3.5 performance improvement over the automatic scheduling. The results from these tests can be seen in Figure 10 and Figure 11.

Overall it was seen that OpenACC's loop scheduling clauses had a dramatic impact on performance. In the matrix-matrix multiplication, the addition of these tuning clauses resulted in a significant improvement for both compilers. In the Gram-Schmidt orthonormalisation, the automatic behaviour was ideal for PGI, however CAPS benefited from manual tuning.

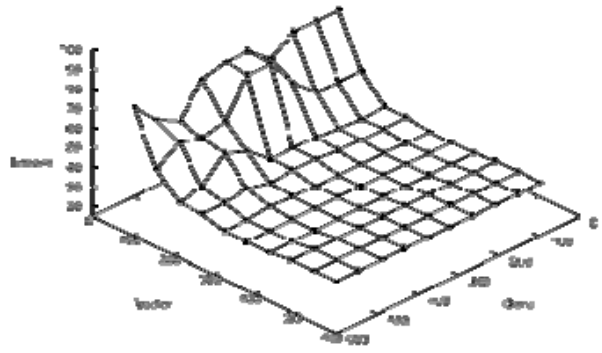

Figure 8: Runtimes for a matrix-matrix multiplication with various gang and vector sizes compiled using PGI and run on an NVIDIA M2090 GPU

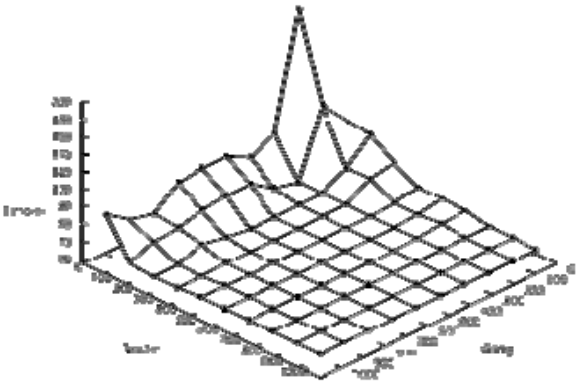

Figure 9: Runtimes for a matrix-matrix multiplication with various gang and vector sizes compiled using CAPS run on an NVIDIA M2090 GPU

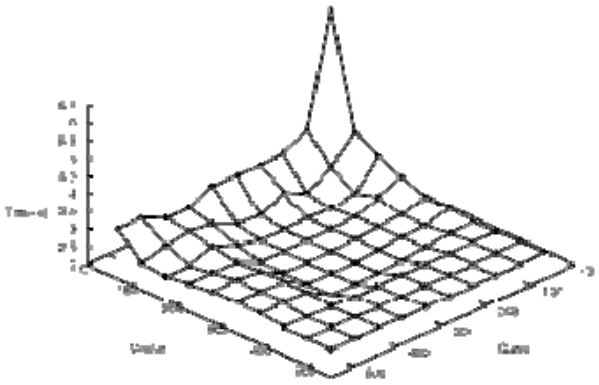

Figure 10: Runtimes for a classic Gram-Schmidt orthonormalisation with various gang and vector sizes compiled using PGI run on an NVIDIA M2090 GPU GPU

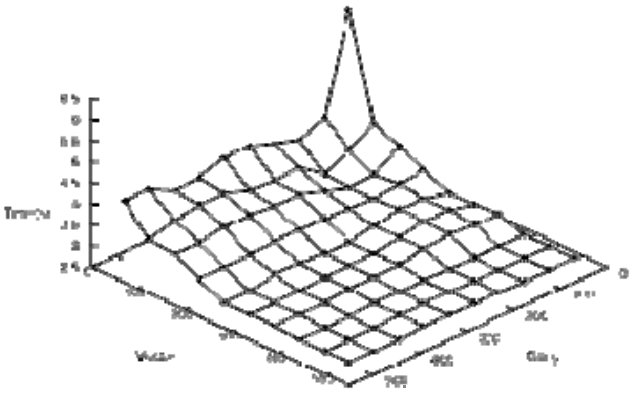

Figure 11: Runtimes for a classic Gram-Schmidt orthonormalisation with various gang and vector sizes compiled using, CAPS run on an NVIDIA M2090 GPU

4 Summary and Conclusions

The main objective of this task is to help programmers refactoring existing codes to new parallel programming languages or techniques in order to efficiently harness current multiprocessor and heterogeneous architectures. This is a research activity exploring techniques and suggesting tools to efficiently develop, port, and debug existing codes.

Four different individual projects carried out work on this task. These four projects are focused on a different programming language or technique.

- Task-based programming model, OmpSs
- Simultaneous multithreading technique, SMT
- Programming standard for accelerators, OpenACC
- Partitioned global address space, PGAS

A variety of state-of-the-art parallel machines has been used on the experimental evaluation such as multicore processors IBM Power7 and Intel Xeon, and NVIDIA Tesla GPUS. Moreover, the result of this task achieved optimized versions of a variety of codes such the Jacobi method, Classical Gram-Schmidt orthonormalisation code, matrix-multiplication code, HYDRO, TRITON, GADGET2, WRF, CPMD, GROMACS, and GPAW.

The following summarize the experiences gained from the different projects.

In order to achieve high performance on task-based programming models it is critical to select an optimal taskification of the code that maximizes the parallelism. Using new tools specifically devoted for this purpose is largely helping programmers to optimize their codes. Results showed 80% performance improvement for the Jacobi method using the optimal taskification.

For the simultaneous multithreading technique the complexity lays on finding the best SMT mode for a particular code. The key solution proposed in this work is to develop a tool that determines the optimal SMT mode based on a detailed analysis of hardware performance counters. Results show a big potential of these technique achieving up to 2X performance improvement on some codes.

Experiments with the PGAS parallel programming language X10 on the HYDRO code and TRITON software shows that tools are needed to debug data race conditions. Moreover, tools to manage accesses to remote data are needed in order to prevent inefficient hidden movements of small packets of data between processors.

And finally, when using OpenACC a key issue to help programmers optimize their code is to assist selecting an optimal loop scheduling. Results showed a factor of 3.5X performance improvement when managing manually the loop scheduling over the traditional automatic scheduling.