SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

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

# PRACE-1IP

## PRACE First Implementation Project

Grant Agreement Number: RI-261557

# D9.2.1
# First Report on Multi-Petascale to Exascale Software

*Final*

Version: 1.1

Author(s): Volker Strumpen, JKU
Iris Christadler, LRZ
Guillaume Colin de Verdiere, CEA
Matthieu Hautreux, CEA

Date: 21.04.2011

## Project and Deliverable Information Sheet

| | | |
|---|---|---|
| **PRACE Project** | **Project Ref. №: RI-261557** | |
| | **Project Title: PRACE First Implementation Project** | |
| | **Project Web Site:** http://www.prace-project.eu | |
| | **Deliverable ID: D9.2.121** | |
| | **Deliverable Nature:** <DOC_TYPE: Report / Other> | |
| | **Deliverable Level:** PU / PP / RE / CO * | **Contractual Date of Delivery:** 30 / April / 2011 |
| | | **Actual Date of Delivery:** 30 / April / 2011 |
| | **EC Project Officer: Bernhard Fabianek** | |

* - 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:** First Report on Multi-Petascale to Exascale Software | |
| | **ID:** <**D9.2.1**> | |
| | **Version:** <1.1 > | **Status:** Final |
| | **Available at:** http://www.prace-project.eu | |
| | **Software Tool:** Microsoft Word 2007 | |
| | **File(s):** D9.2.1.docx | |
| | **Written by:** | Volker Strumpen, JKU |

| | | |
|---|---|---|
| **Authorship** | **Contributors:** | **Computer Systems:** Riccardo Brunino (CINECA), Carlo Cavazzoni (CINECA), Willy Homberg (FZJ), Herbert Huber (LRZ), Radek Januszewski (PSNC), Jochen Kreutz (FZJ), Jacques-Charles Lafoucriere (CEA), , Stephan Michael (FZJ), Stefan Riha (JKU), Alam Sadaf Roohi (CSCS), Ramnath Sai Sagar (BSC), Michael Schliephake (KTH) **Programming Languages:** Ricardo Brunino (CINECA), Iris Christadler (LRZ), Tiziano Diamanti (CINECA), Okan Dogru (UYBHM), Federico Ficarelli (CINECA), Ivan Girotto (ICHEC), Jose Gracia (HLRS), Giannis Koutsou (CaSToRC), Agnieszka Kwiecien (PSNC/WCNS), Pierre-François Lavallée (IDRIS), Ioannis Liabotis (GRNET), Murat Manguoglu (UYBHM), Martin Polak (JKU), Mariusz Uchronski (PSNC/WCNS), Alam Sadaf Roohi (CSCS), Ramnath Sai Sagar (BSC), Philippe Wautelet (IDRIS), Volker Weinberg (LRZ) **System Software:** Axel Auweter (LRZ), Daniela Galetti (CINECA), Matthieu Hautreux (CEA), Herbert Huber (LRZ), Can Ozturan (UYBHM), Alam Sadaf Roohi (CSCS), Andrea Vanni (CINECA) |
| | **Reviewed by:** | Tim Robinson (CSCS), Dietmar Erwin (JSC) |
| | **Approved by:** | MB/TB |

## Document Status Sheet

| **Version** | **Date** | **Status** | **Comments** |
|---|---|---|---|
| 1.0 | 11/April/2011 | Draft version | |
| 1.1 | 21/April/2011 | Final version | |

## Document Keywords

| | |
|---|---|
| **Keywords:** | PRACE, HPC, Research Infrastructure |

# Table of Contents

# List of Figures

# List of Tables

# References and Applicable Documents

[1] P. Andrews, P. Kovatch, V. Hazlewood, T.Baer, *Scheduling a 100,000 Core Supercomputer for Maximum Utilization and Capability*, 39th International Conference on Parallel Processing Workshops (ICPPW), 2010.

[2] Berkley UPC project http://upc.lbl.gov.

[3] F. Cappello, Resilience: *One of the main challenges for Exascale Computing*, EESI 2010.

[4] M. Clark, R. Babich and B.Joo, *Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice Quantum Chromodynamics*, arXiv:1011.0024[hep-lat].

[5] M. Clark, R. Babich, K. Barros, R. Brower, and C. Rebbi, *Solving Lattice QCD systems of equations using mixed precision solvers on GPUs*, Comp. Phys. Comm. 181, 1517 (2010).

[6] Co-Array Fortran, http://www.co-array.org/

[7] Condor High Throughput Computing, http://www.cs.wisc.edu/condor/

[8] Cray Compiler Environment for CAF and UPC (details available at: http://docs.cray.com/).

[9] CUDA zone, http://www.NVIDIA.com/object/cuda_home_new.html

[10] Jack Dongarra et al., *International Exascale Software Project Roadmap*, work in progress, www.exascale.org/iesp

[11] M. Fatica, *Accelerating LINPACK with CUDA on heterogeneous clusters*. GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (New York, NY, USA), ACM, 2009, pp. 46–51.

[12] Faq on Fault-tolerance for parallel MPI jobs, http://www.open-mpi.org/faq/?category=ft

[13] Y. Georgiou, *Resource and Job Management in High Performance Computing*, PhD Thesis, Joseph Fourier University, France, 2010.

[14] GASNet communication system http://gasnet.cs.berkeley.edu/

[15] General-Purpose Computation on Graphics Hardware, http://gpgpu.org/

[16] NVIDIA, *High Performance Computing - Supercomputing with Tesla GPUs*, http://www.NVIDIA.com/object/tesla_computing_solutions.html

[17] http://computing.ornl.gov/HMC/documents/HMC_AppsBreakout_day2.pdf

[18] http://polaris.cs.uiuc.edu/hta/

[19] http://software.intel.com/en-us/articles/intel-cilk-plus

[20] http://www.anandtech.com/show/4023/the-brazos-performance-preview-amd-e350-benchmarked

[21] http://www.conveycomputer.com/, Convey Computer

[22] http://www.heise.de/newsticker/meldung/SC-2010-IBM-zeigt-BlueGene-Q-mit-17-Kernen-1138226.html

[23] http://www.hpcwire.com/features/Argonne-Orders-10-Petaflop-Blue-GeneQ-Super-115593779.html

[24] http://www.hpcwire.com/features/Lawrence-Livermore-Prepares-for-20-Petaflop-Blue-GeneQ-38948594.html

[25] http://www.intel.com/technology/architecture-silicon/index.htm

[26] http://www.khronos.org/registry/cl/

[27] http://www.maxeler.com/, Maxeler Technologies

[28] http://www.prace-project.eu

[29] http://www-03.ibm.com/systems/deepcomputing/solutions/bluegene/index.html

[30] IBM OpenCL Development Kit for Linux on Power and IBM XL C for OpenCL compiler (OpenCL 1.0) http://www.alphaworks.ibm.com/tech/opencl

[31] Intel Parallel Studio XE CAF (details available at: http://software.intel.com).

[32] Java Bindings to OpenCL (JOCL, enables applications running on the JVM to use OpenCL 1.1), http://jogamp.org/jocl/www/

[33] Ian Kuon, Jonathan Rose: *Measuring the Gap Between FPGAs and ASICs*, http://www.eecg.toronto.edu/~ikuon/pubs/fpga2006_kuon.pdf.

[34] Peter Kogge at el., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, 2008, http://www.er.doe.gov/ascr/Research/CS/DARPA exascale - hardware (2008).pdf

[35] K. Kandalla, H. Subramoni, A. Vishnu, D.K. Panda, *Designing Topology-Aware Collective Communication Algorithms for Large-Scale Infiniband Clusters: Case Studies with Scatter and Gather*, IPDPS, 2010.

[36] Charles E. Leiserson, *The Cilk++ concurrency platform*, in proceedings DAC'09, p.522.

[37] Olav Lindtjørn, Robert G. Clapp, Oliver Pell, Oskar Mencer, Michael J Flynn *Surviving the End of Scaling of Traditional Micro Processors in HPC.* http://www.hotchips.org/uploads/archive22/HC22.23.120-1-Lindtjorn-End-Scaling.pdf

[38] LoadLeveler Scheduler, IBM, http://www-03.ibm.com/systems/software/loadleveler/

[39] *Lustre Scalability, Sun Microsystems, 2009,* www.nccs.gov/wp-content/uploads/2009/03/lustrescalabilitywp_updated.pdf

[40] Netlib web-page; http://www.netlib.org/

[41] NVIDIA CUDA C Programming Guide Version 3.2. 11 September 2010, http://developer.download.NVIDIA.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf

[42] NVIDIA CUDA ZONE; http://NVIDIA.com/cuda

[43] Oar Resource Manager, http://oar.imag.fr/

[44] OpenCL - The open standard for parallel programming of heterogeneous systems, http://www.khronos.org/opencl/

[45] Oracle Grid Engine, http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html

[46] PBS Professional, PBS Works, http://www.pbsworks.com/

[47] Platform LSF, Platform Computing, http://www.platform.com/

[48] PRACE Deliverable D6.6, *Report on petascale software libraries and programming models*, http://www.prace-project.eu/documents/public-deliverables-1/public-deliverables-1/public-deliverables-1/public-deliverables-1/public-deliverables-1/public-deliverables-1/public-deliverables-1/public-deliverables-1/public-deliverables-1/public-deliverables/d6-6.pdf

[49] PRACE Workshop on *New Languages & Future Technology Prototypes* (March 1-2, 2010), http://www.prace-project.eu/documents/prace_workshop_on_new_languages_and_future_technology_prototypes.pdf

[50] PyOpenCL (access to the OpenCL API from Python, supports OpenCL 1.1), http://mathema.tician.de/software/pyopencl

[51] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, B. Clifford, *Toward Loosely Coupled Programming on Petascale Systems*, IEEE/ACM Supercomputing 2008.

[52] *Redefining what it's possible. Scientific Computing*. January 7, 2001. http://www.scientificcomputing.com/articles-HPC-GPGPU-Redefining-What-is-Possible-010711.aspx

[53] Rice University CAF 2.0 http://caf.rice.edu/

[54] G.M. Shipman, S. Poole, P. Shamis, I. Rabinovitz, *X-SRQ – Improving Scalability and Performance of Multi-Core Infiniband Clusters*, Euro PVM/MPI, 2008.

[55] G.M. Shipman, T.S. Woodall, R.L. Graham, A.B. Maccabe, and P.G. Bridges, *InfiniBand Scalability in Open MPI*, IPDPS, 2006.

*[56]* SAMSUNG *OpenCL Framework*, http://opencl.snu.ac.kr/

[57] *SLURM: A* Highly Scalable Resource Manager, https://computing.llnl.gov/linux/slurm/slurm.html

[58] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, *System Noise, OS Clock Ticks, and Fine-grained Parallel Applications*, ICS, 2005.

[59] Texas Instruments : *TMS320C6678 MulticoreFixed and Floating-Point Digital Signal Processor*. Literature Number SPRS691, November 2010.

[60] Texas Instruments: *TI's new C66x Fixed- and Floating-Point DSP Core Conquers the 'Need for Speed*'. White paper by Arnon Friedman. Literature Number SPRY147, November 2010.

[61] Texas Instruments: *TMS320C6672/74/78 High-performance multicore fixed- and floating-point DSPs* – Product Bulletin. Literature Number SPRT577a, Februar 2011.

[62] TORQUE, MOAB, MAUI, *Adaptive Computing*, http://www.adaptivecomputing.com/resources/docs/

[63] Unified Parallel C (UPC) http://upc.gwu.edu/

[64] ViennaCL (Linear Algebra and Iterative Solvers) with support for NVIDIA and AMD/ATI GPUs, http://viennacl.sourceforge.net/

## List of Acronyms and Abbreviations

| | |
|---|---|
| AMD | Advanced Micro Devices |
| API | Application Programming Interface |
| APU | Accelerated Processing Unit |
| ArBB | Array Building Blocks (Intel) |
| ARM | Advanced RISC Machines |
| ASIC | Application-Specific Integrated Circuit |
| ATI | Array Technologies Incorporated (AMD) |
| BLAS | Basic Linear Algebra Subprograms |
| BSC | Barcelona Supercomputing Center (Spain) |
| CaSToRC | Computation-based Science and Technology Research Center (of the Cyprus Institute) |
| CAF | Co-Array Fortran |
| CCE | Cray Compiler Environment |
| CCNUMA | Cache Coherent NUMA |
| CEA | Commissariat à l’Energie Atomique (represented in PRACE by GENCI, France) |
| CILK | Multithreaded Programming Language |
| CINECA | Consorzio Interuniversitario, the largest Italian computing centre (Italy) |
| Chapel | Cascade High-Productivity Language (Cray) |
| CLE | Cray Linux Environment |
| CPU | Central Processing Unit |
| CSC | Finnish IT Centre for Science (Finland) |
| CSCS | The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland) |
| COTS | Commercial Off-The-Shelf |
| CUDA | Compute Unified Device Architecture (NVIDIA) |
| CUBLAS | CUDA BLAS Library |
| DARPA | Defense Advanced Research Projects Agency |
| DDN | DataDirect Networks |
| DDR | Double Data Rate |
| DGEMM | Double precision General Matrix Multiply |
| DMA | Direct Memory Access |
| DNA | DeoxyriboNucleic Acid |
| DP | Double Precision, usually 64-bit floating point numbers |
| DVFS | Dynamic Voltage and Frequency Scaling |
| EC | European Commission |

| | |
|---|---|
| ECC | Error-Correcting Code |
| EESI | European Exascale Software Initiative |
| EP | Efficient Performance, e.g., Nehalem-EP (Intel) |
| EPCC | Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom) |
| EX | Expandable, e.g., Nehalem-EX (Intel) |
| Flop/s | Floating point operations per second |
| FFT | Fast Fourier Transform |
| FP | Floating-Point |
| FPGA | Field Programmable Gate Array |
| FPU | Floating-Point Unit |
| FZJ | Forschungszentrum Jülich (Germany) |
| GASNet | Global Address Space Networking |
| 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 |
| GDDR | Graphic Double Data Rate memory |
| GENCI | Grand Equipement National de Calcul Intensif (France) |
| GFlop/s | Giga (= $10^9$) Floating point operations (usually in 64-bit) per second |
| GHz | Giga (= $10^9$) Hertz, frequency =$10^9$ periods or clock cycles per second |
| GNU | GNU’s not Unix, a free OS |
| GPGPU | General Purpose GPU |
| GPU | Graphic Processing Unit |
| GRNET | Greek Research and Technology Network |
| HE | High Efficiency |
| HLRS | High Performance Computing Center (Stuttgart) |
| HMPP | Hybrid Multi-core Parallel Programming (CAPS enterprise) |
| HP | Hewlett-Packard |
| HPC | High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing |
| HPL | High Performance LINPACK |
| HT | HyperTransport channel (AMD) |
| HTA | Hierarchically Tiled Array programming environment |
| IB | InfiniBand |
| IBM | International Business Machines |
| ICE | (SGI) |

| | |
|---|---|
| ICHEC | Irish Center for High-End Computing |
| IDRIS | Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France) |
| IEEE | Institute of Electrical and Electronic Engineers |
| IL | Intermediate Language |
| IMB | Intel MPI Benchmark |
| I/O | Input/Output |
| IOR | Interleaved Or Random |
| IPB | Interprocessor Bus |
| IPMI | Intelligent Platform Management Interface |
| ISA | Instruction Set Architecture |
| ISC | International Supercomputing Conference; European equivalent to the US based SC0x conference. Held annually in Germany. |
| JKU | Johannes Kepler University (Austria) |
| JSC | Jülich Supercomputing Centre (FZJ, Germany) |
| KTH | Kungliga Tekniska Högskolan (represented in PRACE by SNIC, Sweden) |
| LINPACK | Software library for Linear Algebra |
| LLNL | Laurence Livermore National Laboratory, Livermore, California (USA) |
| LRZ | Leibniz Supercomputing Centre (Garching, Germany) |
| MFlop/s | Mega (= $10^6$) Floating point operations (usually in 64-bit) per second |
| MKL | Math Kernel Library (Intel) |
| MPI | Message Passing Interface |
| MTBF | Mean Time Between Failures |
| MxM | Double-Precision matrix-by-matrix multiplication mod2am of the EuroBen kernels |
| NoC | Network-on-a-Chip |
| NFS | Network File System |
| NIC | Network Interface Controller |
| NUMA | Non-Uniform Memory Access or Architecture |
| OpenCL | Open Computing Language |
| OpenGL | Open Graphic Library |
| OpenMP | Open Multi-Processing |
| OMPSs | Programming model based on OpenMP and StarSs developed at BSC |
| OS | Operating System |
| PCIe | Peripheral Component Interconnect express, also PCI-Express |
| PGAS | Partitioned Global Address Space |

| | |
|---|---|
| PGI | Portland Group, Inc. |
| POSIX | Portable OS Interface for Unix |
| PRACE | Partnership for Advanced Computing in Europe; Project Acronym |
| PSNC | Poznan Supercomputing and Networking Centre (Poland) |
| RDMA | Remote Data Memory Access |
| RVDS | RealView Development Suite (ARM) |
| SARA | Stichting Academisch Rekencentrum Amsterdam (Netherlands) |
| SDK | Software Development Kit |
| SGEMM | Single precision General Matrix Multiply, subroutine in the BLAS |
| SGI | Silicon Graphics, Inc. |
| SIMD | Single Instruction Multiple Data |
| SMP | Symmetric Multi-Processor |
| SMT | Simultaneous Multi-Threading |
| SNIC | Swedish National Infrastructure for Computing (Sweden) |
| SoC | System on a Chip |
| SP | Single Precision, usually 32-bit floating point numbers |
| SSD | Solid State Disk or Drive |
| StarSs | Programming Model for Multicores developed at BSC |
| STFC | Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom) |
| TB | Tera (= 240 ~ 1012) Bytes (= 8 bits), also TByte |
| TBB | Thread Building Blocks (Intel) |
| TDP | Thermal Design Power |
| TFlop/s | Tera (= $10^{12}$) Floating-point operations (usually in 64-bit) per second |
| 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 |
| ULP | Ultra Low Power |
| UPC | Unified Parallel C |
| UV | Ultra Violet (SGI) |
| UYBHM | National Center for High Performance Computing of Turkey |
| VHDL | VHSIC (Very-High Speed Integrated Circuit) Hardware Description Language |

# Executive Summary

This report investigates the transition of applications from multi-petascale to exascale performance. Since the recent end of frequency scaling, we observe a rapid evolution of power-efficient computer architectures. Thus, we believe that an investigation of future software systems requires an understanding of future hardware architectures. To that end we have surveyed and evaluated the state-of-the-art in high-performance computer systems, parallel programming languages, and system software and tools. Our goal is to identify trends and developments that have the potential to shape the era of exascale supercomputing.

We summarize our findings separately by topic: computer systems, parallel programming languages and system software and tools.

**Computer Systems**

We have surveyed state-of-the-art computer systems to assess the implications on future exascale software systems. Due to the accelerated evolution of power-efficient high-performance computers, industry presents us with a multi-faceted variety of designs. We considered mature systems as well as prototypes, ranging from hybrid architectures to IBM's BG/Q supercomputer.

High-performance computer architecture has bifurcated into (1) clusters of multicore chips, driven by the mainstream computer market, and (2) accelerated hybrid systems that combine different architectures such as CPUs and GPUs on the board or chip level. At this point in time, it is unclear whether these two directions will reconverge or coexist in future exascale systems. For the programmer, both directions impose a common serious challenge, the lack of programming language support. As a result, programmers have adopted pragmatic hybrid programming techniques, trying to understand and learn how to use these architectures efficiently. On clusters of multicores, the most prevalent programming paradigm is a combination of MPI and OpenMP, and for GPU-accelerated systems programmers combine MPI with Cuda. Consequently, programmer productivity remains a primary concern, in particular for exascale systems.

Our differentiated technology assessment yields the following conclusions. Microprocessor vendors rely on another decade of Moore's law to scale the number of cores per chip, which we expect to exacerbate the efficiency problems caused by the memory wall. Accelerator architectures have the potential to out-perform traditional CPU-based multicore architectures, but the current generation of accelerators fails to deliver a significant advantage in terms of floating-point performance. Furthermore, today's accelerators do not offer the productivity, portability, availability, and resilience of traditional multicore-based systems.

In summary, we are experiencing an era of rapid evolution towards power-efficient high-performance computer systems. Although the outcome is difficult to predict (we're relying on the 1IP-WP9 prototypes to gain deeper insights), current trends are likely to magnify the need for better parallel programming models.

## Parallel Programming Languages

We have evaluated a dozen parallel programming languages for (1) traditional parallel systems based on CPUs with shared-memory and distributed-memory architectures, (2) single and multiple GPUs, and (3) accelerated nodes consisting of both CPU and GPU.

Current de-facto standard for programming traditional clusters of multicores is a pragmatic combination of MPI and OpenMP, and, in addition, Cuda is the de-facto standard for programming NVIDIA GPUs. This mixed programming environment delivers the desired performance up to multi-petaflops.

Most of the evaluated parallel programming languages, including PGAS languages, CAF and UPC, ArBB, Cilk, StarSS, OpenCL, or HMPP, are considered potential candidates for programming exascale applications. At this point in time, we propose to focus on one or very small subset of languages to (1) prepare these languages for exascale computing and (2) offer a clear perspective to application programmers about the languages that are likely to be supported in the future. We have established a framework for the selection process based on benchmarks and a set of evaluation criteria, incl. Performance, productivity, correctness, and sustainability.

## System Software and Tools

We have surveyed the system software and tools used to administrate and operate the supercomputers of the PRACE partners, and observe these trends:

1. Linux has become the dominant operating system used at 80% of all sites.
2. The majority of the tools for system management, data management, monitoring, and resource management are either open-source software of vendor-optimized variants, often based on open-source software.
3. Scalability issues are known for several mission-critical components, including system administration tools, MPI libraries, parallel file systems, and scheduling algorithms for heterogeneous resource management, that require R&D to prepare these components for exascale.
4. Energy management has yet to be introduced to the HPC community that still focuses on raw performance.

We note that much of the system software infrastructure is open-source software, which we suggest to embrace as a powerful means towards developing a common set of tools for exascale machines.

# 1 Introduction

Supercomputing is an essential part of our daily life, ranging from large-scale computations such as weather forecasting to information retrieval via search engines. Today, the PRACE partners operate two petascale supercomputers with the goal to harness and develop the potential of computational science in Europe. Petascale machines offer a peak performance in excess of $10^{15}$ operations per second. While such machines serve scientists today, the next frontier, exascale machines with $10^{18}$ operations per second, shall be reached within the decade.

In this first of two reports on multi-peta to exascale software, we offer the perspective of the PRACE partners on the current state of the art in European petascale computing. Since the end of frequency scaling in 2005, the quest for exascale machines has hit the so-called power wall. Microprocessor speed has topped out at clock frequencies around 2-4 GHz, because dissipating heat beyond about 100 W per chip is not economically viable. Petascale supercomputers are assembled from hundreds of thousands of microprocessors, and consume up to 10 MW of electricity, enough to power thousands of homes. Simply increasing the number of microprocessors to hundreds of millions for an exascale machine would require on the order of 10 GW of electricity, requiring at least one dedicated a power plant. For comparison, such a machine would consume almost one per mille of the world's total energy consumption.

Since alternative low-power technologies to integrated circuits on silicon are not in sight, computer engineers experiment with architectural modifications to the general-purpose programmable von Neumann processor. It is widely known that special-purpose VLSI circuits, for example to compute Fourier transforms, deliver a performance-power ratio roughly three orders of magnitude higher than a software implementation executed on a general-purpose processor. The wide spectrum between special-purpose circuits and general-purpose processor hardware constitutes a broad playground for innovation, with a direct impact on design and use of exascale computers.

Over the past years, the PRACE community has evaluated and experimented with various programming languages, tools, operating environments, and computer architectures, including multicores, manycores, and accelerated hybrid architectures. Each architecture introduces different challenges to the programmer and operator. The trend towards specialized hardware also presents an opportunity to identify new computational fabrics that are particularly suited for computational science. Therefore, we have evaluated state-of-the-art systems from the perspectives of the scientist, computer engineer, and proficient parallel programmer. This report offers an account of these efforts, and an outlook of the opportunities and challenges ahead of us on the road towards exascale computing.

This report reflects the knowledge and experience of the PRACE partners. At the time of this writing, they operate a network of high-performance machines including those shown in Table 1 below. The European PRACE community faces its own challenges and opportunities, and shares many of the problems with other communities. Therefore, the discriminating reader may view the report at hand as complementing other reports on exascale efforts by the US supercomputing community [34] and the International supercomputing community [10].

| Site | Vendor | Type | Model | Processor architecture | # of nodes | # of cores | Mem per core (MByte) | Inter-connect network | Network topology | Internal IO nodes | External IO nodes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BSC | IBM | SMP | JS21 | Powerpc | 2560 | 10240 | 2048 | Myrinet | Fat Tree | N | Y |
| CEA | BULL | ccNUMA | S6010 S6030 | x86_64 | 4000 | 128000 | 2048 | IB QDR | Pruned Tree | Y | Y |
| CINECA | IBM | ccNUMA | P575 | Power | 168 | 5376 | 4096 | IB | Mesh | Y | N |
| CSCS | CRAY | ccNUMA | XT5 | x86 | 1844 | 22128 | 1300 | CRAY SeaStar 2+ | 3D Torus | Y | N |
| EPCC | CRAY | ccNUMA | XE6 | x86_64 | 1856 | 44544 | 1300 | CRAY Gemini | 3D Torus | Y | Y |
| FZJ | IBM | SMP | Blue Gene/P | Power | 73728 | 294912 | 512 | proprietary | 3D Torus | Y | N |
| HLRS | NEC | ccNUMA | | x86_64 | 711 | 5688 + accelerators | 1500 | IB | Tree | Y | Y |
| ICHEC | SGI | ccNUMA | Altix ICE 8200 EX | x86_64 | 320 | 3840 | 2048 | IB DDR | Hyper-cube | Y | N |
| IPB | PARADOX | ccNUMA | | x86_64 | 84 | 672 | 1024 | GbE | Star | N | N |
| JKU | SGI | ccNUMA | Altix 4700 | IA64 | 1 (SSI) | 256 | 4096 | NUMALink 4 | Fat Tree | Y | N |
| KTH | CRAY | CCNUMA | XE6 | x86_64 | 1516 | 36384 | 1333 | CRAY Gemini | 3D Torus | Y | N |
| LRZ | SGI | ccNUMA | Altix 4700 | IA64 | 19 (SSI) | 9728 | 4096 | NUMALink 4 | Fat Tree + 2D mesh | N (SAN) | N (SAN) |
| STFC | IBM | SMP | BlueGene/P | x86_64 | 1024 | 4096 | 512 | Proprietary | 3D Torus | Y | N |
| UYBHM | HP | ccNUMA | | x86_64 | 192 | 1004 | 2048 | IB | Fat Tree | Y | Y |

**Table 1 Contributing PRACE sites**

The remainder of this report consists of three sections. Section 2 assesses the state-of-the-art computer architectures for high-performance computing. Section 3 covers developments in programming languages and environments. Section 4 discusses trends in system management software and tools for high-performance computing.

# 2 Computer Architectures

## 2.1 Hardware Implications on Software for Exascale

In this chapter, we survey the state-of-the-art of high-performance computer hardware from the software perspective, and attempt to deduce trends that affect the transition from petascale to exascale supercomputing. Our survey focuses on hardware features with a dominant impact on the feasibility and viability of scalable software systems for exascale machines.

Our analysis is based on the assessment of advantages and disadvantages of the various hardware platforms with respect to the following criteria:

1. **Scalability**: Multicore and manycore chips raise the issue of scalability within a chip to the same level of concern as scalability across chips. The central issue from the programmer's perspective is the scalability of hybrid programming models for systems of multicores and accelerated architectures, as are details such as architectural support for high-level parallel programming languages and efficient communication.
2. **Performance:** Besides the traditional peak Flop/s or Linpack sustained Flop/s measure, today's supercomputers may be characterized by means of various complementary performance measures, including Flop/$s, Flop/Ws, efficiency relative to peak, etc.
3. **Productivity:** We are concerned about programmability in terms of availability of programming languages, operating systems, tools, as well as comparing the effort required to obtain correct and efficient programs.
4. **Sustainability:** Since supercomputing is a tiny market segment, the affinity of hardware products to the mass market is important to assemble affordable machines from COTS parts, and to ensure competitive upgrades depending on the refresh rate of vendors' offers.
5. **Portability:** Scalable software must be portable across the parts of a large machine, even if different parts are operated at different levels of upgrades. Ideally, runtime systems automate aspects like process placement and load balancing, so that portable programs can be oblivious to the number processors, the network topology, and memory hierarchy.
6. **Availability:** Large-scale machines require adequate manufacturing volumes; supply should match the demand during the construction period with associated cost constraints.
7. **Resilience:** Large-scale machines require fault-tolerance provisions to increase the uptime (MTBF). For example, at the hardware level, ECC protected memories, including caches and hard disks are effective. At the software level, increasing attention is paid to tolerating correctness and efficiency bugs.

Based on the expertise of the PRACE partners, we compile a survey of state-of-the-art hardware in Section 2.2. In Section 2.3, we summarize the trends from the perspective of scalable software systems. Furthermore, we shed some light on complementary research topics pursued with the WP9 prototype projects. In Section 2.4, we provide a watch list of topics that we recommend to be monitored throughout the upcoming year in preparation of PRACE deliverable D9.2.2 “Second Report on Multi-Peta- to Exascale Software” at project month 22.

Disclaimer: Our survey of compute hardware is by no means complete. We have included those systems in Section 2.2 that are available to the PRACE partners, and the partners have gathered sufficient experience as of March 2011 to suggest that this hardware offers a promising contribution towards an exascale system.

## 2.2 Promising Hardware Technologies for Exascale HPC

### 2.2.1 *Intel SandyBridge*

Intel's SandyBridge architecture exemplifies of the evolution of mainstream multicore microprocessor architecture. Integrated on-chip is a special-purpose video processing engine that will be programmable with OpenCL (windows only today).

| Intel Sandy Bridge | Pros | Cons |
|---|---|---|
| Scalability | Intel Sandy Bridge is re-designed version of the Nehalem architecture, so we expect similar on-chip scalability. A 16 core version of the chip is expected. Better power management should allow for denser packaging. | No plans for 8+ socket configuration in the next 2 years. Server chip will not have GPU cores. |
| Performance | Additional set of instructions (AVX) and changes in the architecture (physical register file, Out-of-order cluster, execution cluster and memory cluster) should result in reducing the gap between real life and theoretical performance. Tests and benchmarking will be performed after the hardware shipment. | The integrated GPU cores are programmable yet no experiments are known as of now; therefore it may be difficult to use them for calculations (especially with the server version which lacks of the GPU part). No plans for releasing math libraries using the GPU cores. The theoretical peak performance stays at the same level as in Nehalem. |
| Productivity | No special languages required, all binaries should work more efficient without any changes. | Using AVX requires either new libraries or rewriting part of the codes. Currently there is no simple way to use the GPU cores for calculations. |
| Sustainability | Sandy Bridge and its shrunken version Ivy Bridge will form the backbone of the Intel offer in the next 2 years | |
| Portability | All standard compilers with support for x86 will work. | AVX instruction set must be supported by the compiler. AVX is not backwards compatible |
| Availability | Desktop and mobile versions will are already available. Server 2P configurations will be available 4Q 2011. | 4P servers will not be available during PRACE-1IP. |
| Resilience | Server version will support ECC memory | |

**Table 2 INTEL Sandy Bridge pros and cons**

The Sandy Bridge architecture is an evolutionary step towards better performance and lower power consumption compared to its predecessor Nehalem. Some interesting changes in the architecture resulted in reduced power consumption and increased performance. Intel promotes x86 as a general platform that should cover all possible markets. GPU cores introduced in some of the models have decent performance but since the cores have fixed functionality it may be difficult if not impossible to use them for HPC. In contrast to AMD, who strongly supports accelerated processing units, Intel is more conservative improving the performance with a traditional multicore architecture, new engineering solutions, and adding new instructions.

### 2.2.2 *IBM Power 7*

Released in 2010, it is a RISC processor in 45nm technology. The chip contains 4, 6, or 8 cores, each 4-way SMT-capable, operating at a frequency of 3.0 GHz to 4.25 GHz. As established in the tradition of Power, the SIMD unit (AltiVec) plays a limited role (with respect to the Intel CPUs), but to exploit the power is necessary to use all four FPUs and the Fused Multiply-Add instruction. The chip contains three levels of cache, the first two, 64 kB L1 and 256 kB L2, are private to the core and the 32 MB L3 is shared (but it may be set with core-private partitions).

| IBM Power7 | Pros | Cons |
|---|---|---|
| Scalability | Up to 256 cores, and 1024 threads. | SMT not very efficient, at least under Linux OS. Comparisons made in similar conditions (applications and number of physical cores) show that using SMT either with MPI or OpenMP do not bring better results (compared to SMT turned off). This is in contrast with Power6 processors. |
| Performance | Peak performance of 8-core Power7 is 265 GFlop/s at 4GHz, or 1.25 GFlop/Ws. | Programming for high-performance requires keeping the 4 FPUs busy, and carefully using the cache. |
| Productivity | Linux based system: established tools are available. | GNU compilers and tools suite do not support Power7 extensions yet. |
| Sustainability | IBM Power processors are on the HPC market since the early 1990s. | - |
| Portability | Good for Linux based system. In fact no main problems are expected in porting application from other Linux based HPC systems like BG/P, BG/Q, Cray, Linux clusters. | Problem may arise for the IBM XL compiler suite, due to the limited support for C++ recently introduced standard feature (e.g. Boost library does not compile) |
| Availability | Good. Product line includes workstations, blade systems and supercomputers. | - |
| Resilience | High. Power7 is designed to support PERCS (Productive, Easy-to-use, Reliable Computer System) project in US, so | - |

| IBM Power7 | Pros | Cons |
|---|---|---|
| | resilience has been taken into account by design. | |

**Table 3 IBM Power 7 pros and cons**

The architectural characteristic of Power7 promotes the use of hybrid MPI and OpenMP programming model and cache blocked algorithms.

### 2.2.3 *IBM BG/Q*

The Blue Gene Q (BG/Q) architecture represents a supercomputer architecture that focuses on maximizing the Flop/Ws ratio. Although Blue Gene belongs to the class of general-purpose computers, it is a niche product for supercomputing that has not penetrated the mass market.

| IBM BG/Q | Pros | Cons |
|---|---|---|
| Scalability | System is designed to be extremely scalable up to more than 2M cores. Improved communication network (5-D topology), compared to BG/P | No full (non blocking) fat tree network topology |
| Performance | Very high MF/W ratio target (>2500) (leading Green500 list). System with 20 PFlop/s will be installed at LLNL | Only a first, less efficient prototype system is listed in the Green500 |
| Productivity | Standard MPI, OpenMP programming environment with optimized compilers for standard C/C++ and Fortran available. No hardware specific adaption necessary, compatible to BG/P | System is only really suitable for high scaling applications with more than 16k tasks/threads, nevertheless minimal job size can be 256 tasks/threads without wasting resources. Limited amount of memory per core. |
| Sustainability | 3rd version in Blue Gene series (not COTS but still sustained) | - |
| Portability | Very high, no special, hardware specific programming model/language required | Pure MPI applications might no longer suitable to utilize full performance capability of (this/all) many core architectures. Hybrid programming models might be necessary |
| Availability | Planned to be generally available in 4Q/2011 | Hardware more expensive than x86 solution, but much more power efficient => TCO might be comparable. |
| Resilience | Proven high resilience features from BG/P improved like full system hardware counter collection and real time analyses for pre-emptive hardware replacement strategy (HW will be replaced when recoverable error counters go up before hard errors occure) | - |

**Table 4 IBM BG/Q pros and cons**

Upgraded features of BG/Q compared to predecessor BG/P:

- 4x increased number of cores per node: 16 (+1)
- 2x clock speed: 1.6 Ghz
- 64-bit architecture
- 2x SIMD width: 4-wide double-precision SIMD unit
- 4-way SMT allows up to 64 (MPI) threads per node
- 16 GB DDR3 memory per node

The new hardware features increase the peak floating-point performance from 13.6 GFlop/s of one BG/P node to 205 GFlop/s for one BG/Q node.

Applications can benefit from SMT to hide latencies due to communication and cache misses, provided the application generates a sufficient number of threads (POSIX, OpenMP) per node. Using too many MPI threads per node (max. 64 per node) can be detrimental, because MPI buffers reduce the otherwise available memory capacity and increase the thread scheduling overhead.

The new 5-D topology of the BG/Q interconnection network is transparent to the user. The choice of up to five dimensions should be useful for communication patterns that do not match BG/P's 3-D topology. The optimized MPI communication library folds 3-D patterns automatically into the new 5-D topology.

Two new and unique features in BG/Q are transactional memory and thread level speculation. These features permit executing serial threads speculatively in parallel, and aim at increasing the utilization of the 16-core, 4-way SMT processing node, and to support auto-parallelization of application programs.

### 2.2.4 *AMD Fusion*

AMD's Fusion accelerated processing unit (APU) is another hybrid architecture that integrates a CPU and a GPU on a single chip to share resources and to increase the bandwidth between the two units.

| AMD Fusion | Pros | Cons |
|---|---|---|
| Scalability | The x86 CPU cores are a simplified version of those used in server Magny-Cours chips. Thus, we expect the scalability within the chip to be similar to that of server CPUs. Current releases offer up to 16 cores. The integrated GPU stream cores are equivalent to the Radeon HD 6310. The internal scalability is limited only by the TDP of the enclosure since this architecture is targeting laptops or handheld devices.<br><br>In order to take full advantage of the accelerated processor programmers must use either OpenCL or proprietary Stream technology.<br><br>Current releases are equipped with a Gigabit Ethernet NIC, but it is possible to attach a PCIe card with IB or FC cards. | Current Fusion chips are designed for the low-power, portable device market. Support of multi-socket configurations is missing. |

| AMD Fusion | Pros | Cons |
|---|---|---|
| Performance | The Brazos (AMD) platform has a theoretical peak performance of about 8 GFlop/Ws using single-precision operations.<br>We expect that the integration of CPU and GPU improves the communication latency between the two units, closing the gap between real and theoretical peak performance.<br>We are awaiting shipment of the machines and updated drivers supporting OpenCL. | The memory controller is shared between the GPU and CPU cores. The GPU uses slower DDR3 memory rather than standard GDDR5. |
| Productivity | - | The GPU cores are programmed with OpenGL or DirectX libraries, both of which offer little support for scientific applications. Current drivers support Stream libraries only, although support for OpenCL should be added in the next version. |
| Sustainability | Fusion is an upcoming product. Today, vendors of consumer electronics (Sony, MSI, Toshiba, Asus, HP, etc.) begin shipment of solutions based on the Brazos platform. There are plans for employing the Zacate version of the APU in tablets. | - |
| Portability | The x86 “bobcat” core is fully compatible with x86_64 instruction set. Using the GPU cores requires using the Stream or OpenCL libraries. The GPU-enabled MKL libraries should work without any changes in the code. | - |
| Availability | The Brazos family is represented by the “E” family of low power APUs. Integration of the GPU cores resulted in a cost-reduction. | There are no plans for multi-socket configurations. |
| Resilience | - | Current releases have no ECC protected memory.<br>OpenCL is supported neither in the drivers for Windows nor Linux. |

**Table 5AMD Fusion pros and cons**

The Brazos family (of AMD's Fusion APUs) aims at the low and ultra-low power for the portable consumer electronics market. Tests performed [20] on the first samples of the Brazos platform showed that the performance and power consumption surpassed currently used low power platforms.

### 2.2.5 *NVIDIA Tegra*

The Tegra chip is NVIDIA's flagship product representative for a new breed of system-on-a-chip (SOC) hybrid architectures. Tegra 3 contains three building blocks, a quad-core CPU, a 12-core GPU, and a special-purpose H.264 video decoder. The Tegra SOC is designed to achieve high efficiency with low power consumption on a broad range of applications.

| **NVIDIA Tegra** | **Pros** | **Cons** |
|---|---|---|
| Scalability | Tegra scales across SMP cores with shared memory parallel programming models and across multiple nodes with message passing.<br>On board DDR2-667 memory with peak bandwidth of 5.3 GB/s for a 2 GFlop (peak) chip, which is 2.6 bytes/Flop.<br>Capable of exploiting hybrid shared-memory + MPI programming<br>Capable of exploiting hybrid CPU + GPU programming (Tegra 3)<br>On board storage enables fast checkpoint/restart operations as well as storing large temporary files | Relatively small on-chip memory capacity of 1GB limits size of working set and number of threads of many HPC applications.<br>Off-chip Ethernet connectivity limits the bandwidth of data transfer across nodes. |
| Performance | Uses Quad core low power ARM processor (Tegra 3).<br>Good latency hiding through out-of-order execution and block prefetching<br>Supports double-precision floating-point<br>ARM processor: High performance / power ratio: 2 - 4 GFlop/Ws<br>ULP GeForce GPU, with a good performance<br>Separate ARM7 processor is available for power handling and DVFS support. | Hardware counters on ARM processors are not exposed. Detailed analysis and user level optimization is not possible.<br>Currently no optimized open source scientific libraries available for ARM processors.<br>No detailed information on GPU performance (Hardware counters are not exposed to the user/not present at all) |
| Productivity | Support for multicore programming: OpenMP (gcc for ARM back-end) and OMPSs; and SPMD based programming: MPICH2.<br>Contains ARM RDVS tool chain with compiler, profiler & debugger<br>NVIDIA SDK and PDK are aids application development | GPU PROGRAMMING:<br>Limited support for GPU programming with OpenGL (no CUDA available before a year or so).<br>Efficient Profilers and debuggers are also unavailable for OpenGL. |
| Sustainability | Quad Core Tegra 3 released within one year after Tegra 2 in 2010.<br>Support for CUDA or may be even OpenCL can be expected in future Tegra. | Tegra’s design is targeted towards handheld and netbook devices. It might restrict its widespread application. |

| NVIDIA Tegra | Pros | Cons |
|---|---|---|
| Portability | Multicore applications are easily ported to ARM processor | Limitation in porting from and to OpenGL based GPU applications. |
| Availability | Low cost for the relatively high performance offered.<br>Simple, cost effective installation & maintenance without the need for complex cooling system. | Tegra has not been introduced by NVIDIA in large scale HPC market.<br>NVIDIA might be unprepared for sudden request for creating large-scale HPC cluster. |

**Table 6 NVIDIA Tegra pros and cons**

We expect several promising features from upcoming releases of NVIDIA's Tegra architecture. First, we expect faster CPU processors to appear (Tegra 3 to house Cortex A9 running at 1.5GHz). NVIDIA has announced plans to use ARM's Cortex A-15 processors. To match the performance increase of the CPU, we also expect faster and more GPU cores to be integrated on future chips, with support for CUDA and OpenCL.

Second, NVIDIA envisions support for cache-coherent shared memory between the ARM cores. No cache coherence is planned between the CPU and GPU, though. Memory shall be protected by ECC, and increased memory bandwidth shall support the growing number of cores. To support performance tuning, future Tegra releases shall provide access to hardware counters. Also, higher interconnect bandwidth is expected with multiple Gigabit interfaces and higher PCI Express bandwidth.

Third, we expect improved software support as a result of ARM partnering with NVIDIA, such as open-source scientific libraries tuned for ARM processors, software-controllable voltage and frequency scaling, and optimized versions of the Ubuntu Linux operating system.

### 2.2.6 *NVIDIA Tesla*

NVIDIA's Tesla architecture is the most widely used, programmable GPU in the area of high-performance scientific computing. Attached to a CPU, node-level hybrid systems of CPU and GPU form an important class of contemporary, accelerated supercomputer systems.

| NVIDIA Tesla | Pros | Cons |
|---|---|---|
| Scalability | Support for 10k+ threads of execution per device<br>Data parallel programming at low cost ($).<br>On-chip memory bandwidth, and off-chip memory bandwidth to graphics memory (GDDR5) is very high.<br>Architecture has the potential to scale because of memory design and execution model in terms of # of threads | In chip: adapt the thread to properly hide latency. Poor down-scaling because of high latencies, e.g. of register file accesses.<br>Memories are too small to support the large number of available threads. Thread array (CTA) limit results in code bloat and coding complexity when using more than 512 threads.<br>Limited off-chip bandwidth to host memory and I/O subsystems with respect to the number of threads limits performance benefits of accelerator. |
| Performance | With native double precision (DP) support in Tesla-20 series, improved floating-point performance for scientific applications (double- | Programs should be mostly data parallel (long vectors, few conditionals).<br>Explicit data transfer between host and |

| **NVIDIA Tesla** | **Pros** | **Cons** |
|---|---|---|
| | precision throughput was 1/8th of single-precision on Tesla 10-series) with similar cost envelopes.<br>Effective latency hiding with large number of threads.<br>Realistic improvement in double-precision floating-point performance of about 2-4 times compared to a x86 multi-core chip.<br>Flop/Ws ratio improved by about 2 times compared to contemporary multi-core. | GPU is bottleneck (disjoint address spaces)<br>Rapidly evolving HW and SW stack require continuous tuning and code reengineering to sustain performance.<br>Availability of tools is relatively limited. |
| Productivity | Higher-level languages than CUDA are emerging, e.g. HMPP, PGI, GPUSS, (OpenMP working group on GPGPU [15]) that promise increased productivity.<br>CUDA provides a high productivity-to-performance ratio due to tight coupling of hardware and software stacks. | Higher abstractions, for instance, directives based programming approaches currently do not deliver high performance, and require extensive tuning on individual devices such as different variants of GPUs as well as CPUs.<br>High programming complexity to obtain high performance.<br>Intimate understanding of hardware required to obtain high performance.<br>Rapidly evolving HW and SW stack require continuous tuning and code reengineering to sustain performance. |
| Sustainability | We expect the mass market of graphics applications to drive the evolution of GPUs. | Diverging requirements between graphics and HPC applications, e.g. more memory, higher-precision floating-point operations, and tighter integration with host may not be sustainable HPC-specific enhancements. |
| Portability | CUDA offers (code) portability between NVIDIA devices and generations.<br>OpenCL and other high level interfaces attempt to provide (code and performance) portability across other acceleratored systems | CUDA is a vendor specific language.<br>Code developers typically design and develop codes for both host and GPU.<br>Performance portability using high level interfaces and OpenCL is lacking. |
| Availability | We expect wide-spread availability to continue during the next 3 years, and similar technologies from competitors may surface as alternatives. | Since the business model of NVIDIA is to sell graphics cards as a consumer product, it is unclear whether there will be a wide-spread adoption of Tesla devices at HPC centers, and how the use in HPC centers will affect NVIDIA's R&D, the cost and availability of Tesla chips. |

| NVIDIA Tesla | Pros | Cons |
|---|---|---|
| Resilience | ECC available on GDDR4. | No ECC on instruction paths.<br>Power cycling required in case of buggy GPU programs. |

**Table 7NVIDIA Tesla series GPUs pros and cons**

The Tesla 10 and 20 (Fermi) series devices, along with the CUDA programming environment from NVIDIA have revolutionized the adoption of GPUs as accelerators for scientific computing. Characteristic features include the availability of ECC protected memory, double-precision floating-point arithmetic, and interoperability with communication libraries (MPI) and system management environments have made Tesla series devices a mainstay for high-end, floating-point intensive computing. As a cost-effective product, Tesla GPUs can be considered to be a promising building block for Exascale computing.

We expect that the programming environments, CUDA and OpenCL, continue to evolve, and single address spaces will become available. As of today, PRACE partners are actively targeting and evaluating Tesla hardware, system software and programming environments as a path for Exascale computing.

### 2.2.7 *INTEL MIC*

Intel's Many-Integrated-Core (MIC) architecture is an x86-multicore accelerator chip connected to a host CPU via a PCIe bus. The MIC architecture has evolved from the 80-core Tera-scale research chip, the single-chip cloud computer chip, and the Larrabee project.

| Many Intel Core (MIC) | Pros | Cons |
|---|---|---|
| Scalability | The first Intel MIC product will be made on Intel's 22-nanometer manufacturing process, and will scale to more than 50 cores on a single chip. Each x86 processor core is augmented by a 512-bit wide vector processing unit. All cores are interconnected via a bi-directional on-chip ring network. | Not applicable since MIC is not a product yet. |
| Performance | Not applicable since MIC is not a product yet. | Not applicable since MIC is not a product yet. |
| Productivity | A key advantage for Intel MIC products is the ability to use standard, existing programming tools and methods. Intel MIC cores can be programmed using standard C, C++, and FORTRAN source code. MIC also supports shared memory programming models such as OpenMP and threads. | Not applicable since MIC and it’s software stack are not a product yet. |
| Sustainability | Not applicable | Not applicable |
| Portability | MIC supports the classical shared-memory programming paradigm. The same program source code written for Intel Many Integrated | Not applicable since MIC and its software stack are not a product yet. |

| Many Intel Core (MIC) | Pros | Cons |
|---|---|---|
| | Core products can be compiled and run on a standard Intel Xeon processor. | |
| Availability | MIC development kits, codenamed "Knights Ferry," are already shipping to selected software developers.<br>The first Intel MIC product is codenamed “Knights Corner”. The official product launch is expected not to happen before 2012. | MIC is not yet available as official product. |
| Resilience | Not applicable since MIC is not a product yet. | Not applicable since MIC is not a product yet. |

**Table 8 INTEL MIC pros and cons**

### 2.2.8 *FPGA*

Field programmable gate arrays (FPGA) are gaining momentum in the HPC world, because Moore's law continues to guarantee growing numbers of logic cells that permit synthesizing increasingly complex algorithms directly into hardware.

The application of FPGAs for accelerated HPC can be divided into three groups.

1. Accelerated systems with programmable accelerator architectures inside the FPGA, e.g. Convey Computer's vector personality or Mitrionic's Mitrion Virtual Processor.
2. Accelerated systems with application programs synthesized directly into one or multiple FPGAs, e.g. Maxeler Technologies Ltd [37].
3. Standalone systems based on one or multiple FPGAs, e.g. Pico Computing.

| FPGA | Pros | Cons |
|---|---|---|
| Scalability | **On-chip:** very good scalability, because of a large amount of regular structures with up to 120000 slices and 950000 flip flops, well suited for applications with regular computational patterns.<br>**Across chips:** very good, with high speed interconnects using up to 48 high speed GTX transceivers; maximum GTX transceiver data rate: 4.25 – 6.5 Gb/s. | Requires **hybrid programming:** special tool chains for programming accelerator architectures inside FPGAs or synthesizing FPGA cores. Host applications use accelerated kernels through library calls. Data needs to be copied between host and accelerator memories explicitly. Although commercial solutions exist for point solutions, FPGA-based acceleration remains subject of ongoing research. |
| Performance | Selected applications have been accelerated successfully: bioinformatics speedups 15x-100x [21], financial analytics speedup 47x, compression speedup 10x, seismic imaging speedups 73x-100x, sparse matrix speedups 20x-40x.<br>Typical operating points are:<br>clock frequency: 100-300 MHz | Compared to ASIC designs [33]:<br>**Flops/Ws:** about 12x worse<br>**Performance:** 3-4x worse<br>**Area:** 20-40x larger<br>Results are very application dependent. |

| FPGA | Pros | Cons |
|---|---|---|
| | power consumption: 10-40 W<br>I/O bandwidth: 30 GB/s<br>memory bandwidth: 35 GB/s<br>double-precision floating point: 160 GFlop/s<br>single-precision floating point: 480 GFlop/s<br>Performance and capacity double every year. | |
| Productivity | **Tools:** Vendors supply tool chains for programming, testing, and optimization with every new FPGA generation. High level languages are available, such as HDL's, SystemC, etc. | **Tools:** Automatic synthesis of algorithmic kernels may produce inefficient results. A programmer needs to have expertise in circuit design. Circuit design must be adapted for every new FPGA generation.<br>**Flop/programmer:** worse than with GPU accelerators. User needs programming and circuit design skills. |
| Sustainability | FPGAs are an established mass market with a high product refresh rate. Vendors release new FPGA generations every two years and clock rates are still increasing. | Application dependent. |
| Portability | Verilog and VHDL is widely supported. | Not portable: each FPGA requires different synthesis and explicit data movements. Vendors supply their own non-portable runtime environments. There is no standardization, and interoperability is limited among the different solutions [27]. |
| Availability | At least 2 big vendors, Xilinx and Altera. | Order backlog indicates that manufacturers underestimate or undersupply the market.<br>High unit prices: 1000$ - 17000$ |
| Resilience | **MTBF:** same as microprocessors with the same manufacturing process.<br>**ECC** can be implemented as part of the accelerator core design process.<br>Can be improved with task duplication. | **Debugging:** user needs to be hardware designer to understand the implementation and the timing of a design.<br>Application dependent. |

**Table 9 FPGA pros and cons**

The value of worldwide FPGA shipments is expected to increase from $1.9 billion in 2005 to over $3 billion by 2011, with much of the revenue coming from low-volume shipments, according to a high-tech market research firm. In 2010, the largest end-user segments were communications and industrial, with a combined FPGA market share of 76.8 percent, up from 73.8 percent in 2005, according to a new In-Stat (Scottsdale, Ariz.) report. The HPC market received its last significant investments by FPGA vendors in 2005.

The HPC market potential is not lost on FPGA vendors: some customers are starting to buy thousands of FPGAs to accelerate applications like financial or oil and gas, but are often forced to purchase consultancy, in parts because of a lack of standardization and proprietary technologies. Moreover, most contemporary solutions are oriented toward hardware design rather than software programming.

### 2.2.9 *DSP: Texas Instruments C6000 Multicore*

Digital signal processing arose soon after digital electronics became available. Programs have been implemented on standard devices like computers, microcontrollers and FPGAs as well as on specialized processors like ASICs and digital signal processors (DSP). The typical applications like audio, image, and video processing or signal analysis require a very high number of mathematical operations on large data sets. Specialized DSPs have been tuned to provide high performance at low costs including features like low power consumption and operation at higher temperatures. Recent DSPs benefit from the introduction of increased clock frequencies, IEEE-754 compliant single and double precision floating-point arithmetic and multi core chip designs. The assessment in the table reflects the features of DSP's from the Texas Instruments' TMS320C66x series presented in Fall 2010.

| **Texas Instruments TMS320C6678** | **Pros** | **Cons** |
|---|---|---|
| Scalability | Good scalability on-chip: 8 DSP cores and other subsystems (memory, peripherals, accelerators) connected by a programmable interconnect (Keystone Multicore Architecture, comprising Multicore Navigator, TeraNet, Multicore Shared Memory Controller, Hyperlink)<br>High-speed I/O: PCIe Gen2, Serial RapidIO, TSIP, DDR3-1600, Hyperlink (to other chips) | - |
| Performance | 160 Gflop/s at 1.25 GHz<br>Caches: L1 32kB program, 32kB data, L2 512kB per core, 4 MB shared memory for 8 cores<br>2 Tbps Teranet on-chip interconnect<br>DDR3 ECC memory at 1600 MHz<br>Hyperlink up to 50 Gbps<br>2 PCIe Gen2 lanes with 5 Gbps<br>4 SRIO lanes with 5 Gbps<br><10W at 1 GHz, Operation up to 100°C | - |
| Productivity | Software Development Tools:<br>– Code Composer Studio™ Integrated Development Environment (IDE), including Editor C/C++/Assembly<br>– Code Generation, and Debug plus additional development tools<br>– Scalable, Real-Time Foundation Software (DSP/BIOS™), which provides the basic run-time target software needed to support any DSP application.<br>Hardware Development Tools:<br>– Extended Development System (XDS™) Emulator (supports C6000™ DSP multiprocessor system debug)<br>– EVM (Evaluation Module) | - |

| Texas Instruments TMS320C6678 | Pros | Cons |
|---|---|---|
| Sustainability | Expected to be high. Targeted at large markets and broad application range | - |
| Portability | MS Windows and Linux environment<br>Several application specific libraries available, e.g. BLAS | Specific processor |
| Availability | Available<br>Product family from 2 to 8 cores per DSP at a price from 40 to 200 USD | - |
| Resilience | ECC memory | - |

**Table 10 TI C6000 pros and cons**

The TMS320C6678 is an example of a recent DSP development using modern fabrication facilities and technological paradigms. This allowed adding numerous features useful in many computational intense applications like medical imaging and aerospace applications. Another aspect considered during the chip design was to combine the DSP's with general processors aside their standalone usage. DSP's as accelerators could contribute to the performance and robustness for example in multimedia systems guaranteeing power and space efficiency at the same time. Therefore it is obviously important to evaluate these processors in the field of HPC too.

### 2.2.10 *Tilera TilePro64*

Tilera's TilePro64 architecture exemplifies general-purpose manycore architecture, featuring a grid network of identical RISC processors on a single chip. Past programming experience shows that Tilera can offer power efficient solutions for selected applications with reasonable programming effort.

| Tilera TilePro64 | Pros | Cons |
|---|---|---|
| Scalability | Good scalability on-chip: the 8x8 grid of identical RISC processor cores (tiles) is suited for both signal processing and, in principle, general-purpose computing. | - |
| Performance | Up to 443 billion operations per second (BOPS) at 866MHz<br>37 Tbps of on-chip mesh interconnect<br>Up to 50 Gbps of I/O bandwidth<br>22W at 700MHz | No floating point units<br>32-bit architecture |
| Productivity | C/C++, full SDK under Linux, pthreads<br>Can be used in a standalone machine (ie Linux bootable) | No Fortran<br>Large programming effort to utilize cores with small cache capacity. |
| Sustainability | Used in network and video products | |
| Portability | Linux environment<br>Supports SMP Linux with 2.6 kernel<br>Accelerates pThreaded and shared- | Proprietary C/C++ compiler (will be replaced by Gcc in 2011) |

| Tilera TilePro64 | Pros | Cons |
|---|---|---|
| | memory code<br>iLib™ API's for efficient inter-tile communication<br>Advanced profiling and debugging designed for multicore programming | Specific processor |
| Availability | Available | TilePro64 is now "old", it will be replaced by TileproGX in 2011 (64 bits, 100 tiles) |
| Resilience | ECC memory | - |

**Table 11 Tilera TilePro64 pros and cons**

The Tilera architecture targets massive multi-threading, signal processing and networking applications, relying on its massive internal bandwidth. Lack of floating-point support disqualifies the product for scientific applications.

## 2.3 Analysis of the Hardware Survey

In this Section we identify common trends of the computer hardware surveyed in Section 2.2. To capture the various facets of the technology, we analyse the trends from the perspective of each of the criteria outlined in Section 1.2. Our discussion focuses on the implications of the hardware trends on the software for exascale supercomputer systems.

### 2.3.1 *Scalability*

Computer architects have acknowledged the end of performance growth by frequency scaling at about 4 GHz, because of the power wall. Consequently, industry has refocused, but continues to rely on Moore's law to double the number of transistors per chip about every two years. Hence, we observe the common trend to grow performance by increasing the number of cores per chip without changing the clock frequency. We expect this development to dominate the quest for scalable processor designs in the foreseeable future at the risk of bumping into the memory.

To facilitate large-scale machines, board designers assemble one or more multicore chips into so-called *fat nodes*, featuring cache-coherent shared-memory architecture. This architectural trend emphasizes backward compatibility with existing software stacks. Among the surveyed systems, we find this trend represented by Intel's SandyBridge designs as well as IBM's Power7 and BG/Q. We note that the scalability of fat nodes is limited by the effectiveness of the cache-coherence protocol, just like traditional SMP's in the past. Thus, we expect typical HPC applications to scale up to several tens of cores within a fat node.

In contrast to the traditional fat-node design as an SMP, alternative architectures have entered the market, that we classify as *accelerators*, typically attached to a CPU. Today's accelerators feature larger numbers of smaller cores than in fat nodes and alternative interconnect networks without support for cache-coherence. The most prominent accelerator products are NVIDIA's Tesla, Intel MIC and FPGAs (DSPs from Texas Instruments tailored for scientific computing entered the market recently). AMD's Fusion and NVIDIA's Tegra are hybrid designs that integrate CPUs and GPUs on a single chip. *Accelerated nodes* consist of CPUs and accelerators, and promise to scale beyond fat nodes at the expense of an increased programming effort.

Tilera's TilePro64 is the only architecture in our collection that can be classified as both cache-coherent SMP and accelerator, with a larger number of smaller cores than competing designs. The cores of the TilePro64 chip are so small, however, in parts because they lack a floating-point unit, disqualifying this product for traditional HPC applications.

A de-facto standard has emerged among the programming environments for machines with multiple fat nodes, such as IBM's BG/Q and x86-based clusters. We find that most supercomputer programmers use a pragmatic hybrid model: traditional language environments, including C/C++, Fortran, OpenMP, etc., are used to program each fat node, and a communication library, usually some MPI variant, is used to cope with the distributed-memory architecture when crossing nodes. Alternative parallel programming languages, such as partitioned global address space (PGAS) languages, including CAF and UPC, are neither widely available nor used.

The programming task of an accelerated node may be viewed as a superset of programming a fat node, because most accelerated nodes consist of multiple cores plus one or more accelerators. Programming the accelerator requires additional effort. The de-facto standard for NVIDIA GPUs is the CUDA extension of C. Other programming environments, including OpenCL, HMPP, and PGI Accelerator, fail to be widely available or used. FPGA-based accelerators require hardware design skills and experience with hardware description languages like Verilog or VHDL.

In summary, as primary architectural trend we observe the bifurcation into fat nodes and accelerated nodes. Today, both directions lack proper high-level programming environments. Therefore, it remains unclear, which of the two directions will succeed to offer exascale computing systems with adequate software stacks beyond targeting expert programmers. Furthermore, we are witnessing a period of rapid evolution towards power-efficient high-performance hardware that should be considered an active field of research, and deserves being monitored sharply in the coming years.

### 2.3.2 *Performance*

The notion of performance has changed since we hit the power wall in 2005. The traditional measure of Flop/s is gradually superseded by Flop/Ws. Despite this change, raw performance growth remains the hallmark of supercomputing.

We observe the general trend to improve the performance-to-power ratio as primary system design goal, as exemplified by IBM's Blue Gene supercomputer. Expectations are that exascale machines will have to deliver at least 50 GFlop/Ws to be economically viable. Current architectures are about two to three orders of magnitude off target. However, accelerated architectures attack this goal more aggressively than traditional fat-node architectures by augmenting general-purpose CPUs with power-efficient compute fabrics, leveraging technologies borrowed from the embedded world, as seen for instance in NVIDIA's Tegra. This hardware-centric design direction has the potential to succeed, despite criticisms about their programmability. Obviously, power-efficient high-performance hardware is necessary for exascale machines, yet not sufficient if it requires a disproportional programming effort. Thus, we foresee increased investments in programming environments to complement research on accelerated architectures, including optimizing compilers, runtime systems and performance tools for HPC applications to explore and identify the most promising accelerator architectures.

Most accelerator designs emphasize the performance-to-power ratio at the chip level. Other contenders for low-power chip-level designs may emerge from the embedded market. For example, the ARM instruction set architecture dominates the contemporary mobile market, as it requires less energy per instruction than general-purpose architectures. For exascale machines, at least as much attention must be paid to the energy consumption of the interconnection network, the memory, and I/O subsystems. Today, IBM's Blue Gene is the hallmark of a balanced (but expensive) design. Despite increasing node performance, hundreds of thousands of nodes will be required for exascale computing. At this scale, the programming challenge will be to utilize such a large number of cores efficiently. It remains unclear how many HPC applications may approach a reasonable efficiency level when programmed with today's programming models, even if traditional fat-node architectures can be employed.

In summary, we consider the programming challenge for exascale applications at least as serious as the design of power-efficient chip and system design. Today's programmers focus on raw performance by exploiting various degrees of threading from few fat nodes (OpenMP) to massively data-parallel GPUs (CUDA/OpenCL). Tomorrow's expert programmers may have to worry explicitly about power consumption as well. We expect to see a major leap beyond contemporary hybrid programming with MPI, OpenMP, and CUDA/OpenCL before all but expert programmers will achieve exascale performance.

### 2.3.3 *Productivity*

The evolution of power-efficient hardware is an ongoing process that can only yield winners if both hardware and software in combination achieve exascale performance. While systems based on fat nodes support existing software stacks, new accelerator-based systems offer both challenge and opportunity for new software stacks.

Systems with fat-nodes are in production use today, including IBM's Blue Gene and Power7 systems, as well as Intel-based clusters from various vendors. Contemporary installations operate under the Linux operation system and support traditional programming environments, including Fortran, C/C++, OpenMP, MPI, as well as debugger and performance analysis tools. We expect the software stack to evolve hand-in-hand with advances in hardware. This path to exascale hinges on improvements in power efficiency, and a proportional increase of the number of cores. Thus, productivity is likely to remain at similar levels as today.

Systems with accelerated nodes present a disruptive change not only to the HPC community. Porting legacy codes to new programming environments like CUDA or to FPGAs is often considered too difficult or even technically infeasible. Since the evolution of accelerators is in full swing, the decision to develop new applications for these devices is risky, even when using existing software tool chains. Over the past several years, NVIDIA's CUDA environment for Tesla has matured and is in common use. However, the software stack for Tegra is widely considered unusable for HPC applications. AMD supports OpenCL for its Fusion architecture, which is much less mature than CUDA, and still lacks tools incl. debuggers and performance analysers. Transliterating legacy codes from Fortran to CUDA or OpenCL is not considered productive at this stage. To develop reasonably fast circuits for FPGA accelerators, digital design expertise is required that is neither widely nor readily available to HPC projects.

In summary, the productivity of systems with fat nodes is considered acceptable (albeit low), with the expectation that the event of exascale hardware will not deteriorate the situation disproportionally. Accelerated systems, on the other hand, are still at an experimental stage, precluding a judgement about their productivity.

### 2.3.4 *Sustainability*

Over the past decades, sustainability has been associated with the rate of performance growth: the peak floating-point performance of supercomputers has increased roughly by a factor of 1000 per decade. Chances are that the end of frequency scaling causes this rate to decrease soon.

The predominant method to guarantee sustained performance growth in the recent past has been to assemble supercomputers as clusters from COTS parts. Besides reaping the economic benefits of a mass market, this method also ensures a high degree of backward compatibility on the software side. The following components in Section 2.2 have the potential to remain or become COTS parts in the foreseeable future: IBM Power7, Intel SandyBridge, AMD Fusion, NVIDIA Tesla and, perhaps, Tegra.

A product exposes a medium risk with respect to its expected success to sustain performance growth, if it appears as a niche product targeting the HPC market or if it occupies an unrelated mass market without supporting HPC. IBM's Blue Gene line of supercomputers is a niche product targeting the HPC market. We expect that Blue Gene continues to evolve if IBM can establish a profitable market for this line of supercomputer product.

Prototypical systems expose the highest risk for sustainability, yet present the best opportunity for early adopters to explore or even steer the evolution of alternative architectures. Intel's MIC and Tilera's TilePro64 can be considered prototypes at this stage. FPGA-based accelerated systems have come and gone, for example the Cray XT5h. Although FPGAs have not established their presence in HPC, their potential to outperform other technologies has been demonstrated in several application segments. Therefore, progress of FPGA-based accelerators should be monitored closely. The extended possibilities of DSPs are allowing their usage in HPC as accelerators as well as in a standalone manner are new. This technology is promising. Regardless that it will probably have a high sustainability because the processors will be widely used in many products one hast to see it with respect to HPC as in a very early stage due to the lack of experience at the moment.

From a software perspective, backward compatibility has been the prerequisite for the acceptance of new hardware. With the event of new hardware architectures, in particular accelerated architectures, this stance may weaken in the near future. Unless traditional clusters of fat nodes continue to sustain performance growth with lower power consumption, accelerated architectures are likely offer a significantly better performance-to-power ratio. At this point, we expect major investments to port legacy codes to new environments. Today, we have not reached this point yet. To assess the need for such investments, the PRACE partners are actively exploring and evaluating contemporary accelerated architectures.

### 2.3.5 *Portability*

Computer architecture enables portability if it supports a commonly used software stack, in particular programming environments and operating system. In HPC, we distinguish *functional portability*, the capability to execute a program on a different architecture without major adaptations other than recompilation, and *performance portability*, the ability of a program to achieve similar levels of efficiency across architectures.

Clusters of fat nodes guarantee functional portability. The Linux operating system is established as de-facto standard for high-performance computing on these systems. Furthermore, the Linux environment supports the most common HPC programming languages, C/C++ and Fortran, and MPI and OpenMP for parallel programming. We expect to be able to port programs between IBM's Power7 and Intel's SandyBridge without major code changes, unless proprietary software is used. In contrast, performance portability remains a serious problem. Efficiency varies easily by an order of magnitude from system to system, unless the program restricts its performance critical functionalities to common vendor-tuned libraries. Otherwise, significant algorithmic and machine-specific expertise and programming effort are required to obtain efficient programs even on traditional clusters of fat nodes. We expect similar performance portability problems with future exascale systems.

Programs are not portable across accelerated systems at this point in time. Each system comes with its own programming environment. For example, NVIDIA supports CUDA on its GPUs while AMD supports OpenCL. OpenCL might offer functional portability in the near future through a wide adoption. FPGAs can be programmed with high-level hardware description languages, but commonly lack functional portability even across generations of the same vendor. We should keep in mind, however, that the ecosystem of accelerated architectures is rapidly evolving. The challenge of designing power-efficient high-performance systems leaves little room for portability concerns. Yet computer scientists have to address this issue since most programs will be developed on small systems which may differ significantly in their characteristics from their large-scale counterparts.

### 2.3.6 *Availability*

A product is readily available if it can be purchased for a reasonable price to be delivered within an agreeable time frame. Availability is a function of market size, supply and demand, and size and strength of the vendor.

On the hardware and systems side, today's HPC market has two big companies, IBM and Intel, several mid-sized companies, including AMD and NVIDIA, potentially ARM, Xilinx, and Altera, and a number of smaller companies, including Bull and Cray. In contrast, the software side of the HPC market is dominated by free software; in particular operating system and programming environments, although the majority of HPC applications are proprietary developments pursued within public or private research projects. Furthermore, commercial tools and libraries are in use where highly optimized architecture-specific solutions are required, including optimizing compilers (icc, xlc), scalable debuggers (DDT), and scientific libraries (NAG, MKL, ESSL).

Today's major market drivers are Internet and mobile applications that have relatively little in common with HPC applications from science and engineering. The primary effect of the big market segments is their pull on the evolution of accelerated architectures, such as low-power SOC's, GPUs, FPGAs, and DSP's, including AMD's Fusion, NVIDIA's Tegra, and Tilera's TilePro64. We observe that new hardware releases tend to be in high demand, and feature low availability within the first quarters of their life cycle, in terms of both high price and long delivery periods. In contrast, multicore chips tend to be readily available, perhaps because they serve a mature market that enables vendors to plan ahead.

Availability of components affects the assembly of today’s petascale computers already, which tends to be on the order of one year. Even if we assemble an exascale machine from traditional fat nodes, the assembly period may stretch across multiple technology generations, leading to a high degree of heterogeneity to cope with. We should be prepared to include portability and availability as primary factors in future purchasing decisions.

The availability of vendor-optimized high-performance software is a prerequisite for purchasing any architecture. The core architecture of contemporary multicore chips requires heroic programming efforts without optimizing compilers and high-performance libraries. This situation deteriorates dramatically for accelerated systems, even for well understood vector extensions as found in Intel's MIC. Once again, the market will drive the investments of the vendors in the software stacks (such as Apple pushing OpenCL), potentially widening the gap between the needs of the mass market and the HPC segment.

#### 2.3.7 *Resilience*

Resilience characterizes the ability of a system to survive hard-errors and soft-errors. The miniaturization of silicon technology and the scale of numbers of exascale computing systems require hardware and software techniques to ensure a reasonable uptime and correct execution.

Transient soft-errors, e.g. due to alpha particle hits, as well as manufacturing variability, have already caused some vendors to protect the memories, including caches, with error-correcting codes (ECC) in hardware. These include Intel's SandyBridge, IBM's Power7 and BG/Q. In contrast, products targeting the consumer market lack ECC, incl. NVIDIA's Tegra and AMD's Brazos family.

Software bugs cause machine failures as well. Some machines require a reboot if a user exceeds limits due to careless resource allocation, e.g. the number of threads or processes or the number virtual memory pages. In the worst case, a power-cycle can be required. We observe that the design of an exascale machine requires profound experience with the selection of components, both hardware and software, that extends well beyond the evaluation of a small prototype system.

Henceforth, tackling resilience problems is a very active field of research, with no tangible results yet. It encompasses every actor ranging from the hardware manufacturers to the application developers. Many leads are pursued such as fault-tolerant MPI or using different programming languages. Resilience is yet another field to watch closely in the coming years.

## 2.4 Trends to Watch in the Coming Year

At the time of writing this document, there are more open than answered questions about the future of exascale hardware and software. The evolution of power-efficient high-performance computer systems has only just begun. Vendors are experimenting with different architectures, and are in the process of crystallizing their own philosophies about the architecture of an exascale machine. Thus, it is difficult if not impossible at this time to conclusively identify a common trend towards exascale software. We can, however, offer a list of indicative trends to watch over the coming year.

1. **Low-power architecture:** We expect alternative instruction set architectures to push into the general-purpose processor market. How would a shift from the dominant x86 ISA to another low-power ISA such as ARM's affect HPC? Are our applications sufficiently independent of the ISA to be portable, provided that optimizing compilers and high-performance libraries become available?
2. **Accelerated systems:** We expect the node architecture to evolve towards modestly parallel multicores and massively parallel accelerated nodes. Do our applications exhibit sufficient parallelism to match such architectures, or do these architectures exacerbate the memory wall problem so as to be ill-suited for HPC?
3. **Memory systems/Interconnect:** We expect vendors to differentiate their HPC offers through variations of their memory systems and interconnect design. Such system architectures may improve programmability, and accelerate the proliferation of alternative programming models, such as PGAs. Can our applications benefit from such improvements?

4. **Programming environment:** We expect the evolution of accelerated architectures to produce either a winner or a convergence of programming environments. Such developments may affect the sustainability of HPC applications, for example if CUDA or OpenCL gain momentum at the expense of OpenMP. Are we exposed to such developments, and if yes what are the implications?
5. **Market forces:** We expect the evolution of low-power technologies to impact future COTS components. Are these components suited to assemble high-end exascale machines, or will they cause promising high-performance technologies to disappear?
6. **Education:** Watch the curricula of Universities and HPC professional training. Are we preparing the next generation of scientists properly to cope with the challenges ahead of us?

Deliverable D9.2.2 will answer some of these important questions (issues) and formulate recommendations on Software for Exascale machines.

# 3 Programming Languages

The assessment of programming languages started in the PRACE Preparatory Project (PRACE-PP). First results were published in [48]. A partial update appeared in the PRACE Workshop on “New Languages & Future Technology Prototypes” (March 1-2, 2010) [49]. Since then, the focus or our work as shifted from evaluating multicore CPUs and a plethora of accelerator systems such as GPUs, Cell, Clearspead, and FPGAs, to evaluating a plethora of programming languages for two types of systems, traditional multicore CPUs and GPU-based accelerators. The languages under consideration are CUDA (+MPI), OpenCL, CAPS hmpp, PGI Accelerator Compiler, OpenMP+MPI, UPC, CAF, Chapel, StarSs (+UPC), Cilk (+UPC/MPI), ArBB, TBB and HTA.

Section 3.1 offers an overview for each language, including a description of the programming and memory model, a code example, information on available libraries and tools, and reports experimental results. Section 3.2 summarizes our assessment.

## 3.1 Description & Results for Each Language

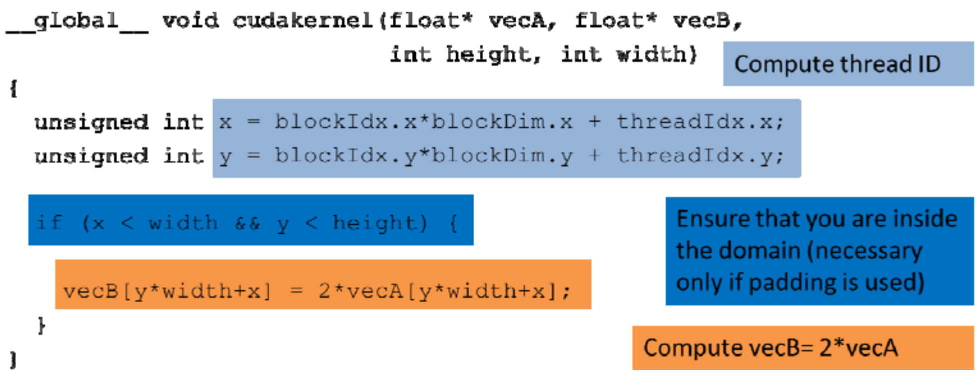
### 3.1.1 *CUDA*

**Description**

CUDA (Compute Unified Device Architecture) is a quickly maturing software development environment provided free of charge by NVIDIA to develop applications for NVIDIA graphics processors. Today, CUDA is the most mature and most widely used development platform for GPGPUs [52]. It is an extension of the C programming language to program NVIDIA GPUs attached to a host CPU. The CUDA programmer must expose enough data parallelism to mask the latencies of multithreaded GPU resources. For further information see Section “Description of Programming and Memory Model” in [48] and [41].

A CUDA program consists of one or more sections that are executed on either the host CPU or a GPU device. The device is used commonly as an accelerated coprocessor where data parallel portions of an application are executed as a kernel which runs in parallel on many threads. A kernel is expressed as a C-language subroutine that uses data allocated in the GPU memory (Figure 1).

Apart from the subroutine arguments, a call to a kernel includes the specification of an execution configuration, added between triple angle brackets “<<<” and “>>>”. The configuration parameters define the mapping of threads to GPU resources (Figure 2). The product of the first two parameters between the angle brackets defines the total number of simultaneously running threads for a given kernel, see [48] and [41].

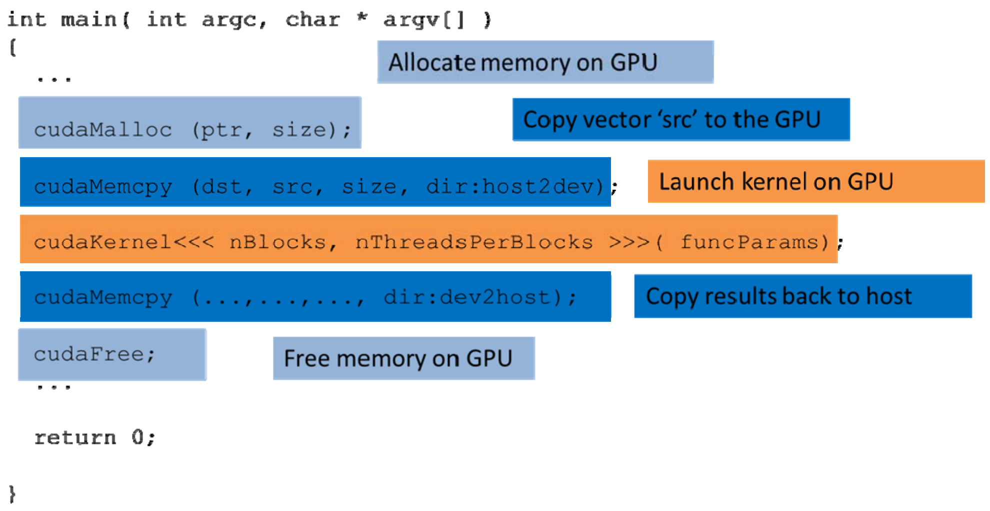


```
__global__ void cudakernel(float* vecA, float* vecB,
                            int height, int width)
{
  unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
  unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;

  if (x < width && y < height) {
    vecB[y*width+x] = 2*vecA[y*width+x];
  }
}
```

Compute thread ID
Ensure that you are inside the domain (necessary only if padding is used)
Compute vecB= 2*vecA

**Figure 1 Simple CUDA kernel**



```
int main( int argc, char * argv[] )
{
  ...
  cudaMalloc (ptr, size);
  cudaMemcpy (dst, src, size, dir:host2dev);
  cudaKernel<<< nBlocks, nThreadsPerBlocks >>>( funcParams);
  cudaMemcpy (...,...,..., dir:dev2host);
  cudaFree;
  ...
  return 0;
}
```

**Figure 2 CUDA host code**

Free libraries, software tools, and development kits such as the NVIDIA CUDA SDK are available to assist the CUDA programmer. Both NVIDIA and the CUDA developer community [42] continue to develop and improve the libraries and software environment. New SDK releases appear approximately every six months.

**Experience & Results**

This section reports ongoing work at ICHEC on the phiGEMM library for accelerated matrix multiplication routines on GPUs.

BLAS 3 routines, such as [S/D/Z]GEMM, are widely used in many scientific applications. NVIDIA has improved the CUBLAS library in the last years, achieving significant performance (e.g.~350GFlop/s double precision for a DGEMM-kernel on the GPU). For medium to large sized datasets, the time required to move data between the host and the device still represents a bottleneck, even though the peak performance is quite impressive.

Starting from a previous NVIDIA project [11], ICHEC has developed the phiGEMM library. This library allows the target application to perform [S/D/Z]GEMM operations using both CPU and GPU simultaneously. It implements both the data transfer management and the split logic (amount of computation to be performed in parallel on each device, CPU and GPU, in order to achieve an optimal load balance). Since phiGEMM follows standard BLAS [40] interfaces, the users just need to link to the library, like any other common library such as MKL, ACML, ESSL, GotoBLAS, etc. The phiGEMM matrix-matrix multiplication calls simultaneously an external BLAS library (single- or multi-threaded) for the CPU-side and a CUBLAS kernel instance on the GPU, as efficiently as possible.

Figure 3 shows the performance of the phiGEMM DGEMM implementation compared to CUBLAS. The split factor is calculated starting from the pure CUBLAS performance and the pure CPU performance, for every test performed varying the matrix size. Figure 4 reports the same tests using page-locked (or pinned) memory. We achieved great performance increasing the matrix sizes in the range from 512x512 up to 10240x10240 that represents almost the limit of the GPU memory. The usage of page-locked memory on the host allows the phiGEMM library to overlap efficiently the CPU computation and both the CUBLAS call and the asynchronous host-to-device data transfers.

All tests were performed using CUBLAS v3.2 on a GPU TESLA C2050 (448 cores and 3GByte of memory on the card) and the latest multi-threaded Intel MKL library on an Intel Xeon CPU X5650 (6 cores, 2.67 GHz).

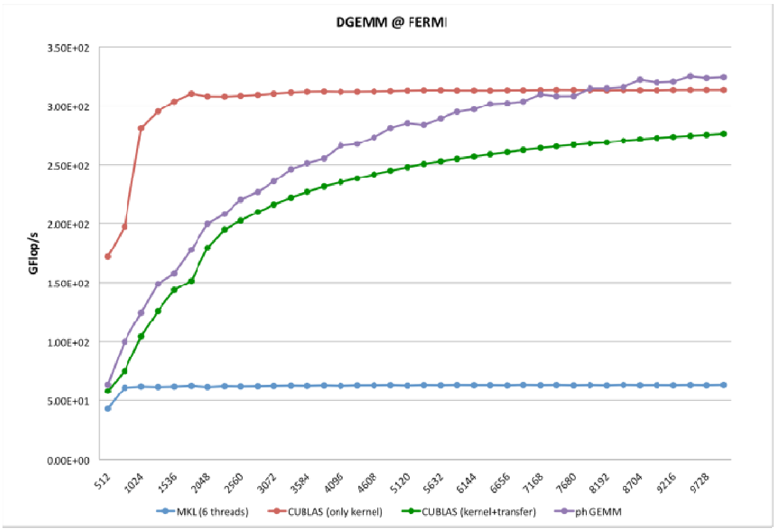


**Figure 3 Comparison between CUBLAS, MKL and phiGEMM (DGEMM)**

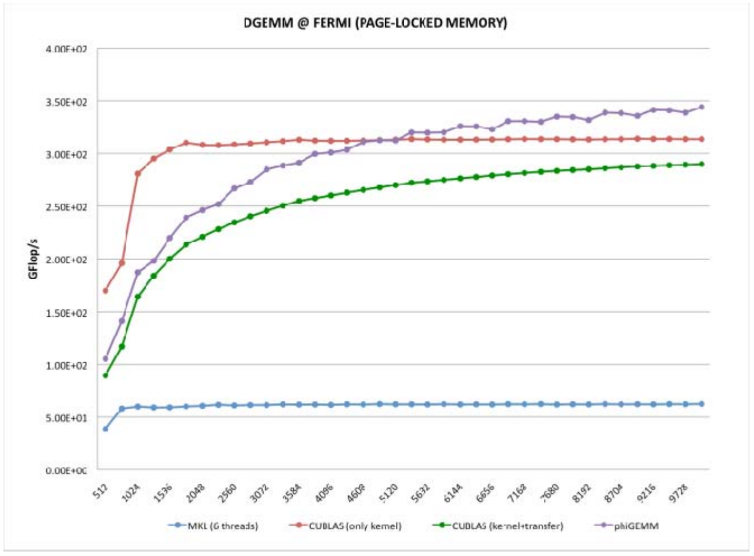


**Figure 4 Comparison between CUBLAS, MKL and phiGEMM (DGEMM, PINNED memory used)**

**Pros & Cons**

| | Pros | Cons |
|---|---|---|
| Scalability | Massive data parallel language. Easy to scale on multiple GPUs with both OpenMP within a single node and MPI across distributed nodes. | Only CUDA 4.0, which still is in the status of a "candidate release", provides RDMA functionality to allow direct data exchange between GPU and GPU both intra- and inter-nodes. |
| Performance | Applications have been sped up to one, two and even three orders of magnitudes. Currently the best supported language for NVIDIA hardware. | Rather low level programming language. Hardware driven performance tuning is often necessary and might become counter-productive on further generation of NVIDIA technology. Effort needed to port an application has to be considered while estimating performance benefits. |
| Productivity | The CUDA programming model is easy to learn and maintain. Its nature as a C extension language along with many useful C++ features makes it easily programmable and readable. The SDK offers a number of productivity tools free of charge (e.g. debugger, profilers, memory checker). | Large effort in performance tuning is commonly requested. NVIDIA CUDA is C/C++ only. Porting codes from other languages implies porting to C/C++ beforehand. PGI provides a CUDA Fortran compiler, but it has to be procured in addition to the free NVIDIA environment. |
| Sustainability | By far the most widely adopted and mature language for GPU computing. Strong support and investment by NVIDIA on both hardware and software. Large ecosystem already developed, with ported codes, library and development tools. | Long-term support of CUDA programming language is fully under NVIDIA's control. Exploitation of both hardware and software of further generations might require substantial code changes. |
| Correctness | Perfect integration with the hardware, which now supports ECC correction and IEEE floating point arithmetic. Effective tools for memory problem detection and debugging are available. | Trickier to handle than high-level languages. It's massively parallel nature along with its characteristic to be a separated entity makes it harder to debug than standard code. |
| Portability | Very good ascending compatibility from a software and hardware point of view. Work on all shipped NVIDIA CUDA enabled device. | Only available for NVIDIA CUDA enabled GPU. Software features improvement often depends on hardware support. Some optimizations implemented for specific target can become counter-productive on a different one. |
| Availability | CUDA is delivered as free software from NVIDIA web site. It's actively maintained and developed. | Only available for NVIDIA CUDA enabled architecture. |

**Table 12 CUDA pros and cons**

### 3.1.2 *CUDA+MPI*

**Description**

Hybrid CUDA+MPI, along with hybrid OpenCL+MPI, are the two programming environments that offer most control to the developer for programming today's clusters of GPGPU systems. This level of control sacrifices code simplicity, because hybrid programming requires explicit programming of two architectures. Hybrid CUDA+MPI programs consist of both code for the host CPU and the kernel code for the GPU device. In principle, the host code can be written in any language interoperable with MPI and CUDA. The kernel code is typically written in CUDA.

The developer is responsible for managing data transfers between (1) the host memory of the CPU and the device memory of the GPU and (2) between MPI processes. Explicit programming of data transfers results in more complex codes, but gives the developer an opportunity to fine tune placement and timing of transfers, e.g. to exploit latency hiding.

To date, direct GPU-GPU communication bypassing the host is not feasible. Instead, applications must be aware of the memory hierarchy. The memory hierarchy introduces high latencies for data transfers between GPUs, which may affect scalability. The forthcoming CUDA toolkit release (version 4.0) will offer two optimizations for multi-GPU programming: (1) GPUs on the same node may communicate directly through PCIe, and (2) modifications to MPI permit inter-GPU communication.

**Experience & Results**

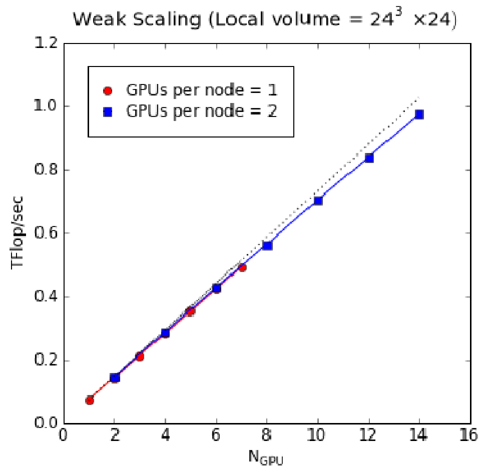
This section reports on the QUDA library developed at CaSToRC, the Cyprus Institute.

QUDA is a publicly available library that provides mixed-precision implementations of the CGNR and BiCGstab sparse linear solvers for inverting Wilson-Dirac matrix on NVIDIA CUDA platforms [5]. The current release includes optimized solvers for a variety of fermion actions, while also providing a C interface to allow integration into existing applications. A parallel version is also available, however at the moment this version has beta status [4].

Programming with MPI on heterogeneous systems, one needs to take into account the memory communication patterns. To fully utilize the PCIe bus, one can use the so-called asynchronous memory copy methods provided by the CUDA API that operate with page-locked host buffers. These non-blocking copies allow overlapping data transfer with other operations, either on the host or on the device. However current network devices can directly access only non-pinned host buffers for communication, which means either synchronous device-to-host (or vice versa) data transfer can be used, or otherwise extra memory operations on the host are required.

In QUDA, parallelization of the Wilson-Dirac matrix inversion across multiple GPUs is done by partitioning the time direction of the lattice while spatial directions reside on a single GPU. Hence, the application of the Wilson-Dirac matrix on the spinor field is performed by computing the inner sites, communicating the neighbors, or boundaries, and finally computing the boundary sites. The first two steps can be overlapped, which is achieved using CUDA's streaming capabilities. This is done in conjunction with asynchronous data transfers between device and host, as well as non-blocking MPI communications.

We performed scaling tests on our local Tesla S1070 cluster of 7 GPU nodes with Infiniband SDR fabric. The single precision BiCGstab solver with overlapping MPI communications was run. Figure 5 shows a weak scaling test. A near linear scaling is observed, with 1 TFlop/s sustained performance on 14 GPUs. We note that the problem size in this situation is very asymmetric, meaning such geometry would not be used in a typical production run. Strong scaling is shown in Figure 6. Because of GPU memory requirements, we reduced the local volume to allow more subdivisions of the global problem size. This geometry represents a more natural choice, suitable for production runs. Under this conditions, the code scales poorly since increasing the surface to volume ratio results in the application being communication bound, with the compute kernels completing earlier than the memory transfers. Reducing latency is a key to improving this situation, which will be addressed in new hardware and software developments.



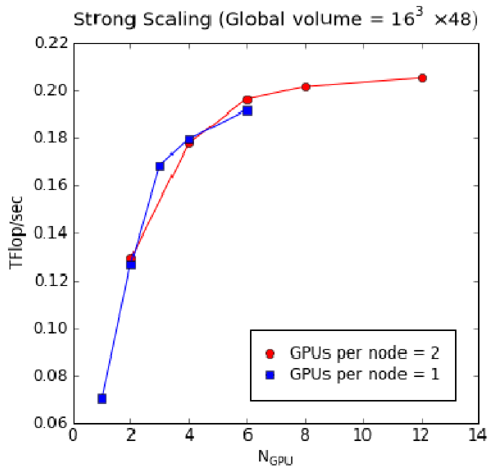


**Figure 5 Weak scaling plot of Quda BiCGstab inverter performance**

**Figure 6 Strong scaling plot of Quda BiCGstab inverter performance.**

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | Excellent, MPI is the most well established paradigm for programing distributed memory HPC systems. | Depending on the underlying communication schemes scalability might be limited by the host-GPU communication bottleneck. |
| Performance | C for host and its extension for device code (C for CUDA) can achieve excellent performance. | - |
| Productivity | - | Complex code is required to manage two architectures and two-level memory hierarchy. |
| Sustainability | Excellent for MPI. | Not clear whether OpenCL will replace CUDA as a standard for programing accelerators. |
| Correctness | - | The well-known pitfalls in MPI programing apply here. Debugging GPU kernels can also be difficult due to large number of threads. |
| Portability | MPI is a standard and implementations are available on any HPC system. | CUDA restricts to NVIDIA GPUs. |
| Availability | Excellent. MPI available on effectively any HPC system. CUDA available on any NVIDIA HPC system. | - |

**Table 13 CUDA+MPI pros and cons**

### 3.1.3 *OpenCL*

**Description**

OpenCL (Open Computing Language) is an open, royalty-free standard for general-purpose parallel programming of heterogeneous systems. It provides a framework for multi-processor computing and for parallel programming of GPUs. The framework includes a programming language, based on C99, for writing functions (kernels) executed on OpenCL devices, and an API. An overview of the OpenCL architecture, its execution and memory model are described in [48]. OpenCL supports both data-parallel programming and task-parallel programming. Furthermore, OpenCL is interoperable with MPI and other standard libraries.

Since the publication of PRACE-PP deliverable D6.6 [48] a new version of OpenCL has been released by the Khronos Group: the OpenCL 1.1 Specification (rev. 36, September 30, 2010) and OpenCL 1.1 C++ Bindings Specification (rev. 4, June 14, 2010) [26]. OpenCL 1.1 adds functionalities for enhanced parallel programming flexibility and performance, including host-thread safety, sub-buffer objects, 3-component vector data types, and new OpenCL built-in C functions.

OpenCL (v1.0 or v1.1) supports a range of hardware from mobile technologies to HPC, including:

1. NVIDIA GPUs (Tesla, Quadro, GeForce and ION),
2. AMD Fusion APU series (hybrid CPU+GPU),
3. AMD/ATI GPU series (Radeon HD, FirePro),
4. AMD x86 CPU SSE 2.x or later,
5. ARM Mali-T604 GPU,
6. Intel CPU series (e.g. Intel Core i7, Intel Xeon 7500),
7. IBM Cell/BE
8. IBM Power servers,
9. built-in graphic cores of Intel Sandy Bridge and Ivy Bridge processors.

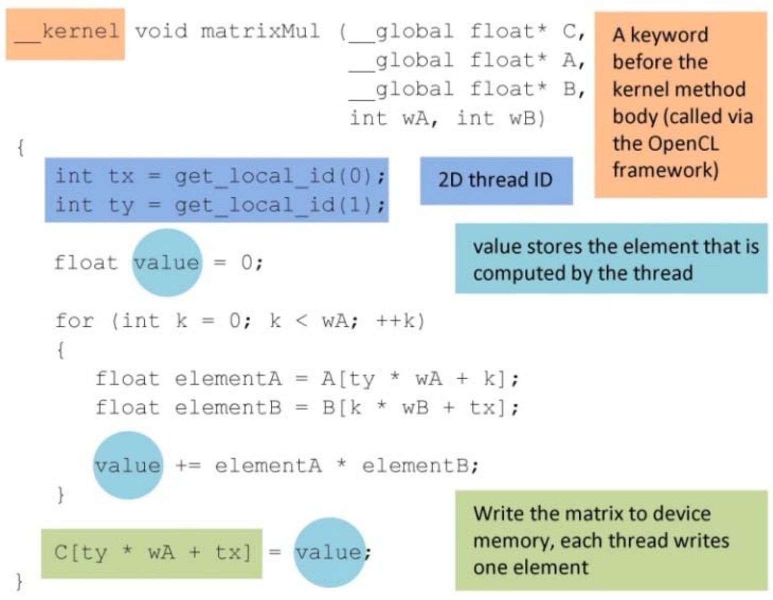
Libraries for scientific computing written in OpenCL include:

1. ACML (BLAS, LAPACK, FFT, RNG) for AMD/ATI GPUs,
2. CUBLAS (BLAS) and CUFFT (FFT) for NVIDIA GPUs,
3. and ViennaCL (Linear Algebra and Iterative Solvers) with support for NVIDIA and AMD/ATI GPUs [64].

Tools for OpenCL software development include:

1. gDEBugger CL (visual debugger),
2. Swan (tool for porting CUDA to OpenCL),
3. AMD Accelerated Parallel Processing SDK v2.3 with full support for OpenCL 1.1,
4. SNU-SAMSUNG OpenCL Framework (supports multiple target machines such as Cell BE processors, ARM processors, DSP processors) [56]
5. IBM OpenCL Development Kit for Linux on Power and IBM XL C for OpenCL compiler (OpenCL 1.0) [30],
6. Java Bindings to OpenCL (JOCL, enables applications running on the JVM to use OpenCL 1.1) [32]
7. PyOpenCL (access to the OpenCL API from Python, supports OpenCL 1.1) [50].

### OpenCL Code Example



```
__kernel void matrixMul (__global float* C,
                         __global float* A,
                         __global float* B,
                         int wA, int wB)
{
    int tx = get_local_id(0);
    int ty = get_local_id(1);

    float value = 0;

    for (int k = 0; k < wA; ++k)
    {
        float elementA = A[ty * wA + k];
        float elementB = B[k * wB + tx];

        value += elementA * elementB;
    }
    C[ty * wA + tx] = value;
}
```

A keyword before the kernel method body (called via the OpenCL framework)
2D thread ID
value stores the element that is computed by the thread
Write the matrix to device memory, each thread writes one element

### Experience & Results

The Euroben benchmark mod2am/MxM was ported to OpenCL at PSNC/WCNS. We report the benchmark results on NVIDIA GTX480, AMD/ATI Radeon HD 5970 and AMD/ATI Radeon HD 5870.

The benchmark uses single-precision floating-point arithmetic. The results in Figure 7 and Figure 8 exhibit performance improvements compared to the results reported in the PRACE-PP tests on NVIDIA Tesla (2xC1060 GPU board). The Radeon HD 5970 is supported in single-GPU mode only. This, along with the lower clock rate, is the main reason why the newer card shows worse performance for HPC compared to the older 5870. Benchmarks with cryptography problems revealed that the code needs adaptation in order to reach optimal performance on GPU architectures. The preliminary conclusion is that even if the single GPU from AMD/ATI shows better performance than NVIDIA, their solutions for HPC are still not mature.

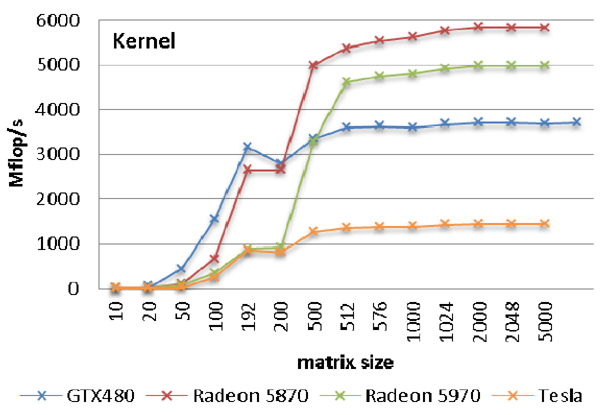


Figure 7 OpenCL performance of mod2am

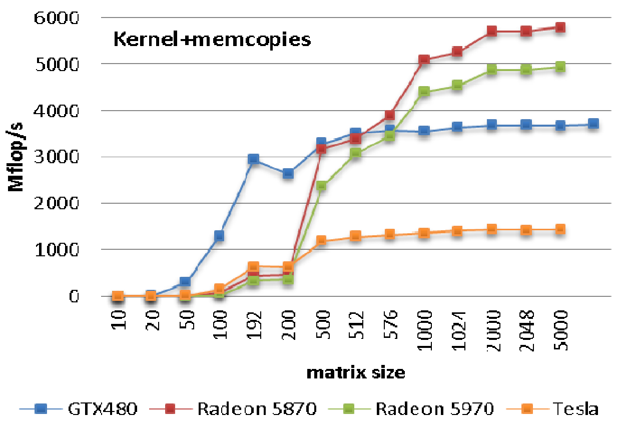


Figure 8 OpenCL performance of mod2am including memory copies

**Pros & Cons**

| | Pros | Cons |
|---|---|---|
| Scalability | Scales extremely well and achieves very high performance on SMP systems. | - |
| Performance | Has big potential for massive-parallelism. It is possible to achieve very high performance on inexpensive GPU hardware. | The code should be architecture-oriented. PCI bus data transfer is still a bottleneck. |
| Productivity | Developing code is quite easy for people that are used to program in C or CUDA. | Obtaining very efficient kernel code and optimal performance requires more effort, experience and using device specific information. |
| Sustainability | The Khronos Group consists of many industry-leading companies and institutions including AMD, IBM, Intel and NVIDIA. | - |
| Correctness | - | - |
| Portability | May run on a number of architectures. The kernel code can be reasonably easy transferred from one architecture to another. The compiler is built into the runtime. | To reach optimal performance the code should be written for specific device. |
| Availability | Open and royalty-free standard.<br>Tools simplifying software development and scientific libraries exist. | - |

**Table 14 OpenCL pros and cons**

### 3.1.4 *Hybrid Multicore Parallel Programming workbench (HMPP)*

**Description**

HMPP offers a high level abstraction for hybrid programming of (multi-)GPU systems without the complexity associated with GPU programming. HMPP syntax consists of annotations for C and FORTRAN programs with pragma directives similar to OpenMP.

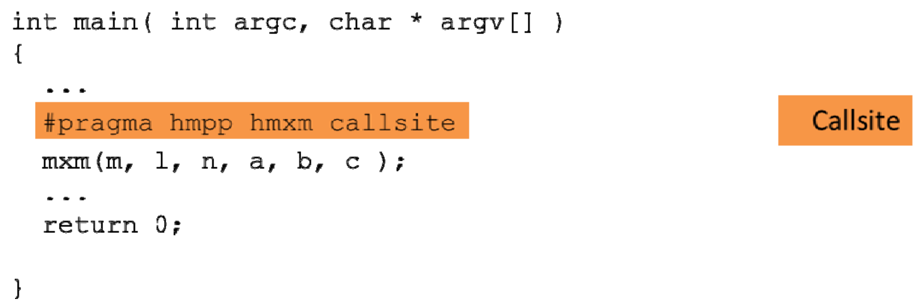
The HMPP Workbench contains C and Fortran compiler drivers, target code generators which support both CUDA and OpenCL, and a runtime library for the execution of parallel hybrid applications. Details can be found in Section 6.4 of [48].

The programmer adds a “pragma hmpp codelet” to declare that a function shall execute on an accelerator and a “callsite” pragma before the function call. The HMPP code generators auto-parallelize the code and translate the accelerator function into CUDA or OpenCL. Hidden from the user, the GPU-vendor SDK is then used to compile the generated code and create a dynamic library which is loaded by the HMPP runtime. The HMPP runtime also takes care of the GPU allocation and all data transfers.

HMPP offers pragmas to define which data must be copied in or out the GPU, to schedule pre-allocation of the GPU and prefetching of data before the codelet call. Furthermore, HMPP offers pragmas to define which data shall be stored in host memory after codelet execution on the GPU is complete, and to specify when the GPU can be safely released. The concept of codelet groups introduces device resident data, which may reduce the overhead caused by data transfers. It is possible to “pin” several codelets to one specific GPU.

HMPP and OpenMP are interoperable, and of interest to program GPUs and multicore CPUs. HMPP and MPI should be interoperable, but we did not verify this capability yet. If necessary, the programmer may edit the generated CUDA/OpenCL code and include library calls manually.

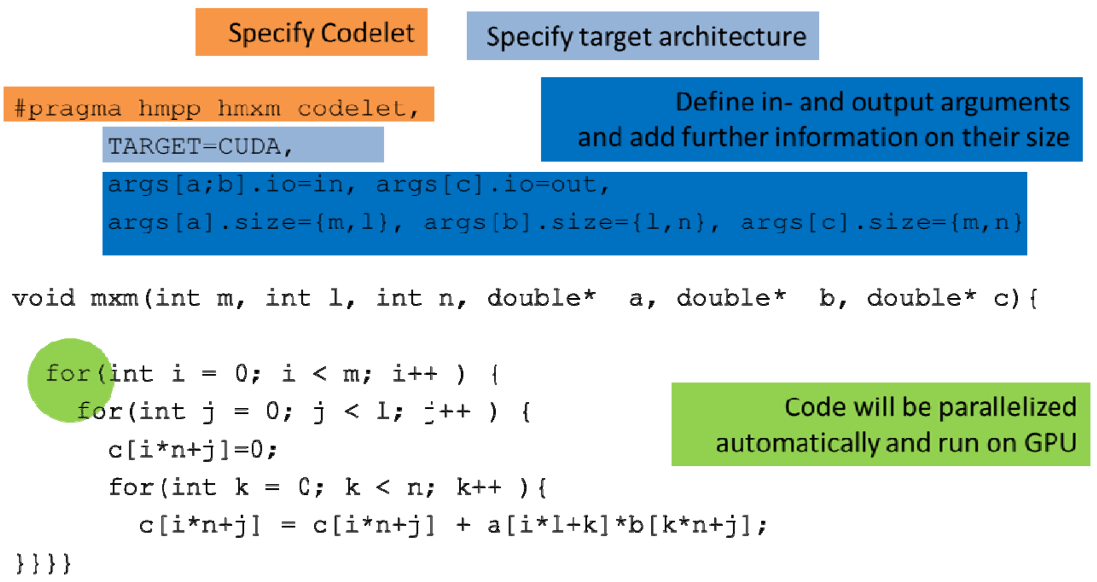
The PRACE Deliverable D6.6 [48] described HMPP release 2.0. Newer releases include support for HMPP regions, the OpenCL code generator and support for several other compilers (SunStudio, Open64, PGI compilers, Absoft Pro Fortran compiler) in addition to GNU and Intel compilers, a Microsoft Windows version of HMPP, a new plugin for Eclipse Galileo and the integration with the Vampir profiling tool and Allinea DDT. We mention that PathScale and CAPS have joined forces in June 2010 to push the HMPP model as a new standard for GPGPU programming ("HMPP Open Standard"). The new PathScale ENZO Compiler Suite supports GPUs using HMPP's directive-based programming model.



```
int main( int argc, char * argv[] )
{
  ...
  #pragma hmpp hmxm callsite
  mxm(m, l, n, a, b, c );
  ...
  return 0;

}
```

**Figure 9 HMPP callsite example**



```
#pragma hmpp hmxm codelet,
      TARGET=CUDA,
      args[a;b].io=in, args[c].io=out,
      args[a].size={m,l}, args[b].size={l,n}, args[c].size={m,n}

void mxm(int m, int l, int n, double*  a, double*  b, double* c){

  for(int i = 0; i < m; i++ ) {
    for(int j = 0; j < l; j++ ) {
      c[i*n+j]=0;
      for(int k = 0; k < n; k++ ){
        c[i*n+j] = c[i*n+j] + a[i*l+k]*b[k*n+j];
}}}}
```

Specify Codelet
Specify target architecture
Define in- and output arguments and add further information on their size
Code will be parallelized automatically and run on GPU

**Figure 10 HMPP codelet example**

**Experience & Results**

This section reports benchmark results for Euroben kernels mod2am and mod2as, performed at LRZ.

We have compared the performance of mod2am/MxM and mod2as/SpMV using HMPP and the PGI accelerator compiler with CUDA implementations. Figure 11 shows the performance on NVIDIA Tesla C1060 and C2050 (“Fermi”) GPUs. The optimized HMPP version uses prefetching and various loop manipulating directives like unrolling.

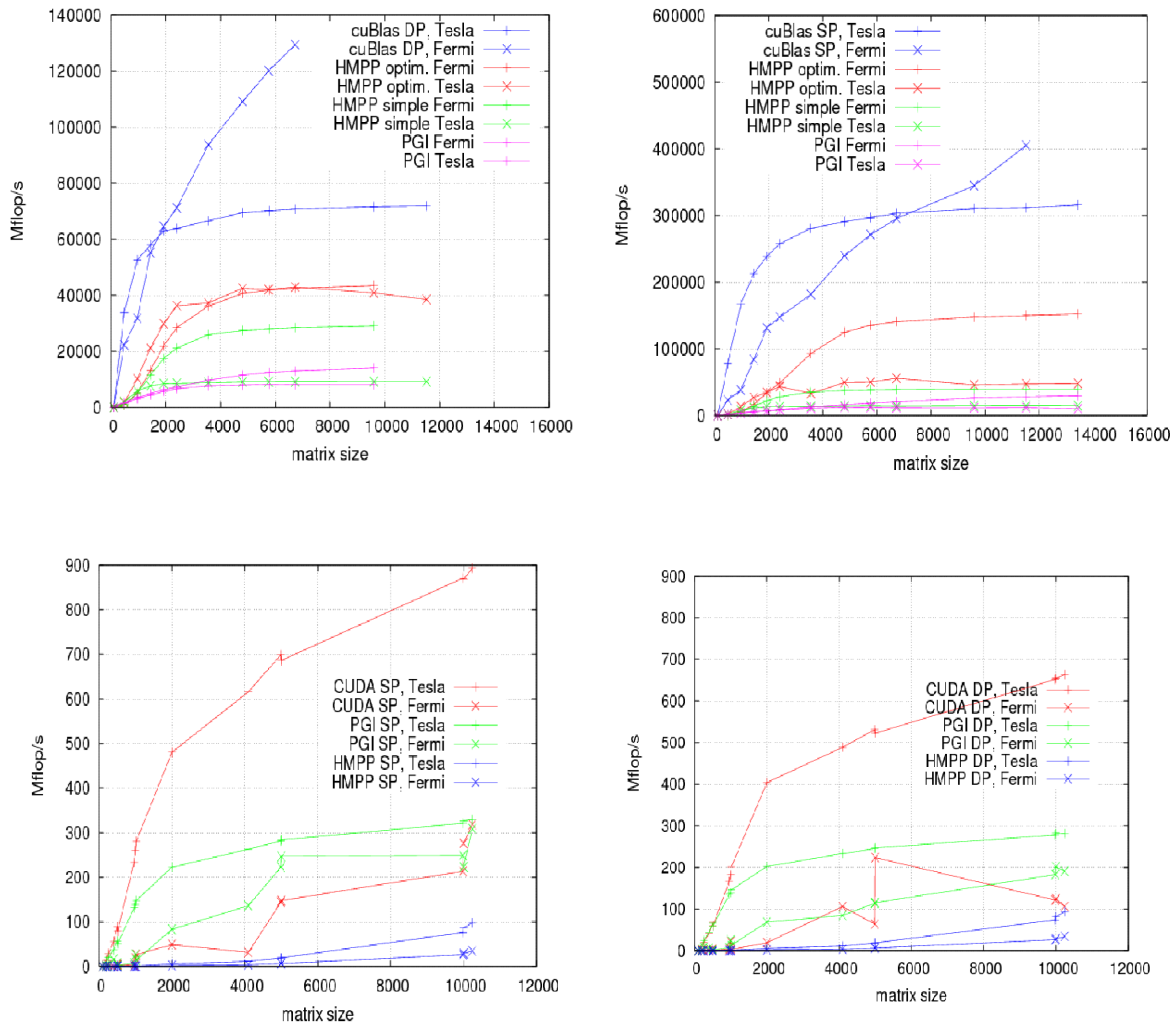


**Figure 11 Comparison of implementations using HMPP and the PGI compiler with CUDA versions. The upper row shows the performance for mod2am, the lower row the performance for mod2as, both for single (left) and double precision (right).**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | HMPP runtime library can dispatch computations on multi-GPU systems. | - |
| Performance | - | Rather low compared with CUDA. |
| Productivity | Main strength of HMPP. Simple OpenMP-like directives enable rapid development. | Productivity is always a combination of development time (+) and performance (-). |
| Sustainability | PathScale and CAPS joined to establish the HMPP Open Standard. | Unclear whether the “HMPP Open Standard” will be implemented by other vendors and accepted by the community. |
| Correctness | Much less error-prone than CUDA. | - |
| Portability | High. Code generators for CUDA and OpenCL (NVIDIA & AMD ATI) are | - |

| | | |
|---|---|---|
| | available | |
| Availability | Both Linux and Windows are supported. | Very high license costs for CAPS HMPP. |

**Table 15 HMPP pros and cons**

### 3.1.5 *Portland Group (PGI) Fortran and C compiler for accelerators*

**Description**

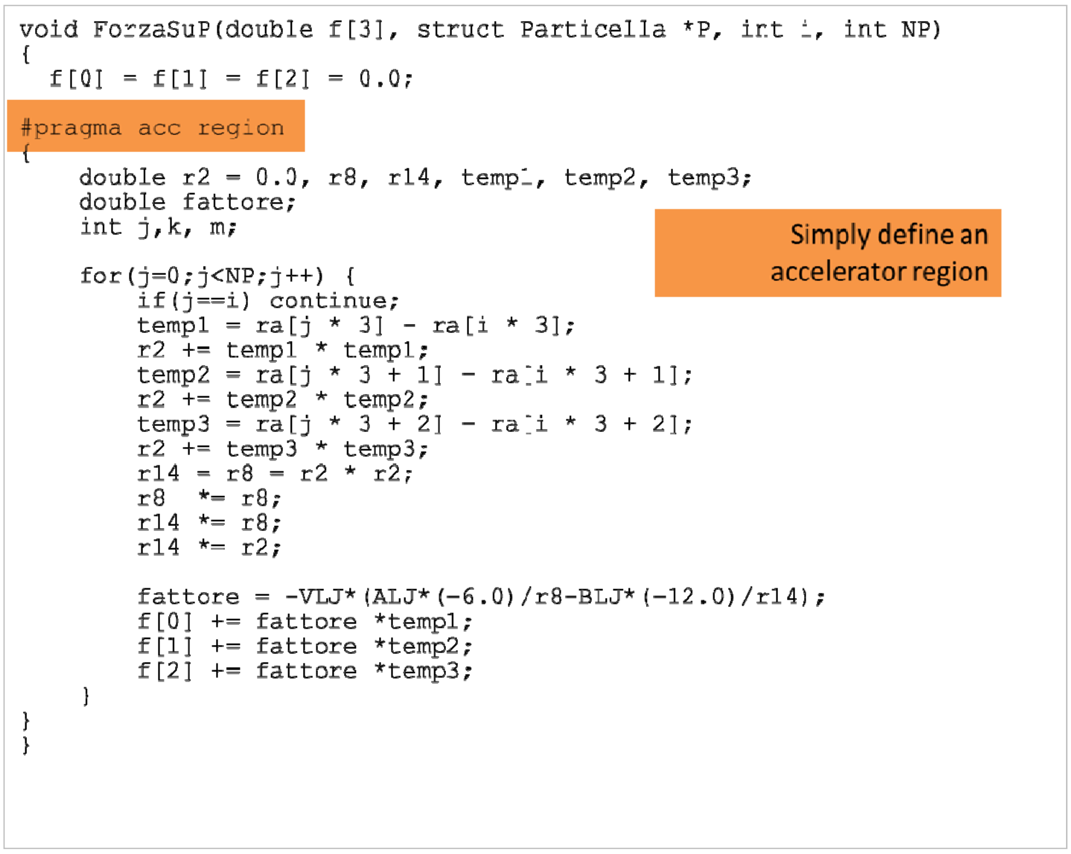
The PGI compilers were among the first to include a backend for C and Fortran targeting GPUs. The idea is to simplify porting existing C or Fortran codes without the need to rewrite the code in a language for the accelerator, e.g. CUDA. To date, all NVIDIA CUDA enabled GPUs are supported, ranging from NVIDIA CUDA architecture 1.0 (GeForce GTX 8800) to 2.0 (Fermi architecture). A particular architecture is specified by compiler command line options.

Directives must be used to identify those portions of a program to be executed on the accelerator. In C, these directives are pragmas, and in Fortran comments with a specific syntax. For example, C pragma “#pragma acc region” tells the compiler that the code block following the pragma shall be compiled and executed on the accelerator rather than the host. The general syntax of an accelerator directive for C is:

```
#pragma acc directive-name [clause [,clause]…] new-line
```

In Fortran, directives are specified in free-form source files as

```
!$acc directive-name [clause [,clause]…]
```



```
void ForzaSuP(double f[3], struct Particella *P, int i, int NP)
{
  f[0] = f[1] = f[2] = 0.0;
#pragma acc region
{
    double r2 = 0.0, r8, r14, temp1, temp2, temp3;
    double fattore;
    int j,k, m;

    for(j=0;j<NP;j++) {
        if(j==i) continue;
        temp1 = ra[j * 3] - ra[i * 3];
        r2 += temp1 * temp1;
        temp2 = ra[j * 3 + 1] - ra[i * 3 + 1];
        r2 += temp2 * temp2;
        temp3 = ra[j * 3 + 2] - ra[i * 3 + 2];
        r2 += temp3 * temp3;
        r14 = r8 = r2 * r2;
        r8  *= r8;
        r14 *= r8;
        r14 *= r2;

        fattore = -VLJ*(ALJ*(-6.0)/r8-BLJ*(-12.0)/r14);
        f[0] += fattore *temp1;
        f[1] += fattore *temp2;
        f[2] += fattore *temp3;
    }
}
}
```

**Figure 12 PGI C code example**

Since accelerators have their own memory, the compiler generates the code to copy data from host memory to GPU memory. Similar to CAPS HMPP, the programmer can choose various optional features within the pragma to control the data transfer.

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | - | The performance relative to hand coded CUDA will be multiplied by the number of accelerators in the cluster |
| Performance | - | Lower than hand coded CUDA |
| Productivity | It’s the main strength of the product, the idea is to have minimal changes to existing code | Handling pragmas in the best way for the compiler can require time and effort, less than hand coded CUDA, but good performance is not automatically achieved |
| Sustainability | The accelerators part of the compiler is only one of the many features, so it’s not a huge risk for PGI to support them | - |
| Correctness | Using pragmas and letting the compiler deal with the details of CUDA programming is less error-prone than hand coded CUDA | - |
| Portability | Theoretically the compiler could generate code for any kind of accelerator (AMD, FPGA, and so on) | Currently only CUDA enabled devices are supported |
| Availability | Currently available | - |

**Table 16 PGI compiler pros and cons**

### 3.1.6 *OpenMP+MPI*

**Description**

Today, mixing OpenMP and MPI constitutes the de-facto standard among the programming environments for petascale machines. Programs are parallelized in two stages. OpenMP is used to program the cores within a node, and MPI to communicate across nodes. Compared to using MPI within a node, OpenMP exploits the shared-memory architecture of the nodes better, and incurs lower communication and synchronization overheads. This programming paradigm has been described in detail in PRACE-PP deliverable D6.6 [48].

**Experience & Results**

In this section we report performance results of an OpenMP/MPI CFD program, developed and measured at IDRIS.

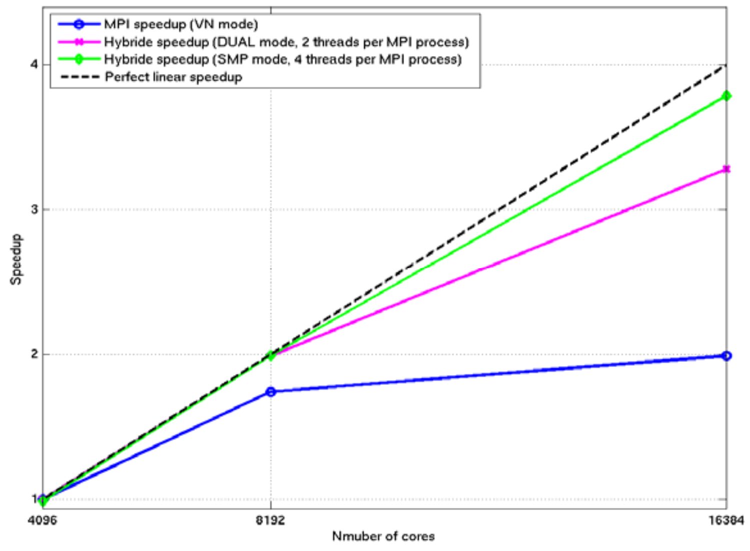


**Figure 13 OpenMP+MPI speedup of HYDRO on BG/P**

HYDRO is a 2D Computational Fluid Dynamics code (~1500 lines), that solves Euler's equation with a Finite Volume Method using Godunov's scheme and a Riemann solver at each interface on a regular mesh. The HYDRO code was ported to the OpenMP+MPI paradigm. We used a two-dimensional domain decomposition for the MPI parallelization and a coarse-grained OpenMP parallelization. The results presented here were run on the Blue Gene/P system at IDRIS. On small to relatively high number of cores, the performances of the pure MPI code and the hybrid MPI+OpenMP approach are very similar. But, once the number of cores is over 4096, the pure MPI implementation begins to lose scalability, whereas the hybrid approach keeps a near perfect scalability.

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | Excellent. The best that can be achieved on Petascale architectures. The two levels of parallelism perfectly fit the hardware characteristics of various machines (either fat nodes or thin nodes). | - |
| Performance | Excellent. | - |
| Productivity | Variable. It strongly depends on the characteristics of the code. It is relatively easy to add OpenMP directives to an existing MPI program, but results in terms of performance and scalability may be poor. | Development time to get an optimized and scalable application could be very high. |
| Sustainability | Very high (OpenMP and MPI are standards which are widely used). | - |
| Correctness | - | Combining two levels of parallelization can leads to more and more complex bugs. The lack of robust and mature tools to debug such a code is very penalizing. |
| Portability | Very portable (based on standards). | - |
| Availability | Nearly everywhere (just needs an OpenMP compiler and a MPI library). | - |

**Table 17 OpenMP+MPI pros and cons**

### 3.1.7 *Coarray Fortran (CAF)*

**Description**

Coarray Fortran (CAF) [6] is a Partitioned Global Address Space (PGAS) language. PGAS languages promise to provide ease of programming and high performance on platforms with shared and distributed address spaces. The integrated CAF compiler of the Cray compiler framework [8] has been evaluated during PRACE-PP [48]. Here, we highlight recent developments in the compiler, runtime system, and tools for CAF code development. CAF has been considered as part of the FORTRAN 2008 standard with minimal extensions to the Fortran syntax along with synchronization and control constructs. We note that the Rice compiler for CAF [53], extends the language even further.

CAF extends Fortran with *coarrays*, which are data structures shared between different *images* of a program. Accesses to coarrays result in remote memory accesses. Some interconnection networks offer hardware support for remote memory accesses that are far more efficiently than exchanging data with MPI. Moreover, since CAF compilers are interoperable with MPI, existing MPI programs can be ported incrementally to exploit the benefits of CAF implementations.

The array declarations below illustrate the CAF syntax. The first line is a regular Fortran declaration of array A. The CAF compiler creates a private copy of A for each CAF image. In contrast, CAF array A_caf is distributed across the given number of images.

```
DOUBLE PRECISION A(ndim)
DOUBLE PRECISION A_caf(ndim)[*]
```

Recent developments of the CAF language include the following: Intel has made available the beta version of its Intel CAF compiler [31]. Cray has produced a performance-optimized version of the integrated CAF compiler for the Cray XE6 platform with Gemini interconnect, and included a limited set of features for performance evaluation of CAF codes into its Cray performance toolset. The beta version of the Intel CAF compiler is an important development as it could be the first multi-platform compiler for CAF. Hence, we should watch both the Cray CCE and Intel CAF compilers concerning performance and scalability improvements and as a vehicle for functional evaluation of CAF features. Furthermore, the Cray XE6 Gemini interconnect has been optimized for remote memory access (RMA) operations. Cray has implemented a proprietary interface called DMAPP, replacing the public-domain GASNet runtime [14] that was available on Cray XT series systems.

**Code example: 2D dense matrix-matrix multiplication**

```
$ derived from Rice CAF 1.0 compiler test suite
  double precision :: a(:,:)[:,:]
  double precision :: b(:,:)[:,:]
  double precision :: c(:,:)[:,:]
  double precision :: buffa(:,:)[:]
  double precision :: buffb(:,:)[:]

$ more initialization ...
     buffa(:,:) = a(:,:)[myRow,col]
     buffb(:,:) = b(:,:)[col, myCol]
     do i=1, blockSize
        do j=1, blockSize
           do k=1, blockSize
              c(i,j) = c(i,j)-buffa(i,k)*buffb(k,j)
           end do
        end do
     end do
...
```

**Experience & Results**

In this section, we present benchmark results for CAF versions of the Euroben kernels mod2am/MxM and mod2f/FFT on Cray systems, measured in MFlop/s at CSCS.

We have evaluated both the Cray CCE CAF compiler and the beta version of the Intel CAF compiler. Table 18 and Table 19 list the benchmark results in MFlop/s for two CAF benchmarks that have been measured during the preparatory project on the Cray XT5 and Cray XE6 platforms. The XT5 has dual 6-core nodes, operated at 2.4 GHz, with DDR2 memory and SeaStarII interconnect. The XE6 offers an interconnect optimized for remote memory access operations. The XE6 has dual 12-core nodes, operated at 2.1 GHz, with DDR3 memory and Gemini interconnect.

| **Number of CAF images** | **Cray XT5** | **Cray XE6** |
|---|---|---|
| 4 | 24588.904 | 6097.445 |
| 16 | 56253.845 | 26012.030 |
| 64 | 21143.744 | 53266.750 |

**Table 18 CAF performance of mod2am/MxM in MFlop/s for 400 x 400 matrices**

| **Number of CAF images** | **Cray XT5** | **Cray XE6** |
|---|---|---|
| 4 | 3026.223 | 2710.906 |
| 8 | 4087.747 | 5067.822 |
| 16 | 2807.321 | 9286.350 |
| 32 | 1019.839 | 13247.659 |
| 64 | 502.681 | 15049.973 |

**Table 19 CAF performance of mod2f/FFT in MFlop/s (bits=16 and length=65536)**

The Intel CAF compiler has been gradually improving allowing development and execution of complex CAF applications. The execution model is in progress allowing for jobs running in distributed memory nodes. Overlapping of computation and communication is still work in progress.

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | With appropriate runtime support for RMA and fine-grain synchronization operations, the codes can scale to large number of cores. | Runtime and compiler optimization are work in progress. The higher abstraction level for ease of programming is an issue. Interoperability standards with other |

| | | |
|---|---|---|
| | | programming models such as MPI could also limit scalability. |
| Performance | With appropriate hardware and runtime support, performance and scaling has improved significantly | Locality issues on multi-socket multi-core systems are work in progress |
| Productivity | Smaller set of Fortran extensions offer high productivity to language developers. A higher abstraction level for parallel programming reduces the entry barrier. | Restricted language syntax, which only allows for data parallel programming. Immature tool support. Performance efficiencies very low. |
| Sustainability | Part of the Fortran language standard | Very few Fortran compiler developers have it on their roadmaps |
| Correctness | - | - |
| Portability | Beta compiler from Intel can be ported to any x86 platform | Very few Fortran compiler developers have it on their roadmaps. Extensions and availability to heterogeneous systems unclear. |
| Availability | - | No stable open-source compiler version available |

**Table 20 CAF pros and cons**

### 3.1.8 *Unified Parallel C (UPC)*

**Description**

UPC [63] is a Partitioned Global Address Space (PGAS) language. PGAS languages shall provide ease of programming and high performance on platforms with shared and distributed address spaces. The Integrated UPC compiler is part of the Cray compiler framework [8], and has been evaluated during PRACE-PP [48]. Here, we highlight recent developments in the compiler, runtime system, and tools for UPC code development. UPC is an extension to the C language, offering the benefits of the PGAS model to programs written primarily in C. UPC compilers are compliant to a UPC specification that is not part of the ANSI C standard. A number of UPC compilers are available for parallel multi-core systems, including the UPC CCE compiler from Cray and Berkley UPC [2].

UPC programs instantiate threads and data are shared among or private to threads. Qualifier keywords are used to declare whether data are shared and how arrays could be distributed among threads. The number of threads can be specified at compile time or runtime. Although the UPC language specification does not address the issue of interoperability with other programming environments, existing UPC compilers interoperate with other language like Cilk, for example.

The array declarations below illustrate the UPC syntax. The first line is a regular C declaration of array A. The UPC compiler creates a private copy of A for each thread. In contrast, array A_upc is distributed across the given number of threads.

```
double A[ndim];
shared double A_upc[ndim];
```

**Code example: 2D dense matrix-matrix multiplication**

```
/* GWU UPC matrix multiply */
shared [N*P /THREADS] int *a;
shared [N*M /THREADS] int *c;
shared [M   /THREADS] int *b;

a = upc_all_alloc(THREADS,(N*P/THREADS)* upc_elemsizeof(*a));
c = upc_all_alloc(THREADS,(N*M/THREADS)* upc_elemsizeof(*c));
b = upc_all_alloc(P*THREADS,(M/THREADS)* upc_elemsizeof(*b));

upc_forall(i = 0 ; i<N ; i++; &c[i*M]) {
  for (j=0 ; j<M ;j++) {
    c[i*M+j] = 0;
    for (l= 0;l<P; l++)
      c[i*M+j] += a[i*P+l]*b[l*M+j];
  }
}
```

**Experience & Results**

In this section, we present benchmark results in MFlop/s for UPC versions of the Euroben kernels mod2am, mod2as/SpMV, and mod2f/FFT on Cray systems, measured at CSCS.

We have evaluated the Cray CCE UPC compiler and beta versions of the code development and performance toolset. Benchmark results in MFlop/s for three UPC benchmarks that have been developed during PRACE-PP for modam/MxM, mod2as/SpMV and mod2f/FFT on the Cray XT5 and Cray XE6 platforms. The XE6 offers an interconnect optimized for remote memory access operations, that UPC should benefit from.

| Number of UPC threads | Cray XT5 | Cray XE6 |
|---|---|---|
| 4 | 10237.51 | 25731.66 |
| 16 | 16262.74 | 79492.82 |
| 64 | 7286.96 | 162179.28 |

**Table 21 UPC performance of mod2am/MxM in MFlop/s for 800 x 800 matrices**

| Number of UPC threads | Cray XT5 | Cray XE6 |
|---|---|---|
| 4 | 1578 | 1031 |
| 8 | 2093 | 1905 |
| 16 | 3439 | 3572 |
| 32 | 15402 | 8542 |
| 64 | 16110 | 17500 |

**Table 22 UPC performance of mod2as/SpMV in MFlop/s for 10000 x 10000 matrices (3.5% fill)**

| Number of UPC threads | Cray XT5 | Cray XE6 |
|---|---|---|
| 4 | 1963.51 | 2744.29 |
| 8 | 2756.52 | 5018.32 |
| 16 | 1235.76 | 9381.20 |
| 32 | 579.33 | 11835.47 |
| 64 | 258.80 | 9643.86 |

**Table 23 UPC performance of mod2f/FFT in MFlop/s (bits=16 and length=65536)**

**Pros & Cons**

| | Pros | Cons |
|---|---|---|
| Scalability | With appropriate runtime support for RMA and fine-grain synchronization operations, the codes can scale to large numbers of cores. | Runtime and compiler optimization are work in progress. The higher abstraction level for ease of programming is an issue. Interoperability standards with other |

| | | |
|---|---|---|
| | | programming models such as MPI could also limit scalability. |
| Performance | With appropriate hardware and runtime support, performance and scaling has improved significantly. | Locality issues on multi-socket multi-core systems are work in progress. Scaling of the portable UPC compiler on different distributed memory clusters is an issue. |
| Productivity | Smaller set of UPC extensions and collective APIs offer high productivity to language developers. A higher abstraction level for parallel programming reduces the entry barrier. | Restricted language syntax, which only allow for data parallel programming. Immature tool support. Performance efficiencies very low. |
| Sustainability | A UPC standard is available | Very few compiler developers have UPC support on their roadmaps |
| Correctness | - | - |
| Portability | Berkley UPC compiler is portable to a large number of clusters. | Very few compiler developers have UPC support on their roadmaps. Extensions and availability to heterogeneous systems unclear. |
| Availability | Stable open-source (functional) compiler version available | - |

**Table 24 UPC pros and cons**

### 3.1.9 *Chapel*

**Description**

Chapel is a new parallel programming language developed as part of the DARPA HPCS project by the University of Washington and Cray. See the PRACE-PP deliverable D6.6 [48] for a detailed description of the important language concepts. Here, we discuss significant developments and new concepts of Chapel.

Chapel can be classified as a Partitioned Global Address Space (PGAS) language. The global view on distributed arrays is tightly integrated into the language, however, in contrast to language extensions like CAF or UPC. In fact, for the Chapel programmer, distributed and local arrays can behave exactly the same, if desired. Of course, the programmer is expected to exploit the data-parallelism inherent in using arrays. Besides data-parallelism, Chapel also supports task-parallelism. Tasks may be created locally or on remote hosts, either as an implicitly synchronized team or with explicitly programmed synchronization.

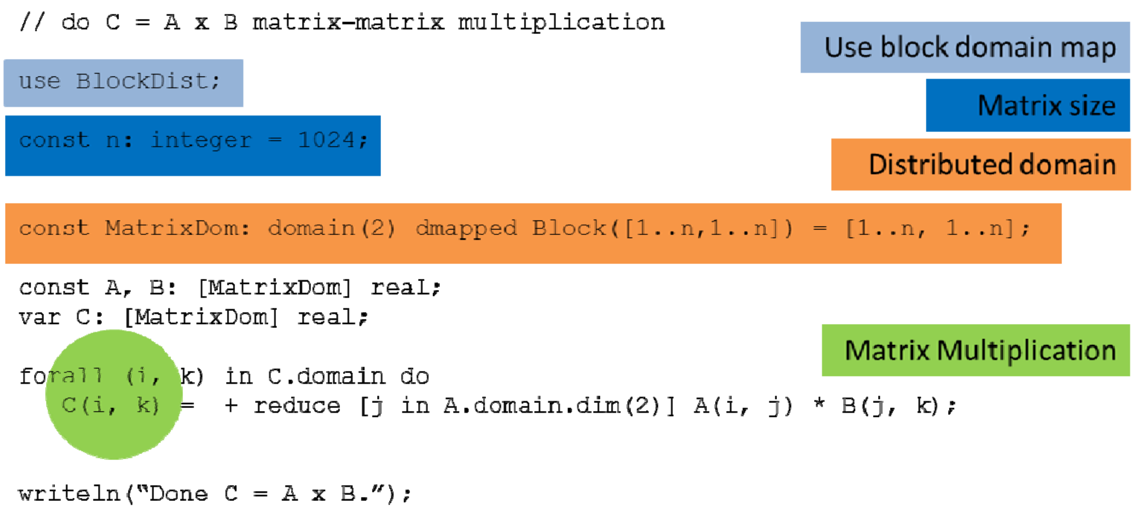
The memory mapping is represented through Chapel's *domain map* concept, formerly also called *distribution*. A domain map controls how a given set of indices, e.g. of an array, is distributed across the machine. In the default domain map all indices are local. Chapel enables the programmer to write interfaces for domain maps and to create arbitrarily complex domain maps. Chapel's standard module library includes domain maps for block, cyclic, and block-cyclic distributions. Recently, it also offers a domain map for replicating data on all hosts. All arrays (or other data collections/containers) are associated with a domain map. If no domain map is specified the default domain map applies. Other than in the declarations, the program does not distinguish distributed from local arrays.

Since the initial evaluation of Chapel during PRACE-PP the language and the compiler/runtime have evolved significantly. One of the most notable areas of improvement is interoperability with other languages. Chapel now supports calling C functions, converting to/from C data-types and using native C data-types. In principle, this permits linking C libraries such as LAPACK, BLAS, or MKL. However, it is unclear as of now, how regular C functions access distributed data. Chapel has not been designed to interoperate with other parallel programming models. However, as long as the model is implemented as a library, such as MPI, Chapel could be used in a hybrid setting.

We are not aware of strong tool support for Chapel. It is possible to debug Chapel code with any debugger supporting the GNU gdb interface. There are also some Chapel-specific command-line tools to monitor communication events and remote tasks spawning within Chapel programs. But to our knowledge, none of the common performance and debugging tools supports Chapel.

The most recent version of Chapel (V1.2.1 as of March 2011) is quite mature and produces much faster scalar code than its predecessors. The most notable improvement is that all domain maps support multi-threaded traversals of arrays on a single node and also across nodes. However, they are not yet optimized for performance. In particular, RMA transfer across nodes is very inefficient. The Chapel team is aware of the RMA problem and currently working on it. In general the focus of the development work appears to be shifting from implementing new language concepts or prototyping new library modules to performance optimizations of the Chapel compiler and runtime. It is not possible, however, to estimate when this process will lead to acceptable performance and scaling for the Euroben kernels or other HPC applications.

**Code example: non-optimized, non-blocked matrix-matrix multiplication**



```
// do C = A x B matrix-matrix multiplication

use BlockDist;

const n: integer = 1024;

const MatrixDom: domain(2) dmapped Block([1..n,1..n]) = [1..n, 1..n];

const A, B: [MatrixDom] real;
var C: [MatrixDom] real;

forall (i, k) in C.domain do
   C(i, k) = + reduce [j in A.domain.dim(2)] A(i, j) * B(j, k);

writeln("Done C = A x B.");
```

**Experience & Results**

The conclusions of the Chapel evaluation at HLRS are as follows.

Remote memory accesses are implemented inefficiently, and the primary reason why the Euroben kernels do not perform well. This situation has not changed since the initial evaluation. Tests with the latest Chapel compiler release on the Gemini interconnect of the Cray XE6 exhibit improved performance due to very low network latency. However, our benchmarks do not scale beyond an unacceptably small number of nodes.

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | Task spawning and synchronization across the machine seems to be reasonably efficient. | RMA transfer very inefficient; does not allow to scale communication intensive applications |
| Performance | Scalar performance has improved significantly; in particular as efficient libraries may be used. | - |
| Productivity | Very short, readable code. Easy to program and | Practically no tools support. |

| | | |
|---|---|---|
| | maintain. Clear and powerful concepts for parallel programming. | |
| Sustainability | - | Will Cray continue to support it? |
| Correctness | - | - |
| Portability | May run on a number of network conduits (e.g. MPI, GASNET, native). Compiler is standard C code and can be ported onto any Unix-like platform. | - |
| Availability | Open source with direct access to development SVN. | - |

**Table 25 Chapel pros and cons**

### 3.1.10 *StarSuperscalar (StarSs) Programming Model*

**Description**

StarSs is a multicore programming model with functional parallelism specified by annotating sequential applications. In principle, StarSs supports a wide range of devices like, Cell (CellSs), GPU (GPUSs), SMP (SMPSs), and can also be mixed with MPI for hybrid programming. ClusterSs, which is currently under development, is the combination of the StarSs programming model with the GASNet communication layer allowing a multicore program to be run on distributed machines.

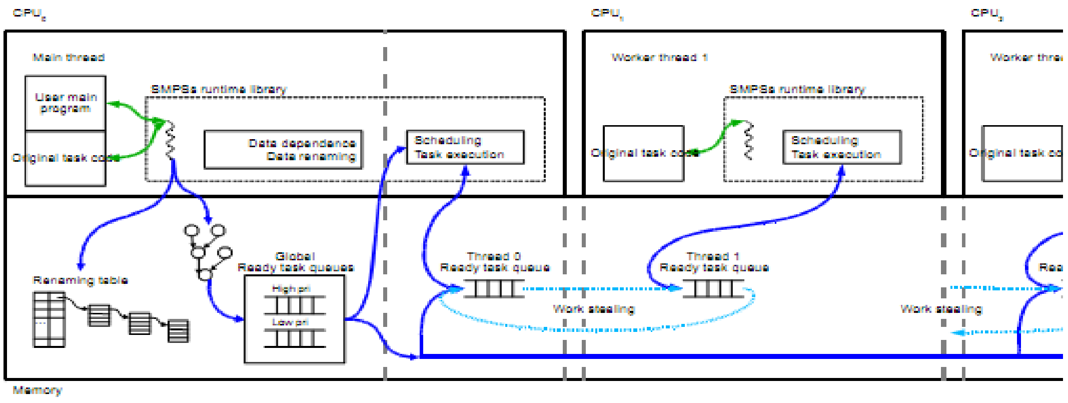


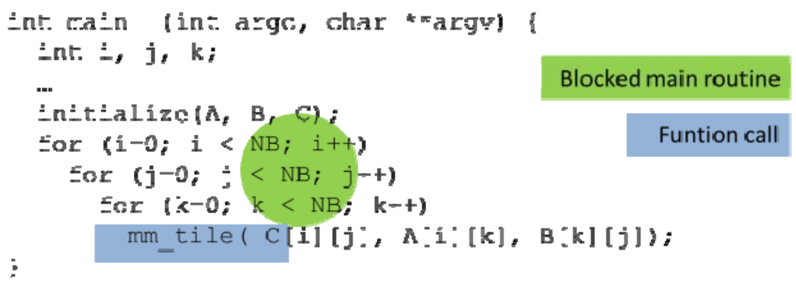
**Figure 14 SMPSs Implementation**

StarSs is implemented by means of a source-to-source compiler and a runtime library. Given a sequential application in C or Fortran with StarSs annotations, the source-to-source compiler generates output files in the target languages. Compiling an annotated program with CellSs generates a source specific to Cell, whereas compiling it with SMPSs or GPUSs will generate sources for an SMP or GPU, respectively. Figure 14 shows the process flow to generate executables on an SMP based machine

StarSs syntax is based on code annotation by means of pragmas. A sample matrix multiplication with pragmas is shown as code example below.

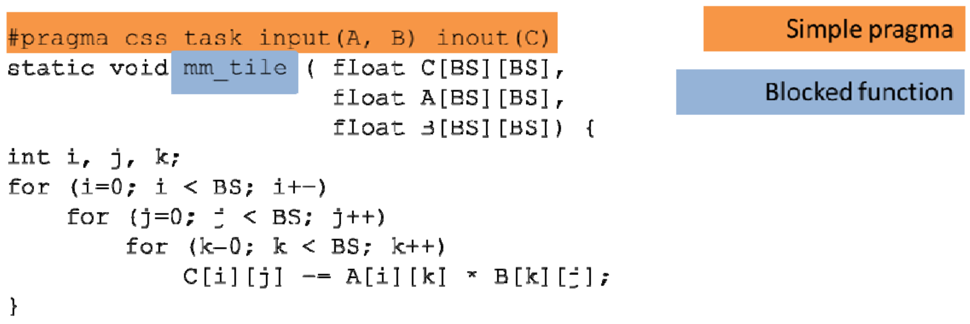
StarSs is interoperable with standard libraries like BLAS or LAPACK. Currently, the CEPBA tool (developed at BSC) can be used for performance analysis. The Extrae tool has been developed to generate traces for hybrid StarSs+MPI programs. The gdb debugger can be used to debug the C program; however the user has to take care of data dependencies herself.

**Code example**



```
int main  (int argc, char **argv) {
  int i, j, k;
  ...
  initialize(A, B, C);
  for (i=0; i < NB; i++)
    for (j=0; j < NB; j++)
      for (k=0; k < NB; k++)
        mm_tile( C[i][j], A[i][k], B[k][j]);
}
```

**Figure 15 StarSs blocked main routine with function call.**



```
#pragma css task input(A, B) inout(C)
static void mm_tile ( float C[BS][BS],
                      float A[BS][BS],
                      float B[BS][BS]) {
int i, j, k;
for (i=0; i < BS; i++)
    for (j=0; j < BS; j++)
        for (k=0; k < BS; k++)
            C[i][j] += A[i][k] * B[k][j];
}
```

**Figure 16 StarSs function with pragma.**

**Pros & Cons**

| | Pros | Cons |
|---|---|---|
| Scalability | Scales well across the cores. | Currently limited to shared memory systems, though a version for distributed memory is under development. |
| Performance | Performance has been high/acceptable. | Performance is limited to capabilities of user in creating data dependency and extracting parallelism. |
| Productivity | Same code can be run across multiple platforms, with minor modifications in the source code based on the architecture. | Development time is affected to some extent due to unavailability of debuggers. |
| Sustainability | StarSs has been actively developed at BSC | - |
| Correctness | - | Care must be taken making sure that data dependencies are correct. |
| Portability | Easily portable across several architectures as the source code remains the same. | - |
| Availability | Available as free and open source for different languages under StarSs programming model. | - |

**Table 26 StarSs pros and cons**

### 3.1.11 *Cilk*

**Description**

Cilk [36] is an algorithmic multithreaded language with a provably efficient runtime system suited for divide-and-conquer style programming with a weak shared memory model. The programmer expresses parallelism explicitly by prepending keyword *spawn* to function calls and by synchronizing parent and child threads with keyword *sync*. Cilk's work-stealing scheduler automates the tasks of process placement and load-balancing a computation.

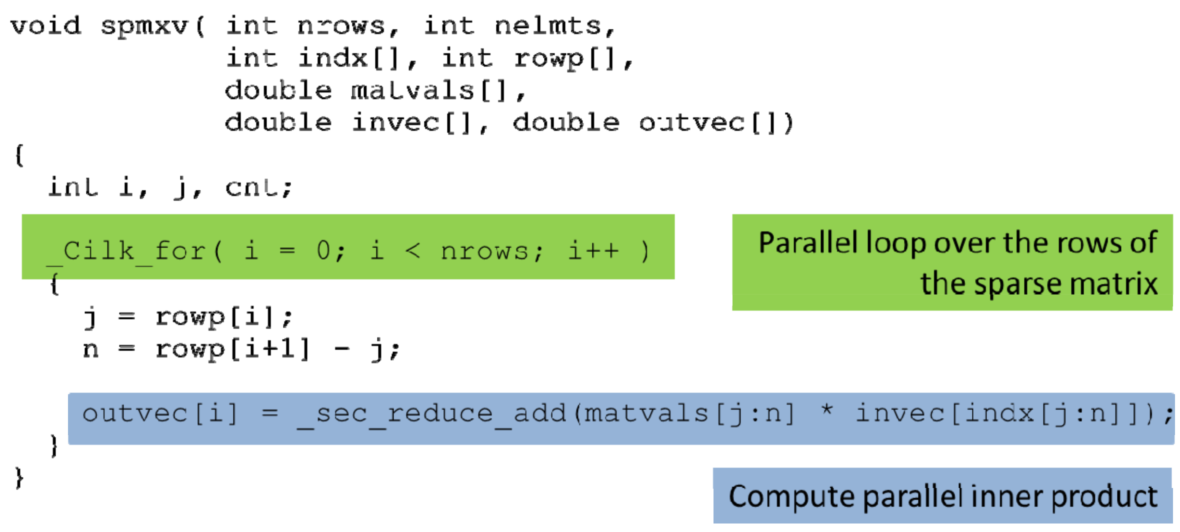
Since the serial semantics of a Cilk program equals that of the sequential C program without spawn's and sync's, familiar debuggers like gdb can be used. The Cilkscreen race detector tool is available for monitoring and detecting unintended data races.

Cilk emerged as a research project at MIT. Intel supports the CilkPlus variant for C and C++ as part of its Parallel Building Blocks [19]. CilkPlus offers additional programming features:

1. a parallel loop construct “Cilk_for”, which executes each loop iteration in parallel with the others,
2. so called “Hyper Objects” which offer consistent views of non-local variables and simplify mitigating races without creating lock contention,
3. full C++ exception support,
4. and a library for mutex-locks.

**Code example**

The code example shows the kernel of the Euroben kenel mod2as/SpMV using Intel CilkPlus:



```
void spmxv( int nrows, int nelmts,
            int indx[], int rowp[],
            double matvals[],
            double invec[], double outvec[])
{
  int i, j, cnt;
  _Cilk_for( i = 0; i < nrows; i++ )
  {
    j = rowp[i];
    n = rowp[i+1] - j;

    outvec[i] = _sec_reduce_add(matvals[j:n] * invec[indx[j:n]]);
  }
}
```

Parallel loop over the rows of the sparse matrix
Compute parallel inner product

The parallel inner product is computed using the parallel “+”-reduction and an element-wise product of the j th column of sparse matrix `matvals` and dense vector `invec`.

**Experience & Results**

In this section, we report benchmark results of the Euroben kernels transliterated into Cilk and measured at JKU.

Porting the Euroben benchmarks mod2am/MxM and mod2as/SpMV to Cilk required a programming effort of a few hours. Instead of transliterating mod2f/FFT into Cilk, we measured the 1D-FFT program included in the MIT Cilk distribution. We present results produced with the open-source MIT Cilk version and Intel's CilkPlus.

| | **n-processor runtime (sec)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Benchmark | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
| mod2am-blas | | | | | | | | | |
| 1K x 1K | 0.65 | 0.32 | 0.19 | 0.12 | 0.09 | 0.09 | - | - | - |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2K x 2K | 4.13 | 2.31 | 1.25 | 0.67 | 0.57 | 0.50 | - | - | - |
| 8K x 8K | 247 | 127.1 | 73.1 | 38.7 | 26.3 | 18.73 | 16.3 | 14.98 | - |
| 16K x 16K | 2614 | 1309 | 678 | 319.9 | 181.7 | 90.53 | 44.9 | 23.34 | 12.94 |
| Mod2as | | | | | | | | | |
| 10K, 5.7% fill | 0.18 | 0.08 | 0.067 | 0.039 | 0.034 | 0.029 | - | - | - |
| 100K, 4.7% fill | 2.83 | 1.33 | 0.93 | 0.56 | 0.46 | 0.41 | - | - | - |
| 1D fft | | | | | | | | | |
| 2M | 0.53 | 0.353 | 0.193 | 0.121 | 0.102 | 0.107 | - | - | - |
| 16M | 6.79 | 5.15 | 2.87 | 1.59 | 1.13 | 1.07 | - | - | - |

**Table 27 MIT Cilk results on an SGI Altix 4700, using 256 Itanium2 cores @1.6GHz**

| | n-processor runtime (sec) | | | |
|---|---|---|---|---|
| Benchmark | 1 | 2 | 4 | 8 |
| mod2am-mkl | | | | |
| 1K x 1K | 0.285 | 0.152 | 0.088 | 0.057 |
| 2K x 2K | 2.037 | 1.022 | 0.552 | 0.332 |
| 8K x 8K | 130.5 | 65.5 | 33.9 | 20.67 |
| 16K x 16K | 1046 | 522.2 | 274.0 | 166.7 |
| Mod2as | | | | |
| 10K, 5.7% fill | 0.012 | 0.007 | 0.005 | 0.004 |
| 100K, 4.7% fill | 0.243 | 0.139 | 0.085 | 0.068 |
| 1D FFT | | | | |
| 2M | 0.28 | 0.145 | 0.071 | 0.044 |
| 16M | 3.39 | 1.80 | 0.939 | 0.491 |

**Table 28 Intel CilkPlus results on an Altix ICE 8200 node, using 8 Nehalem-EP cores @2.8Ghz**

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | The Cilk scheduler is capable of delivering speedup close to the theoretical parallelism of the algorithm. The Cilkview scalability analyzer tool aids understanding the performance and scalability of a Cilk program.<br>Hybrid programming of Cilk together with MPI or UPC works. | Cilk is supported on cache coherent systems only and will not work in Exascale environments unless computer architectures can efficiently support shared memory at Exascale. |
| Performance | Cilk achieves near optimal performance in practice. Cilk interoperates with C/C++ libraries for processor-specific tuning. | Today's cache coherent shared-memory machines cause performance non-monotonicities and do not scale. |
| Productivity | Parallelizing divide-and-conquer programs requires relatively little programming effort. CilkPlus supports parallel loops. | Cilk is limited to divide-and-conquer parallelism. |
| Sustainability | Cilk development continues at MIT. Intel offers the commercial CilkPlus dialect. | Cilk is not mainstream. |
| Correctness | Reasoning about the correctness of a Cilk | - |

| | Pros | Cons |
|---|---|---|
| | program is no more difficult than reasoning about the correctness of its serialization. The Cilkscreen race detector tool supports the programmer in finding undesirable races. | |
| Portability | MIT Cilk ports to shared-memory machines where the GNU toolchain exists. | Intel Cilk Plus is available where Intel supports its toolchain. |
| Availability | MIT Cilk is free software, Intel distributes the CilkPlus dialect. | - |

**Table 29 Cilk pros and cons**

### 3.1.12 *Intel Array Building Blocks (ArBB)*

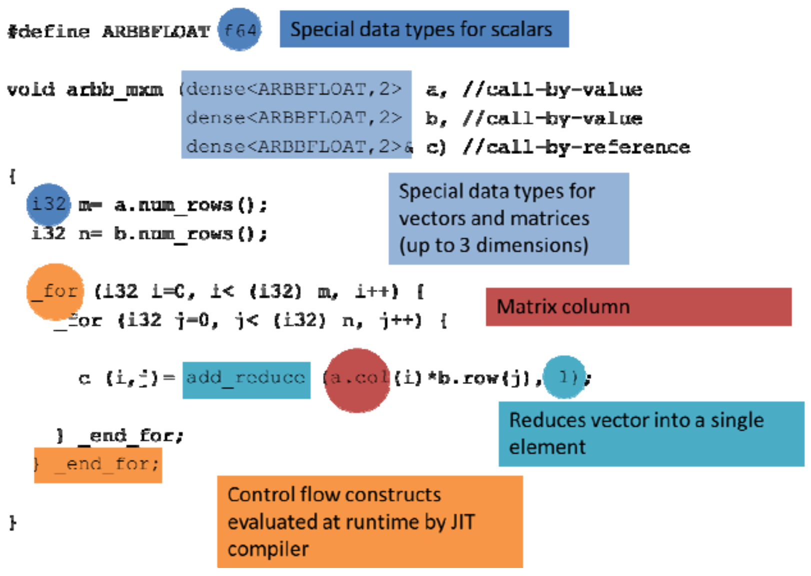
**Description**

*ArBB* is a high-level data parallel programming solution that frees application developers from dealing with low-level mechanisms of hardware architectures. ArBB will produce scalable, portable, and deterministic parallel programs from a single high-level, maintainable, and application-oriented implementation. Intel ArBB is a combination of RapidMind, which was acquired by Intel in late 2009, and Intel Ct, a former Intel research project. ArBB is currently available under public beta (ArBB v1.0 Beta 4).

ArBB hides parallelism from the user to simplify programming, and allows the user to focus on the data objects and their organization by adding special ArBB data types for vectors and matrices to C++. Furthermore, ArBB offers special operators and control-flow constructs. The compiler generates parallelism automatically. The ArBB runtime includes a JIT compiler for performance optimizations and for extracting parallelism.

ArBB uses standard C++ features, including templates and operator overloading, to create new data types and operators. Recent implementations are restricted to shared-memory systems, because they are based on pthreads, OpenMP, and TBB. However, this is not an inherent restriction. An MPI backend is under development at Intel to support ArBB on distributed systems, but will not become a product in near future. To date, no math library exists that uses ArBB data types. Since ArBB is interoperable with C++, it is possible to use standard libraries such as MKL. ArBB is supported by standard C++ debuggers such as gdb, and comes with supporting scripts for pretty printing ArBB scalars and dense containers, provides insight into opaque types, etc. We expect ArBB to be released as a product within 2011.

**Code example**



```
#define ARBBFLOAT f64

void arbb_mxm (dense<ARBBFLOAT,2> a, //call-by-value
               dense<ARBBFLOAT,2> b, //call-by-value
               dense<ARBBFLOAT,2>& c) //call-by-reference
{
  i32 m= a.num_rows();
  i32 n= b.num_rows();

  _for (i32 i=0, i< (i32) m, i++) {
    _for (i32 j=0, j< (i32) n, j++) {

      c (i,j)= add_reduce (a.col(i)*b.row(j), 1);

    } _end_for;
  } _end_for;

}
```

Special data types for scalars
Special data types for vectors and matrices (up to 3 dimensions)
Matrix column
Reduces vector into a single element
Control flow constructs evaluated at runtime by JIT compiler

**Experience & Results**

In this section, we report benchmark results for Euroben kernels mod2am/MxM and mod2as/SpMV, developed and measured at LRZ.

We have successfully ported mod2am/MxM and mod2as/SpMV to ArBB and compared performance to an MKL version and a naïve implementation on a Intel Core i7 CPU with 8 cores @ 2.67GHz.

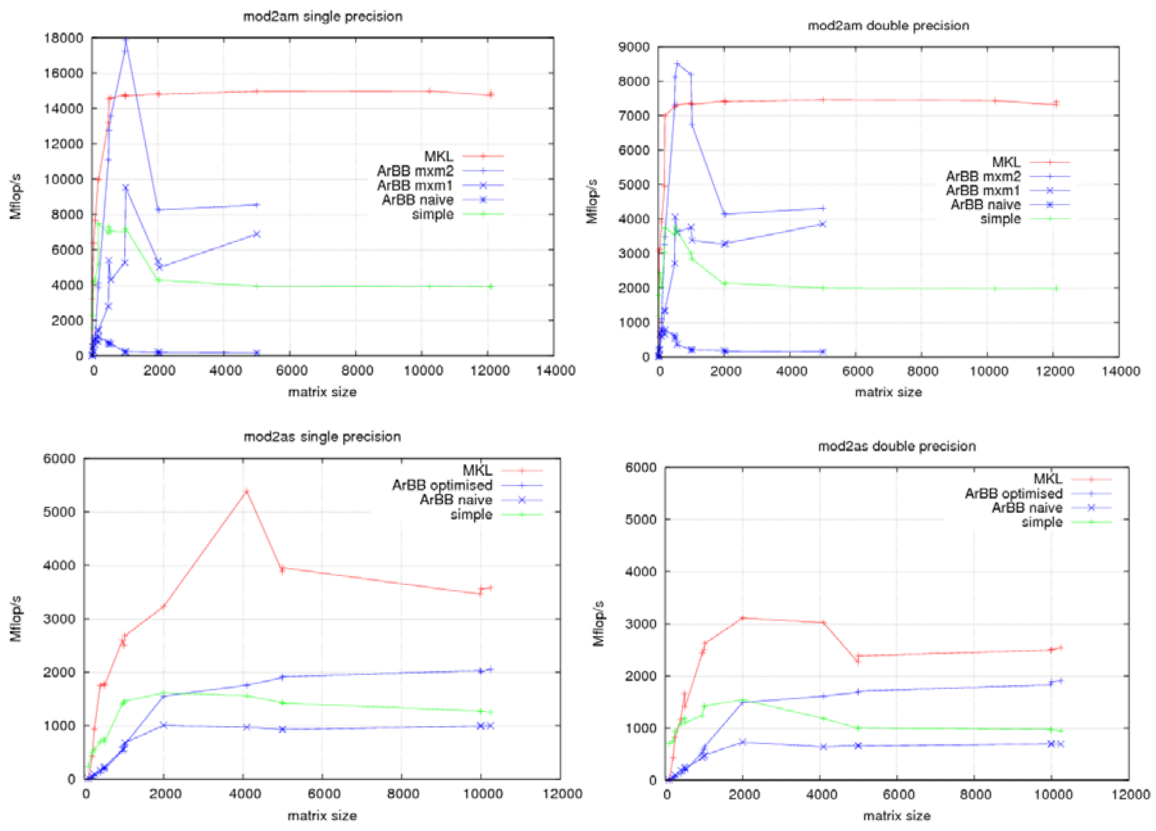


**Figure 17 Comparison of various ArBB implementations with MKL and a naïve implementation. The upper row shows the performance for mod2am/MxM, the lower row the performance for mod2as/SpMV, both for single (left) and double (right) precision.**

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | - | Limited to shared memory systems. Scalability needs improvements. |
| Performance | - | Measured performance is still low. Performance is limited by the capabilities of the compiler to extract enough parallelism to fully make use of SIMD units and available cores. |
| Productivity | Development time is rather low for people that are used to programming C++. Although the language is still in beta, we have not experienced compiler problems. | - |
| Sustainability | Intel is a big company with enough market share to introduce a new language. | Unclear. Depends on whether the language will be taken up by a larger circle. |
| Correctness | Since ArBB takes the burden of performance optimization from the programmer it is easier to write correct codes. | - |
| Portability | Currently limited to x86 architectures (from Intel and AMD). Performance portability should be high. AVX is supported. | RapidMind supported GPUs, Cell and multi-core architectures. Intel ArBB does not support GPUs (yet?). |
| Availability | Currently available under public beta (ArBB v1.0 Beta 4) free of charge. | - |

**Table 30 ArBB pros and cons**

### 3.1.13 *Intel Threading Building Blocks (TBB)*

**Description**

TBB is a shared-memory parallel library for developing efficient, scalable, and portable software that exploits the growing number of cores in modern CPUs. The library interface consists of a set of algorithms and containers, similar to those provided by STL. The computation will be divided recursively by the runtime into subtasks as long as their size is greater than a parameter called *grainsize*. Although the library has the ability to automatically carry out the partitioning, the developer is encouraged to provide the grainsize value. The actual operations to be executed inside a task are written as C++ methods, and are provided to the library constructs as a functor or a lambda (since TBB is fully C++0x compliant). Once the runtime builds the dependency tree, each task is scheduled on the available computing resources using the *work stealing* policy inherited from the Cilk project. A work stealing scheduler balances the work dispatched to the CPU cores in order to increase their occupancy and achieve the best scaling.

The latest TBB version, available for download from http://threadingbuildingblocks.org/, is available under two licenses: a commercial license with technical support and product updates and an open-source license, *GPLv2 with runtime exception*.

TBB promises to hide all the issues involved in writing massively parallel codes on shared memory architectures allowing the developer to avoid all error-prone and concurrency-limiting activities (like locks, mutexes or thread communication). In addition to this key feature, TBB allows to exploit parallelism on a broad set of architectures (x86, SPARC, Power), operating systems (Linux, Windows, AIX, Solaris, FreeBSD) and compilers in a portable manner since the burden of interfacing code to platform dependent concurrency libraries/paradigms is hidden by the library implementation. TBB provides a rich set of algorithm templates (loop, reduce, prefix scan, pipeline execution, sorting), STL-like containers (queue, vector, map, graph), memory allocators (all of them implemented extending std::allocator) and fine-grained synchronization control tools (better atomic operations and lock objects).

**Code example, matrix multiplication using parallel_for**

```
#include <tbb/parallel_for.h>
#include <tbb/blocked_range.h>

using namespace tbb;

#define DOMAIN_SIZE 1000

float A[DOMAIN_SIZE][DOMAIN_SIZE];
float B[DOMAIN_SIZE][DOMAIN_SIZE];
float C[DOMAIN_SIZE][DOMAIN_SIZE];

class MatMatMul
{
public:
    void operator()(blocked_range<int> task) const {
        for (task::iterator x = task.begin(); x != task.end(); ++x)
            for (int y = 0; y < DOMAIN_SIZE; ++y)
                for (int z = 0; z < DOMAIN_SIZE; ++z)
                    C[x][y] += A[x][z] * B[z][y];
    }
};

int main()
{
    parallel_for(blocked_range<int>(0,DOMAIN_SIZE),MatMatMul());
    return 0;
}
```

TBB namespace and headers are standard C++

The functor must implement the () operator!

The work range to be recursively split.

We use the parallel_for algorithm template.

Functor instance.

**Experience & Results**

In this section, we present results porting an MD code to TBB, developed and measured at CINECA.

We have ported a classical molecular dynamics algorithm with van der Waals interactions to TBB and used it to analyze its scaling potential achieving satisfactory results (see Figure 18). All tests have been carried out on a 4-core Intel Corei7 920 with 8 SMT cores.

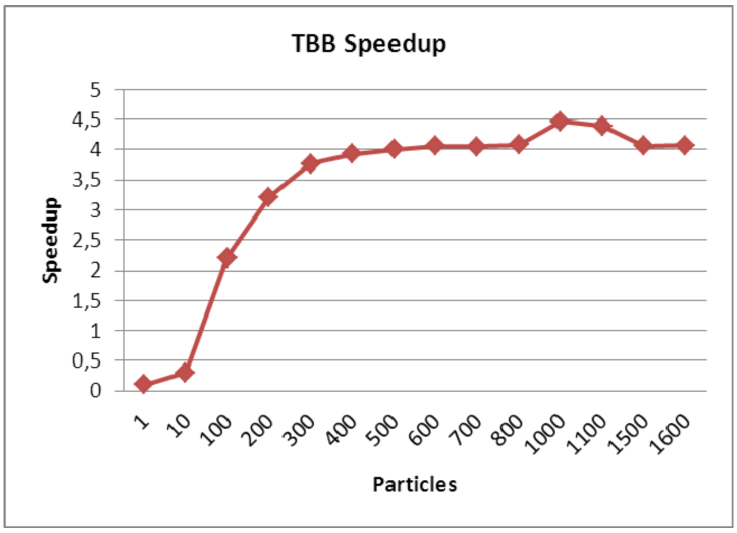


**Figure 18 TBB speedup of molecular dynamics code**

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | Task based parallelism upon task-stealing scheduler is meant to be highly scalable on a growing number of cores; the lock-free implementation helps in reaching the goal. | Interoperability (calling from the inside of functors/lambdas) with classic paradigms (MPI) should be verified. |
| Performance | The overhead introduced by the library is still low since its scheduling mechanism is meant to mask stall latencies. | - |
| Productivity | Development time is low for people that are used to C++ template meta-programming, STL algorithms and iterators. | Cannot be directly used from codes other than C++. |
| Sustainability | Supported by Intel and backed-up by a fairly large community of developers and users. | - |
| Correctness | TBB masks all the complexity of massively multithreaded programming, helping write efficiently scalable, less error-prone codes. | - |
| Portability | The current release has been successfully ported and tested on x86, Power, SPARC architectures, several operating systems and compilers. Performance portability should be high due to the shared-memory parallelism paradigm common to all platforms. | Since TBB is designed using advanced C++ meta-programming features and interfacing itself with pthreads, it could be a problem to reach a successful build on some untested architectures. |
| Availability | The whole library is open-source (GPL) and publicly available for download. | - |

**Table 31 TBB pros and cons**

### 3.1.14 *Hierarchically Tiled Array (HTA)*

**Description**

Hierarchically Tiled Array (HTA) [18] is a C++-class designed for object oriented programming to exploit data locality and parallelism using “tiled arrays”. It is developed by David Padua and his team at the University of Illinois at Urbana-Champaign. HTA uses MPI for distributed memory and Intel Threading Building Blocks (TBB) for shared memory architectures. As the name implies the idea is to subdivide arrays into tiles in which each tile could be subdivided further. Exploiting data locality is achieved via hierarchical subdivisions as today’s multicore processors and distributed memory architectures have complex and hierarchical cache/memory structures. This allows expressing parallelism since the operations on each tile are independent in many algorithms. In other words, the tiles of HTA data types map naturally to the well-known block structure of many numerical algorithms. The high-level of abstraction ensures that HTA programs are portable across multiple platforms. The resulting code is usually compact and easy to understand. Hybrid programming, using MPI and Threads simultaneously, is not supported. The programmer is encouraged to use optimized sequential or threaded kernels available from libraries like BLAS, LAPACK, etc. No specialized debugger is available for HTA. However, since the programming language is C++, well known debuggers and performance analyzers such as GNU debugger (GDB), Intel debugger (IDB), or Intel Vtune performance analyzer can be used effectively.

**Code example**

Since HTA is based on tiled arrays, code has to be blocked. The following is a simple recursive blocked matrix-matrix multiplication example.

```
typedef HTA<double,2> HTA2;
typedef Tuple<2> T;
HTA2 A = HTA2::alloc(1,(T(n,n), T(n,n)), NULL, TILE);

void matmul(HTA2 A, HTA2 B, HTA2 C) {

  int M = A.shape()[0].size();
  int N = B.shape()[0].size();
  int Q = B.shape()[1].size();

  if(A.level()==0) {
    for (int i = 0; i< M; i++)
      for (int k = 0; k < N; k++)
        for (int j = 0; j< Q; j++)
          C[ T(i,j) ] += A[ T(i,k) ] * B[ T(k,j) ];
  }else {
    for (int i = 0; i< M; i++)
      for (int k = 0; k < N; k++)
        for (int j = 0; j< Q; j++)
          matmul( A( T(i,k) ), B( T(k,j) ), C( T(i,j) ) );
  }
}
```

Inner leaf

Actual computation

Recursion

**Experience & Results**

In this section, we report benchmark results for Euroben kernel mod2am/MxM in HTA, developed and measured at UYBHM.

We implemented mod2am/MxM using HTA and Intel TBB. For comparison we used the HTA built-in matrix-matrix multiplication routine. The benchmarks were performed on an Intel Core i7-740QM processor. Figure 19 shows the wall clock times of matrix multiplications using up to 8 cores.

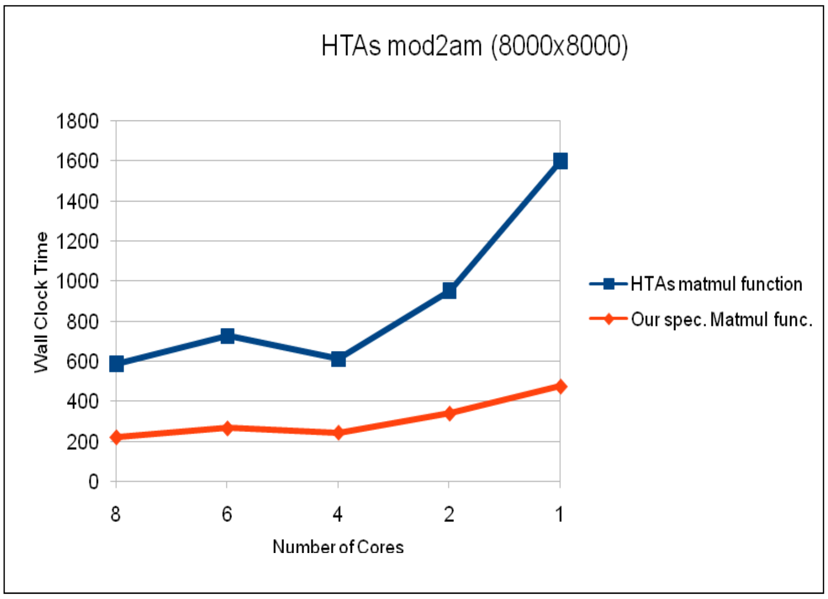


**Figure 19 Wall clock time (in seconds) as the number of cores increase for our implementation and built-in matrix-matrix multiplication routine. Tile size is the same for both implementation and fixed.**

We have fixed the number of cores at four and varied the tile (block) size.Figure 20 shows the effect of the tile size on performance for our matrix-matrix multiplication implementation and the built-in function.

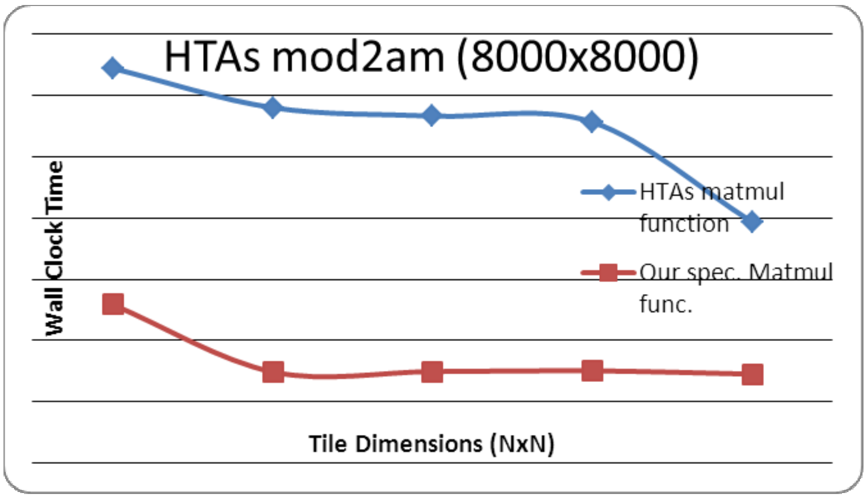


**Figure 20 Wall clock time (in seconds) as we increase tile size for our implementation and build-in matrix-matrix multiplication routine**

**Pros & Cons**

| | **Pros** | **Cons** |
|---|---|---|
| Scalability | Scalable for multicore processors. | - |
| Performance | - | Built-in functions might be optimized for a specific architecture. |
| Productivity | You can use its functions easily in your own implementation. To change block (tile) size, you do not need to re-write your code. | User documentation needs to be improved. |
| Sustainability | Uses new technologies like Intel TBB | - |
| Correctness | - | - |
| Portability | It has both distributed and shared-memory versions. | - |
| Availability | Open source | - |

**Table 32 HTA pros and cons**

## 3.2 Summary of new Programming Languages and Paradigms

### 3.2.1 Brief Overview and Classification

We have evaluated the parallel programming languages and environments shown in Table 33. These languages target traditional multicore CPUs, GPU accelerators, or both. In addition, we distinguish the use of languages and mixed programming environments for single nodes, i.e. cache-coherent fat nodes, versus multiple nodes with distributed memory architecture, and accelerated nodes with one or multiple GPUs per node (*single-node accelerator*) and multiple GPUs distributed across multiple nodes (*multiple-node accelerator*).

| **CPU languages** | | **GPU languages** | | **Languages targeting CPU and GPUs** |
|---|---|---|---|---|
| Single node | Multiple nodes | Single-node Accelerator (one or multiple GPUs) | Multiple-node Accelerator (multiple GPUs) | - |
| OpenCL<br>UPC<br>CAF<br>Chapel<br>Cilk<br>StarSs<br>TBB<br>ArBB<br>HTA (TBB) | UPC(+MPI)<br>CAF(+MPI)<br>ArBB (in the lab)<br>StarSs<br>HTA (MPI)<br>Chapel | CUDA<br>OpenCL<br>CAPS hmpp<br>PGI Accelerator Compiler<br>StarSs | CUDA+MPI<br>OpenCL+MPI<br>CAPS hmpp(+MPI) | OpenCL<br>CAPS hmpp<br>StarSs |

**Table 33 Parallel programming languages and environments.**

In Table 34 we classify the programming languages by the method used to express parallelism from the programmer's perspective, In addition, we note that some languages express parallelism explicitly, including CUDA, OpenCL, and Cilk. Other languages, including OpenMP, CAPS hmpp, and the PGI accelerator compiler, facilitate compiler-assisted parallelization of existing sequential programs by expressing parallelism via pragma-style annotations. The other languages rely on a compiler to translate a mix of declarations of parallel data structures and parallel control constructs into low-level communication and synchronization primitives.

| **Type of Parallelism** | **Method** | **Languages** | **Expressing parallelism** |
|---|---|---|---|
| Data-parallel | SIMD data-parallel programming | CUDA, OpenCL | Specific constructs enable the explicit handling of hierarchical computing architectures from fine-grained units of parallelism (threads) to more complex structures. |
| | Annotation with directives | OpenMP, CAPS hmpp, PGI Accelerator compiler, StarSs | Add directives to serial code that indicate regions which can be executed in parallel (done by compiler auto-parallelization). StarSs requires code that is blocked to allow an efficient automatic parallelization. |
| | PGAS | UPC, CAF, Chapel | Allow the programmer to treat the (physically) distributed memory as one (virtually) shared address space to simplify programming. |
| | Declaration of parallel data structures | Chapel, ArBB, HTA | Make use of the underlying data structures (e.g. dense matrices) to automatically decompose the data domain by means of an auto- |

| Type of Parallelism | Method | Languages | Expressing parallelism |
|---|---|---|---|
| | | | parallelizing compiler. |
| Divide-and-conquer parallelism | Spawn tree-recursive procedure calls | Cilk | Programmers expose parallelism in divide-and-conquer programs by spawning recursive function calls. The runtime system automates tasks like process placement and load balancing |
| Libraries | Encapsulation of parallel procedures | MPI, TBB, HTA | Encapsulate parallel procedures in a library for portability. Instead of relying on a separate compiler, the library can be optimized. |

**Table 34 Language classification by method for expression parallelism.**

### 3.2.2 Availability and Sustainability

The decision to use a new programming language bears significant risks concerning the expected return of programming investment. In an attempt to mitigate this risk, we list several facts about the availability and indicators for the sustainability of the programming languages in Table 35.

| | Size of user community | Restricted to HPC/scientific computing? | Size of development team | Open standard? | Free compiler? | Associated with a big company? |
|---|---|---|---|---|---|---|
| CUDA | Big | No | Big | No | Yes | Yes |
| OpenCL | Big | No | Big (scattered across hw plattforms) | Yes | Yes | Yes |
| CAPS hmpp | Small | No | Small | Ongoing discussions | No | Small company |
| PGI Accelerator Compiler | Small | No | Small | Ongoing discussions | No | Small company |
| UPC | Medium | Yes | Small | Yes | Partly, small fees necessary for some compilers | Academic project |
| CAF | Medium | Yes | Medium | Yes, part of Fortran | Partly, small fees necessary | Yes, picked up by several compiler |

| | Size of user community | Restricted to HPC/scientific computing? | Size of development team | Open standard? | Free compiler? | Associated with a big company? |
|---|---|---|---|---|---|---|
| | | | | 2008 | for some compilers | vendors |
| Chapel | Small | No | Small | Yes | Yes | Maintained by Cray as research project |
| StarSs | Small | No | Small | Not yet, might become a possible OpenMP extension | Yes | Academic project |
| Cilk | Medium | No | Small | n/a | Partly, small fees necessary for some compilers | Yes, CilkPlus is part of the Intel Compiler Suite |
| ArBB | Very small, (very new) | No | Medium | n/a | Probably not. | Yes, ArBB is maintained by Intel |
| HTA | Small | No | Medium | Standard C++ | yes | Academic project |
| TBB | Medium | No | Medium | Yes, library available under GPL | use custom C++ compiler | Yes, TBB is maintained by Intel |

**Table 35 Availability and indicators for sustainability of programming languages**

We note that most of the recently developed languages analyzed in this study focus and build on C or C++. Only the PGI Accelerator compiler, CAPS HMPP, StarSs and CAF target Fortran code.

# 4 System Software

Future-generation supercomputers will be assembled from millions of components, including compute nodes, network switches, and storage nodes. We have launched a survey about existing solutions for managing today's supercomputers, up to the multi-petascale, within the PRACE community. This chapter presents the results, analyzes the results, and draws conclusions by topic: operating systems, system management, data management, MPI and communication libraries, and ressource management.

## 4.1 Operating System

Operating systems provide essential services to manage the hardware resources of a computer. At first glance, operating systems for large-scale machines have to cope with different requirements than desktop machines. Our survey reveals, however, that Linux is the most popular operating system for today's supercomputers in Europe.

### 4.1.1 *Survey*

Our survey about operating systems lists for each contributing PRACE partner the machine by vendor and number of nodes, and the operating system. We classify the operating system (OS) as *lightweight* or not. A lightweight OS provides only services needed for high-performance computing, including process management and low-level communication capabilities. In contrast, a heavyweight OS offers commodity services not used by HPC applications. Furthermore, we classify the OS as configured *diskless* or not. A diskless OS relies on network services to provide remote storage rather than using the local harddisk of a node. Diskless OS configurations do not require the presence of a local harddisk in each node.

| **Site** | **Vendor** | **# Nodes** | **OS Type** | **Lightweight** | **Diskless** |
|---|---|---|---|---|---|
| CEA | BULL | 4000 | Linux<br>BULL A.E-1.0 (based on RedHat Enterprise Linux 6.0) | No | No |
| CSCS | CRAY | 1844 | Linux<br>CLE 2.2 (Cray Linux Environment) | Yes | Yes |
| EPCC | CRAY | 1856 | Linux<br>CLE | Yes | Yes |
| KTH | CRAY | 1516 | Linux<br>CLE | Yes | Yes |
| UYBHM | HP | 192 | Linux<br>Modified CentOS 5.4 | No | No |
| BSC | IBM | 2560 | Linux<br>SLES 10 SP2 (SUSE Linux Enterprise Server) | No | No |
| CINEC | IBM | 168 | Proprietary UNIX | No | No |

| Site | Vendor | # Nodes | OS Type | Lightweight | Diskless |
|---|---|---|---|---|---|
| A | | | AIX-6.1 | | |
| FZJ | IBM | 73728 | Proprietary (POSIX compliant)<br>CNK | Yes | Yes |
| STFC | IBM | 1024 | Proprietary (POSIX compliant)<br>CNK | Yes | Yes |
| HLRS | NEC | 711 | Linux<br>Scientific Linux 5.3 | No | Yes |
| IPB | PARADOX | 84 | Linux<br>Scientific Linux 5.5 | No | No |
| ICHEC | SGI | 320 | Linux<br>SLES 11 (SP1) | No | Yes |
| JKU | SGI | 1<br>(SSI) | Linux<br>SLES 10 | No | No |
| LRZ | SGI | 19<br>(SSI) | Linux<br>SLES 10.3 | No | No |

**Table 36 Operating systems used at PRACE sites**

### 4.1.2 *Analysis*

Our survey reveals that all operating systems are UNIX systems, with the exception of IBM's proprietary lightweight kernel CNK for Blue Gene. The majority of the UNIX systems are Linux variants. We note that all OS's are POSIX compliant, and observe a converging historical trend towards Linux.

Most of the Linux installations are heavyweight public-domain distributions. Some vendors offer customized Linux variants, such as IBM. The existence of these proprietary Linux variants demonstrates the need for an optimized OS tailored to large-scale supercomputers. Large-scale systems expose unique technical challenges to OS design such as desynchronizing noise that reduces the efficiency of tightly synchronized applications [58].

About half of the PRACE partners operate diskless OS configurations, either because the machine has no harddisk on the compute nodes or other reasons including reliability and power savings. We observe that most machines with more than 1000 nodes use a diskless OS configuration. Diskless configurations require a remote storage system with sufficient bandwidth to be competitive with local harddisks.

### 4.1.3 *Conclusions*

Linux has become the dominant operating system used by PRACE partners. The convergence towards this commonly used, open-source OS suggests focussing our efforts for future many-peta- to exascale machines on Linux. Besides evaluating Linux's readiness for exascale, we suggest to recognize the fact that Linux is open-source software that enables us to contribute exascale specific optimizations. We see an opportunity for developing an optimized exascale Linux, which could serve all PRACE sites in a uniform way.

## 4.2 System Management

The operating system relies on a number of external services to manage a computer, including boot services, name services, account management, and configuration. This section surveys the services in use at the PRACE sites.

### 4.2.1 *Survey*

Our survey of system management software consists of two parts. Part I lists for each contributing PRACE partner basic system management services: boot protocol, user adminstration, host administation, package and system update services, and configuration management. These services are used to manage not only the compute nodes of a machine but other components as well, including storage servers, login nodes, disk arrays, and network switches.

Boot protocols use a configuration server to assign network addresses and other information necessary for the boot process of each node in a network. User administration covers the distribution of information about user accounts, e.g. for access control as a first level of security. Host administration provides name services for network resources. Tools for managing software packages and updating system software automate the distribution of security patches, bug fixes, and release upgrades. Configuration management services automate consistency and policy checks of installed packages and their configuration across a network.

| **Site** | **Vendor** | **# Nodes** | **Boot protocol** | **User admin** | **Host admin** | **Package/System update** | **Configuration management** |
|---|---|---|---|---|---|---|---|
| CEA | BULL | 4000 | DHCP/PXE on Ethernet | LDAP<br>Flat files | DNS<br>Flat files<br>NSCD | YUM over HTTP | Puppet<br>Git |
| CSCS | CRAY | 1844 | Cray Proprietary via HSN | LDAP<br>NSCD | DNS<br>Flat files<br>NSCD | Cray XtopView | RCS |
| EPCC | CRAY | 1856 | Cray Proprietary | LDAP | DNS<br>NSCD | Cray Proprietary | Cray<br>Proprietary |
| KTH | CRAY | 1516 | - | Flat files | DNS<br>Flat files | - | - |
| UYB HM | HP | 192 | DHCP/PXE on Ethernet | LDAP<br>NSCD | DNS<br>Flat files<br>NSCD | YUM over HTTP | In-house |

| **Site** | **Vendor** | **# Nodes** | **Boot protocol** | **User admin** | **Host admin** | **Package/System update** | **Configuration management** |
|---|---|---|---|---|---|---|---|
| BSC | IBM | 2560 | BOOTP | Flat files | DNS<br>Flat files | - | - |
| CINECA | IBM | 168 | IBM proprietary on Ethernet | LDAP | DNS<br>Flat files | NIM | CSM |
| FZJ | IBM | 73728 | - | LDAP | DNS | - | Cfengine |
| STFC | IBM | 1024 | - | LDAP | - | - | - |
| HLRS | NEC | 711 | DHCP/PXE on Ethernet | LDAP<br>Flat files | DNS<br>Flat files | - | Cfengine |
| IPB | PARADOX | 84 | None | Flat files | DNS | YUM over HTTP | Kickstart |
| ICHEC | SGI | 320 | DHCP/PXE on Ethernet | LDAP<br>Flat files<br>NSCD | DNS<br>Flat files | YUM over HTTP | SGI Tempo |
| LRZ | SGI | 19 (SSI) | EFI | Kerberos5<br>Flat files | DNS<br>Flat files<br>NSCD | In-house | Cfengine<br>SVN |

**Table 37 System management software used at PRACE sites, Part I**

Part II of our survey of system management software covers tools for remote access and monitoring. Remote consoles offer administrative access to individual nodes of a machine. System logging and monitoring tools gather and display events and state of network nodes. Remote command execution via a remote shell facilitates non-privileged access to individual nodes of a machine. Remote power management tools provide control over the power consumption of individual nodes.

| **Site** | **Vendor** | **# Nodes** | **Remote console** | **Sys Log and event monitoring** | **Remote command execution** | **Remote power management** |
|---|---|---|---|---|---|---|
| CEA | BULL | 4000 | Conman<br>(BULL flavor) | Syslog-ng<br>snmpd<br>SEC<br>Nagios | ClusterShell | Nodectrl (BULL software stack based on IPMI) |
| CSCS | CRAY | 1844 | Cray | Syslog | Pdsh | Cray |

| Site | Vendor | # Nodes | Remote console | Sys Log and event monitoring | Remote command execution | Remote power management |
|---|---|---|---|---|---|---|
| | | | proprietary (Xtconsole) | Nagios Ganglia | (service nodes only) | proprietary |
| EPCC | CRAY | 1856 | Cray proprietary | Syslog | Pdsh | Cray proprietary |
| UYBHM | HP | 192 | HP ILO2 | Rsyslog | Pdsh | HP ILO2 |
| BSC | IBM | 2560 | - | Syslog | Pdsh | - |
| CINECA | IBM | 168 | IBM rconsole | Syslog IBM errpt | IBM dsh | IBM rpower |
| FZJ | IBM | 73728 | - | Syslog | Pdsh | - |
| STFC | IBM | 1024 | - | - | - | - |
| HLRS | NEC | 711 | - | Syslog | - | Ipmitools |
| IPB | PARADOX | 84 | Via IPMI | Syslog | Inhouse | Impitools |
| ICHEC | SGI | 320 | Conserver (SGI Temp software stack) | Syslog-ng | Pdsh | Ipmitools |
| LRZ | SGI | 19 (SSI) | SGI proprietary | Nagios Logwatch | SSH SGI proprietary array services | SGI proprietary |

**Table 38 System management software used at PRACE sites, Part II**

### 4.2.2 *Analysis*

We encounter a variety of boot protocols across the PRACE sites. The combination of the DHCP auto-configuration protocol and the preboot execution environment PXE dominates on small to medium sized systems with an Ethernet. Larger systems feature proprietary protocols. We note that DHCP/PXE is widely used for commodity system management other than supercomputers. For supercomputers with highspeed interconnect networks, no portable solution appears to exist.

The lightweight directory access protocol LDAP is the dominant tool for managing user accounts. As secondary solution, we find flat files, stored in protected areas. The simplicity of flat files yields superior reliability and scalability, whereas LDAP adds convenience of use. Therefore, we find hybrid solutions combining LDAP and flat files. Some sites also include a name service cache (NCSD) to improve scalability.

The domain name system DNS is the dominant tool for managing host names. Besides relying on DNS, the solutions for host management resemble those for user account management.

The software package manager YUM is the dominant tool for installing and updating software packages on Linux systems.

There is no commonly used tool for configuration management. However, most sites use a tool similar to the popular cfengine.

Tools for remote access and monitoring vary from site to site. Only the syslog tool appears to be in common use. We note that open-source solutions exist for these tasks, including syslog-ng for logging, IPMI for side-band node management, openssh for remote command execution, conman for remote console management, etc. Some vendors use these solutions, others provide proprietary tools. We are aware of scalability problems of monitoring tools, more succinctly their storage and compute requirements both of which are proportional to the number of nodes of a machine.

### 4.2.3 *Conclusions*

We encounter a large number of system management tools across different sites. While some tools are established de-facto standards, such as LDAP, DNS, and YUM, other services use open-source software, are vendor proprietary or site-specific solutions. We note that a convergence towards a standardized set of portable tools is desirable and feasible, because open-source tools exist already, although not necessarily with the scalability needed for exascale computing. We recommend to pursue the standardization of system management tools, and where necessary by developing and contributing scalable solutions to open-source projects.

## 4.3 Data Management

Supercomputers impose a challenging set of requirements on data management, including high-performance I/O and handling huge data sets. Our survey reveals that today's data sets are managed with a mix of tailored file systems.

### 4.3.1 *Survey*

Our survey of data management solutions distinguishes four common usage classes. (1) Users need easy yet protected access to personal data, ideally not restricted to the domain of the supercomputer, but via secure access to remote workstations and other network domains. (2) Scientific applications operate on large data sets, where I/O operations are performance critical. (3) Some data sets are stored for long periods of time, yet are not used very often. (4) System services and tools require temporary workspaces independent of user and application data.

| **Site** | **Vendor** | **Nodes #** | **Diskless** | **User data** | **Large data sets** | **Long term storage** | **System service workspace** |
|---|---|---|---|---|---|---|---|
| CEA | BULL | 4000 | No | NFS | Lustre | Lustre and HPSS | NFS |
| CSCS | CRAY | 1844 | Yes | NFS/GPFS | Lustre/GPFS | GPFS | NFS/GPFS |
| EPCC | CRAY | 1856 | Yes | NFS | External Lustre | Netbackup (DDN backend) | NFS |
| KTH | CRAY | 1516 | Yes | AFS Lustre | Lustre | - | Lustre |

| Site | Vendor | Nodes # | Diskless | User data | Large data sets | Long term storage | System service workspace |
|---|---|---|---|---|---|---|---|
| UYBHM | HP | 192 | No | Lustre | Lustre | Lustre | Lustre |
| BSC | IBM | 2560 | No | GPFS | GPFS | HSM | GPFS |
| CINECA | IBM | 168 | No | GPFS | GPFS | GPFS | GPFS |
| FZJ | IBM | 73728 | Yes | GPFS | GPFS | HSM | NFS<br>GPFS |
| STFC | IBM | 1024 | Yes | - | GPFS | - | GPFS |
| HLRS | NEC | 711 | Yes | NFS | Lustre | HPSS | NFS |
| IPB | PARADOX | 84 | No | NFS | NFS | glite_DPM | NFS |
| ICHEC | SGI | 320 | Yes | Panasas | Panasas | None | Panasas |
| LRZ | SGI | 19 (SSI) | No | NFS | CXFS | TSM | NFS |

**Table 39 Data management systems used at PRACE sites**

### 4.3.2 *Analysis*

We find that the PRACE systems employ different solutions for the four usage classes. Common to all sites is the use of tailored file systems, featuring a mix of network file systems, high-performance file systems, and long-term file systems.

The network file system NFS is the dominant commodity file system, widely used everywhere because of its reliability and portability. The PRACE sites employ NFS for user data and to manage the workspace of system services and tools. These usage classes are less performance critical than servicing large data sets.

Two high-performance file systems are widely used. GPFS is an IBM product and Lustre is an open-source file system supported by several vendors. The predominant use of high-peformance file systems is the management of large data sets and workspaces of system services. Both GPFS and Lustre are optimized for high throughput. They scale well when serving data but have known scalability limitations managing metadata [39]. Another known scalability problem affects the recovery time after a failure of individual system components, which is proportional to the number of nodes of the machine [39].

Long-term file systems include the proporietary file system HPSS, which is optimized to scale to large capacities. We note that HPSS is a joint development of several DOE laboratories and IBM to develop a viable long-term storage solution. Some long-term file systems are integrated into a hierarchical storage management (HSM) system, where data migrate between levels of a storage hierachy analogous to cache based memory hierarchies. IBM offers such a storage solution for GPFS, marketed under the name HSM, which is used in several IBM installations at PRACE sites. Another HSM solution is under development for Lustre in collaboration with CEA.

### 4.3.3 *Conclusions*

We observe that file systems are the dominant solution for data management, tailored to serve different usage requirements. High-performance file systems appear to meet contemporary needs, while gradual improvements to the commodity file system NFS are closing the performance gap. We are aware of several scalability problems that require our attention for next-generation exascale systems. We encourage the development of a portable open-source file system suited to serve as a vendor independent standard.

## 4.4 MPI and Communication Libraries

The message passing interface (MPI) is a key element of the software infrastructure for supercomputing. Our survey reveals that a number of well-known scalability problems plague different MPI implementations.

### 4.4.1 *Survey*

The section surveys the use of MPI implementations and underlying communication libraries. The contributing PRACE sites offer information about the interconnect network and the supported MPI implementations.

| **Site** | **Vendor** | **Nodes Number** | **Interconnect** | **MPI implementation** |
|---|---|---|---|---|
| CEA | BULL | 4000 | IB QDR | OpenMPI<br>BullxMPI |
| CSCS | CRAY | 1844 | CRAY SeaStar | Cray MPI |
| EPCC | CRAY | 1856 | CRAY Gemini | MPICH |
| KTH | CRAY | 1516 | CRAY Gemini | CRAY Mpich2 |
| UYBHM | HP | 192 | IB | MPICH<br>OpenMPI<br>MVAPICH<br>IntelMPI<br>Platform MPI |
| BSC | IBM | 2560 | Myrinet | OpenMPI<br>MPICH |
| CINECA | IBM | 168 | IB | IBM MPI |
| FZJ | IBM | 73728 | Proprietary | MPICH |
| STFC | IBM | 1024 | Proprietary | MPICH |
| HLRS | NEC | 711 | IB | OpenMPI<br>MVAPICH |

| Site | Vendor | Nodes Number | Interconnect | MPI implementation |
|---|---|---|---|---|
| | | | | iMPI<br>PACX-MPI |
| IPB | PARADOX | 84 | GbE | MPICH<br>Mpich2<br>OpenMPI |
| ICHEC | SGI | 320 | IB DDR | SGI-MPT<br>OpenMPI<br>MVAPICH<br>IntelMPI |
| JKU | SGI | 1<br>(SSI) | NUMALink 4 | OpenMPI<br>SGI-MPT |
| LRZ | SGI | 19 (SSI) | NUMALink 4 | SGI-MPT<br>IntelMPI |

**Table 40 MPI and communication libraries used at PRACE sites**

### 4.4.2 *Analysis*

As the standard among message passing interfaces, and the workhorse of contemporary high-performance programming, MPI is supported by all PRACE sites. Proprietary low-level communication libraries are typically augmented with an MPI compliant library. Systems based on commodity interconnects, such as IB, tend to support open-source MPI implementations like OpenMPI.

The performance and scalability of MPI different operations, in particular collective operations like broadcasts, reductions, and barriers, varies substantially across MPI implementations and interconnect networks. We briefly describe two of the most severe scalability problems that require solutions for exascale machines.

One scalability problem is caused by the memory requirements for send and receive buffers. MPI implementations like OpenMPI are known not to scale on IB because of their buffer management [39]. Network interface hardware offers buffering support for communication, such as queue pairs in Infiniband (IB), yet hardware resources are limited. Handling large numbers of connections per node requires virtualizing the hardware buffers, which introduces a memory overhead proportional to the number of connections when implemented naively. At the time of this writing, several solutions have been proposed but no dominant solution has emerged yet. For example, Mellanox IB cards share communication buffers between tasks within a node [54]. However, even with such sharing, the memory footprint for send and receive buffers consumes a significant amount of node memory.

Another scalability problem is caused by network topologies. Today's systems are networks of multicore nodes. Hence intranode and internode communication are facilitated by different hardware machanisms, that require an optimized software stack to obtain high performance. In particular, it is widely believed that portable implementations of collective operations do not offer the performance and scalability of topology-aware implementations [35].

### 4.4.3 *Conclusions*

MPI is the dominant communication layer for high-performance programming, and as such considered to be mission-critical software infrastructure. However, MPI implementations suffer performance and scalability problems, in particular collective operations like broadcasts, reductions, and barriers. Some of these problems are well-known, but no common solutions have emerged yet. We have an opportunity to develop and contribute solutions for exascale computing to open-source projects like OpenMPI.

## 4.5 Resource Management

The resource manager is responsible for the efficient use of a supercomputer.  Its tasks include allocation of exclusive and/or non-exclusive access to resources such as compute nodes for a specific duration of time, starting, executing, and monitoring jobs on a set of allocated nodes, and  arbitration of competing requests for compute resources by managing a queue of pending work.

The scheduler component of the resource manager decides which resources to allocate to a job according to a given policy.  The simplest policy is the first-come-first-served (FCFS) behavior implemented by a queue.  The FCFS policy is not adequate for contemporary supercomputer architectures with multicore nodes and accelerators, because:

1. *Users* expect fast response and short turnaround times. They may also expect to obtain a fair share of resources. Hence, the scheduler must implement advanced sharing policies that take into account not only the jobs but also the users, so that resource time is divided among the users. Many schedulers address this issue with so-called fairshare policies. Preemption is another policy that refers to suspension of a currently executing job so that the released resources can be allocated to higher priority jobs.
2. *Jobs* have different resource requirements. For example, parallel applications have different dependency and communication patterns, some jobs require checkpointing, others exploit  dynamic voltage and frequency scaling to minimize power consumption.
3. *Resources* of a machine may be heterogeneous. Accelerated architectures feature multicore processors plus one or more accelerators, such as a GPU. Different interconnect network topologies benefit from topology-aware schedulers.
4. *Power consumption* has become a primary concern. Schedulers can help managing thermal aspects of heat dissipation by exploiting the ability to power on/off resources. Saving energy necessitates the development of energy-aware scheduling policies.

To improve the utilization of resources most schedulers employ the backfilling technique, that permits jobs requesting a small number of resources to occupy empty ressource slots without modifying the order of the execution of previously submitted jobs.

### 4.5.1 *Survey*

To evaluate the state-of-the-art in resource management, we present (1) insights from a literature  survey and (2) a survey of the resource managers in use at the PRACE sites.

Our survey of contemporary scientific literature reveals the existence of several resource managers. Georgiou's PhD thesis [13] includes an excellent assessment, including those resource managers  widely used systems in supercomputer centers:

1. SLURM [57] is open-source resource management software, designed with simplicity, portability, and scalability in mind. It has a plug-in mechanism for developers to extend

its functionality. New versions of SLURM extend the simple FCFS scheduler with features such as backfilling, fairshare, preemption, multi-priority, advanced reservation, application licenses, external scheduler support, simple topology awareness, and a plug-in mechanism for generic resource scheduling. Rudimentary support for GPUs is provided via the generic resource scheduling mechanism. SLURM received a lot of attention recently, and some supercomputer centers use SLURM, sometimes in combination with other schedulers such as MOAB, Maui, and LSF.

2. PBSpro [46] is a commercial successor of the PBS resource manager originally developed at NASA. The original PBS code is available as unsupported open-source version, called OpenPBS. PBSPro has the usual scheduler features: FCFS, backfilling, fairshare, preemption, multi-priority, external scheduler support, advanced reservation support, and application licenses. Recent versions support topology-aware scheduling by means of so-called 'placement sets,' which are sets of cores and nodes for placement of MPI tasks specified by the site administrator. Support for GPU has been introduced in two modes: *(i)* the simple mode schedules one GPU job at a time on a given node exclusively, and *(ii)* an advanced distributed mode offers sharing of nodes by multiple jobs at the same time and grants access to individual GPUs by device number.
3. MOAB [62] is a commercial job scheduler that originated from the PBS system. It supports the usual features: FCFS, backfilling, fairshare, preemption, multi-priority, advanced reservation, and application licenses. MOAB is just a scheduler and, hence, needs to be integrated into a resource manager system.
4. TORQUE and MAUI [62]: TORQUE is the open-source version of the PBSPro resource manager and MAUI is the open-source version of the MOAB scheduler. MAUI supports FCFS, backfilling, fairshare, preemption, multi-priority, advanced reservation, and application licenses, and rudimentary topology awareness. Besides supporting fat (ccNUMA) nodes, TORQUE also has some built-in support for GPUs.
5. LSF [47] is a commercial scheduler that supports FCFS, backfilling, fairshare, preemption, multi-priority, advanced reservation, and application licenses. Newer features include live cluster reconfiguration, SLA-driven scheduling, delegation of administrative rights, topology-awareness, and support for GPUs. Also, LSF is capable of using thermal data to avoid hot spots by balancing workloads.
6. LoadLeveler [38] is a commercial product from IBM, initially based on the open-source CONDOR system [7]. It supports FCFS, backfilling, fairshare, preemption, multi-priority, advanced reservation, and application licenses. LoadLeveler features a specialized scheduling algorithm for Blue Gene.
7. OAR [43] is a recently developed open-source resource manager for high-performance computing. It is the default resource manager of Grid500, a large-scale experimental platform for computer scientists to run distributed computing experiments under real life conditions.

In addition to the above scheduling software packages, Condor [7] and Oracle Grid Engine [45] (formerly known as Sun Grid Engine) systems are also widely used, especially in grid environments.

Our survey of resource managers at the PRACE sites is shown in Table 41

1. Scheduler
2. Meta-Scheduler: combines multiple distributed schedulers into a single collective view, and coordinates scheduling of jobs by directing them to the appropriate scheduler for execution.

3. Scheduling Policies: QOS (quality of service), first-come-first-served (FCFS), fairshare, preemption, or backfilling.
4. Topology Awareness: refers to the optimized mapping of a job to the network topology such that the nodes of a job are in close proximity to each other.
5. Allocation Granularity: refers to the resource allocation units available on the machine. These can be cores, nodes, board, or memory.
6. Checkpointing / Restart: refers to the ability of the system or application to save checkpoints of the application for restart after failures.
7. Job Dependencies: refers to the structure of dependencies among subjobs of a larger job, including linear or pipeline dependencies and arbitrary directed acyclic graphs, i.e. workflows.
8. MPI Integration: refers to the launcher integration for MPI process management.

| **Site** | **Scheduler** | **Meta-Scheduler** | **Scheduling Policies Used** | **Topology Awareness** | **Allocation Granularity** | **Check-point/ Restart** | **Job depen-dencies** | **MPI integration** |
|---|---|---|---|---|---|---|---|---|
| BSC Spain | SLURM MOAB | | Fairshare Backfilling | Yes | Cores | No | Graphs | SLURM launcher |
| CEA France | SLURM | Inhouse | QOS+FCFS Fairshare Backfilling | Yes | Cores Memory | Yes App Level | Linear | SLURM/ OpenMPI Launcher |
| CINECA Italy | Load-leveler | No | Backfilling Preemption | Yes | Cores Memory | Yes Application Level | Workflow | Embedded launcher |
| CSCS Switzer-land | PBSpro (SLURM planned) | No | Backfilling | Yes | Nodes | Yes Application Level | Linear | Cray proprietary ALPS |
| EPCC UK | PBSpro | N | Backfilling | No | Nodes | No | Linear | Explicit job launcher |
| FZJ Germany | Load-leveler | Yes | Backfilling | Yes | Nodeboard (32 nodes) | Yes Application Level | Linear | None |
| HLRS Germany | Torque MOAB | No | Backfilling | Yes | Nodes Memory | Yes Application Level | None | Other |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ICHEC<br>Ireland | Torque<br>MOAB | No | Fairshare<br>Backfillling | No | Nodes | No | Linear<br>Workflow | OSC MpiExec |
| IPB<br>Serbia | Torque<br>MAUI | Yes<br>glite_WMS | Fairshare | No | Cores<br>Memory | Yes<br>Application<br>Level | Graphs (DAG) | MPI-START |
| JKU<br>Austria | Under full user control without resource management system | | | | | | | |
| KTH<br>Sweden | MOAB | No | FCFS<br>Fairshare<br>Backfilling | Yes | Nodes | Yes<br>Application<br>Level | Yes | Other |
| LRZ<br>Germany | PBSpro | Inhouse | First fit<br>Backfilling<br>Starvation<br>mechanism | Yes | Altix 4700 nodes<br>(physical parts of an SSI instance) | No | Linear | Embedded launcher |
| STFC<br>UK | Other | No | Backfilling | Yes | Nodes | Yes<br>Application<br>Level | Linear | Embedded launcher |
| UYBHM<br>Turkey | LSF | No | Fairshare<br>Preemption | No | Nodes<br>Cores<br>Memory | Yes<br>Application<br>Level | Workflow | Embedded launcher |

**Table 41 Scheduling software, policies and capabilities used at PRACE sites**

### 4.5.2 *Analysis*

Our surveys yield several observations and suggestions for improving the scalability of resource managers:

1. 5 out of 14 PRACE sites use open-source schedulers. We note that CEA has chosen SLURM as the resource management system for Tera100, which is currently Europe's most powerful system according to the Top500 list, and CSCS is planning to switch to SLURM. With the exception of one site, all of the remaining sites employ commercial resource managers.
2. GPU support: As supercomputer vendors are experimenting with GPU accelerators, schedulers are starting to support various node configurations, either as accelerators attached to CPU cores, or as separate resources to be shared dynamically among multiple nodes.
3. Topology awareness: is widely believed that topology-aware mapping of applications to resources in close topological vicinity improves performance. Large numbers of nodes and jobs result in a combinatorial explosion of the search space for possible mappings. Handling faulty nodes by changing the network topology dynamically exacerbates the

mapping problem. Traditional search algorithms of first-fit best-fit type may be insufficient for large-scale systems, and a more sophisticated approach based on combinatorial optimization should be pursued.

4. Scalability: multi-petascale systems already consist of more than 100,000 cores. SLURM, currently supports only up to 65,000 nodes and MOAB 40,000-50,000 nodes.
5. Energy awareness: today's resource managers have only initial, rudimentary support for energy awareness. We observe the need for developing scheduling algorithms that decide when to power off or put idle machines into sleep mode, and that provide support for DVFS.
6. Portability: a large variety of resource managers is in use at the PRACE site. There is a lack of portability among job scripts, access and control of resources.
7. Malleability: current schedulers assign resources statically, i.e. once they are assigned to a job, they remain assigned for the life time of a job. For long running applications, system utilization can be improved by adopting the resources dynamically during the life time of a job. We observe an opportunity for developing malleability techniques for schedulers, because we expect them to be particularily effective on exascale systems.
8. Workflows: specify the composition of larger jobs from smaller subjobs. Workflows are not widely used in scientific computing although they are well suited for scheduling with backfilling algorithms. We observe an opportunity to enhance supercomputer schedulers with a workflow scheduling capability [51].
9. The experience of supercomputer users with existing schedulers can be invaluable for the design of improved schedulers. To-date, little experience [1] has been reported in the literature on this topic. Several PRACE sites have responded to make available their scheduler logs for research purposes. We suggest establishing a repository for PRACE scheduler logs to support future research on supercomputing schedulers.

### 4.5.3 *Conclusions*

A large variety of open-source and commercial resource managers are in use at the PRACE sites. Common to almost all existing resource managers is a lack of support for GPU accelerators and energy awareness. Furthermore, we expect to experience suboptimal utilization with today's scheduling algorithms when handling the large number of jobs and heterogeneous resources in future systems. We observe an opportunity to develop advanced scheduling mechanisms and algorithms for improved system utilization of exascale systems.