SEVENTH FRAMEWORK PROGRAMME

Research Infrastructures

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

PRACE-2IP

PRACE Second Implementation Phase Project

Grant Agreement Number: RI-283493

D11.2
System Software and Application Development Environments

Final

Version:	1.0
Author(s):	Jerry Eriksson, HPC2N/SNIC
	Radoslaw Januszewski, PSNC,
	Olli-Pekka Lehto, CSC
	Carlo Cavazzoni, CINECA
	Torsten Wilde, LRZ
	Jeanette Wilde, LRZ
Date:	29.08.2014

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-283493**	
	Project Title: PRACE Second Implementation Phase Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: < **D11.2**>	
	Deliverable Nature: <Report>	
	Deliverable Level: PU	**Contractual Date of Delivery:** 31 / 08 / 2014
		Actual Date of Delivery: 31 / 08 / 2014
	EC Project Officer: Leonardo Flores Añover	

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

Document Control Sheet

Document	**Title: System Software and Application Development Environments**	
	ID: D11.2	
	Version: <1.0 >	**Status:** ***Final***
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D11.2-Update.docx	
Authorship	**Written by:**	Jerry Eriksson, HPC2N/SNIC
	Contributors:	Radoslaw Januszewski, PSNC, Olli-Pekka Lehto, CSC Carlo Cavazzoni, CINECA Torsten Wilde, LRZ Jeanette Wilde, LRZ
	Reviewed by:	NN, Organisation
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	16/06/2014	Draft	
0.2	25/072014	Draft	Comments and suggestions from Jonathan Follows
0.3	11/08/2014	Draft for internal review	Michael Krieger, JKU; Thomas Eickermann, FZJ
1.0	22/08/2014	Final	Incorporates changes based on PMO review

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure

Table of Contents

List of Figures

List of Acronyms and Abbreviations

AMD	Advanced Micro Devices
API	Application Programming Interface
CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CUDA	Compute Unified Device Architecture (NVIDIA)
DARPA	Defense Advanced Research Projects Agency
DDN	DataDirect Networks
DDR	Double Data Rate
DRAM	Dynamic Random Access memory
EC	European Community
ERE	Energy Reuse Effectiveness
FPGA	Field Programmable Gate Array
GB	Giga (= 2^{30} ~ 10^{9}) Bytes (= 8 bits), also GByte
GCC	GNU CC compiler
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPU	Graphic Processing Unit

HDD	Hard Disk Drive
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HSA	Heterogeneous System Architecture
HSAIL	HSA Intermediate Layer
IB	InfiniBand
IBA	IB Architecture
IBM	Formerly known as International Business Machines
IPMI	Intelligent Platform Management Interface
KPI	Key Performance Indicator
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MIPS	Originally Microprocessor without Interlocked Pipeline Stages; a RISC processor architecture developed by MIPS Technology
MPI	Message Passing Interface
MPP	Massively Parallel Processing (or Processor)
MPSS	Manycore Platform Software Stack
NFS	Network File System
NIC	Network Interface Controller
NUMA	Non-Uniform Memory Access or Architecture
OpenCL	Open Computing Language
OpenGL	Open Graphic Library
Open MP	Open Multi-Processing
OS	Operating System
PCIe	Peripheral Component Interconnect express, also PCI-Express
PCI-X	Peripheral Component Interconnect eXtended
PGI	Portland Group, Inc.
PPE	PowerPC Processor Element (in a Cell processor)
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PSNC	Poznan Supercomputing and Networking Centre (Poland)
PUE	Power Usage Effectiveness
QDR	Quad Data Rate
SDK	Software Development Kit
SSD	Solid State Disk or Drive
SLURM	Simple Linux Utility for Resource Management
STL	Standard Template Library
TBB	Threading Building Blocks
WUE	Water Usage Effectiveness

Executive Summary

This deliverable summarizes the work associated with task 11.2 "System Software and Programming Environments" in the PRACE-2IP Work Package 11 "Prototyping". The goal of this task is to evaluate the system software that is needed in order to have a working prototype.

This deliverable covers the experience of each of the prototypes in terms of: porting necessary system software and programming environments to the prototype platforms; the differing software stacks used by the prototypes; and evaluating parallel programming environments and system management tools for novel computing platforms.

1 Introduction

This deliverable documents the final results of Task 11.2 of the PRACE-2IP Work Package 11. Task 11.2 focused on prototype system software evaluation. This activity was very beneficial for PRACE and the involved PRACE partners since it has:

- Evaluated system software including monitoring tools;
- Looked at the differing software for accelerators including the software stacks and operating systems; and
- Evaluated application software including schedulers, system management tools, and parallel programming environments.

The three selected PRACE-2IP WP11 prototypes (as documented in D11.1.1 “Prototyping Project Plan”) have investigated the latest cooling and hardware technologies with a goal of obtaining an energy efficient system. This deliverable will go through each of the items listed above and provide experiences and results for each prototype.

The main audience of this deliverable are the PRACE data centre management. Its dissemination level is public.

A basic description of each of the prototypes is given in Chapter 2. Chapter 3 covers the different operating systems that were used. Chapter 4 contains a discussion of the software supporting the accelerators. Chapter 5 reviews differing schedulers. An evaluation of system management tools is provided in Chapter 6. Parallel programming environments are discussed in Chapter 7. And finally a summary is provided in Chapter 8.

2 Prototype Overview

The following sections provide a basic overview of each of the prototypes. D11.1.2 "Prototyping and Technical Evaluation Summary" has a more in depth discussion of each prototype, their hardware, final results, and lessons learned.

2.1 Scalable Hybrid at CSC

The overall goal of the Scalable Hybrid prototype is to combine a number of evolutionary technologies that are representative of a likely next generation multi-petaflop system. The size of the system is intended to be sufficiently large to provide the ability to perform scalability tests. The purpose of this prototype is to evaluate power efficiency, manageability, and novel programming environments (both in terms of performance and productivity) in a system that combines evolutionary innovation in packaging and cooling, interconnect and accelerator co-processing technology as well as OS and systems management solutions. The prototype is also used to develop parallel hybrid applications which are highly scalable due to the novel interconnect architecture. The prototype is designed to be large enough to enable accurate performance projections into the trans-petascale regime and provide a platform for collaboration with the EURORA consortium.

The system was deployed in several phases (dates of deployment in parentheses):

- T-Platforms air-cooled prototype
  - Phase 1: 10 air-cooled nodes with NVIDIA Fermi (10/2012)
  - Phase 2: Upgrade of 5 nodes with Intel Xeon Phi (03/2013)
- Bull liquid-cooled prototype
  - Phase 1: 45 Xeon Phi nodes and racks (11/2013)
  - Phase 2: 38 Kepler K40 (03/2014)

The original work plan was to build a liquid-cooled system with T-Platforms. However, this plan was not realized due to export restrictions imposed upon T-Platforms by the U.S. Bureau of Industry and Security (BIS) which caused stoppage of supply of key components to the company until the end of 2013.

Bull was selected to provide the final, liquid-cooled prototype as they could build a similar system within the timeframe of the project.

The initial test system is a T-Platforms system that consists of the following elements:

- 2 blade chassis, each with
  - 4 x 1600W PSU
  - Management processor and integrated management switch
  - 3 air-cooling modules with 120mm coaxial fans
- 10 T-Blade V-200F blades, each with
  - 2*Intel Xeon E5-2640 CPUs (Sandy Bridge)
  - 16 GB DDR3 Memory
  - 10 NVIDIA Tesla M2090 GPGPUs
    5 replaced by Intel Xeon Phi 5110P in Phase 2

The system contained a T-Platforms software stack based on xCAT provisioning and SLURM batch job queuing systems. The system is connected with FDR InfiniBand using a single switch provided by CSC. Each blade and chassis has comprehensive power and temperature monitoring facilities accessible via IPMI and a Web interface.

The prototype is located in the CSC facility in Espoo and will continue to be used for system-level experiments such as testing new versions of software stacks. This will enable the

environment on the large-scale prototype to be stabilized for application development and porting work.

2.2 EURORA at CINECA

The HPC prototype system installed at CINECA is a new concept of HPC cluster cooled by a "hot" water cooling system. The machine is able to work at a higher temperature than traditional clusters. This has an interesting outcome for cooling power consumption. The energy used for the air conditioning systems, or to cool down the water, is lower than the energy used for standard water-cooling of a cluster with the same computing power. Furthermore, the direct-current supply lowers the power needs of the system.

EURORA is a single rack cluster and implements Intel Xeon Sandy Bridge CPUs in 64 double-socket nodes for a total of 1024 cores. The internal network is made of one FPGA (Altera Stratix V) per node, an IB QDR interconnect, and a 3D torus interconnect. Thanks to the innovative cooling and system engineering, EURORA was ranked in a top position of the Green 500 chart[1] in June 2013, with a sustained performance of 3,150 MFlops/W. Intel Xeon Phi nodes (MIC nodes) are equipped with Intel E5-2658 Xeon Sandy Bridge with a clock of 2.10GHz; NVIDIA K20 nodes (GPU nodes) are equipped with Intel E5-2687W Xeon Sandy Bridge with a clock of 3.10GHz. Both GPU and MIC nodes have 2 accelerators per node. The operating system running on EURORA is Centos 6.3.

The cluster is equipped with

- Intel Xeon E5-2687W sandy bridge processors for a total of 1024 cores hosted in
- 64 computing nodes capable of delivering 3,150 megaflops per watt of sustained performance.

Each computing node is equipped with:

- 16GByte DDR3 1600MHz
- 2 NVIDIA K20
    - In half of the nodes, to be substituted (end of June 2013) with Xeon Phi cards
- 1 FPGA (Altera Stratix V)
- IB QDR interconnect
- 3D Torus interconnect
- 160GByte SSD

2.3 Hybrid CPU/GPU at PSNC

The goal of the CPU/GPU prototype was to investigate the changes AMD introduced in the new series of GPU units. The research focused on the possibility of implementing these solutions in HPC as an alternative to currently used solutions. In addition, during the implementation phase it was possible to use warm-water cooling for both CPU and GPU. In the test cluster, the interworking of immersion cooling with GPU cards was observed.

CPU/GPU is a single rack cluster built by Iceotope and funded by PRACE-1IP and PRACE-2IP. It hosts 40 nodes running dual Xeon Sandy Bridge 2620 clocked at 2.0GHz and 2.3GHz when using turbo mode on all cores. Each node is equipped with 32GB ram, dual 1Gbit eth links, and InfiniBand QDR. One node is connected to a dedicated 10TB SSD matrix exposed to the nodes using Lustre. 6 nodes are connected to slave modules that host 2 AMD S9000

[1] http://www.green500.org/

GPUs each. Ubuntu 12.04 server is the selected operating system. In addition, a machine with Richland APU AMD A4-5300 was made available.

The system consists of two 42U racks. One is hosting the computing system with the internal cooling loop. The second rack is used for the dedicated external cooling loop hydraulics and control. The servers are inserted into 6 chassis (3 on each side of the rack), 8 modules in each chassis. Each chassis may be powered by two redundant PSU modules. In the rack, only switches and PDU units are cooled by air.

The compute rack consists of:

- 2xGbit ethernet switches
- 2xQDR Mellanox InfiniBand switches
- 2x power distribution units that powers up 6 PSU units (one powering 8 modules)
- 34*modules with:
    - 2*Intel Xeon Xeon(R) CPU E5-2620
    - 32GB DDR3 1600Mhz memory
    - One InfiniBand FDR Mellanox chip, 2xGbit eth ports
    - One 60GB SSD HDD
- 6*modules with:
    - 2*Intel Xeon Xeon(R) CPU E5-2620
    - 64GB DDR3 1600Mhz memory
    - One InfiniBand FDR Mellanox chip, 2xGbit eth ports
    - One 60GB SSD HDD
    - Dual PCI-e x16 external uplink ports
- 6*modules with:
    - 2*AMD S9000 GPU
    - Dual PCI-e x16 external uplink ports
- 2 redundant heat exchangers
- 2 redundant pumps

3 Operating System

3.1 Linux

3.1.1 *Experiences from Scalable Hybrid*

The only options for having robust support for both Intel Xeon Phi, NVIDIA GPGPUs and Mellanox Infiniband were SLES and RHEL. As there was a familiarity with RHEL and it is used in the other clusters, CentOS, a free variant of RHEL was chosen. One issue was the selection of the version number. Due to limited support for the latest version of CentOS/RHEL (6.5 at the time) in the Xeon Phi software stack, CentOS version 6.3 was used. At the time of writing, support was also present for the latest version.

One lesson learned is that the accelerator drivers and software stacks tend to have a lag from when they have certified support for the latest OS and kernel versions. This causes problems if there is an urgent need to upgrade (for example due to a security vulnerability). In general, the accelerator vendors should work closer with the OS suppliers during the beta testing of a new OS version to ensure that their products are ready and certified when the new OS version is released. Furthermore, vendors should test their accelerator drivers against any major interim kernel security updates as soon as they are released. The accelerators should not cause administrators significant constraints in keeping their systems up-to-date.

3.1.2 *Experiences from EURORA*

For EURORA, CentOS was chosen as it was closer to the preferred distribution Red Hat. EURORA had no issues with the operating system aside from ensuring that the OS had the current version. A learned lesson is that fixes to the kernel (especially security related) should be implemented despite the fact that the machine is a prototype.

3.1.3 *Experiences from CPU/GPU*

The selection of the Linux distribution for CPU/GPU was a simple one: it had to be on the supported list for the FirePro cards which left either Ubuntu or Red Hat releases.

The Ubuntu server was chosen because the preferred distribution is Debian and Ubuntu is debian-based but more up-to date and has all proprietary drivers included in the distribution repositories. All of the installation and set-up was performed by the PSNC team.

The overall experience was very good, the system is stable and there were no compatibility issues. The update procedures were smooth and non-disruptive even for major release changes (12.04 Long Time support to 14.04 and then to 14.04 LTS). All required packages, including monitoring and resource management, were included in the repositories so there was no need for manual installation of 3rd party packages.

The only package that was installed from outside the repositories were FirePro drivers. There are official AMD drivers installed in the repositories but the refresh cycle of the drivers was not sufficient as only stable versions of the drivers are allowed to be installed in the official repositories. Some of the features that were to be tested were introduced in the beta-versions of the drivers so it was decided to manually select and install these packages.

Because it was decided not to use the SSD drives installed in the servers to boot the machines, a NFS compatible release of Ubuntu was prepared. This was the only challenging task in the entire installation procedure. Because Ubuntu is a Debian-based distribution, all manuals and HOWTOs available on the Internet were almost 100% compatible and it was possible to use them directly.

4 Software Stack

4.1 Challenges for Accelerators

4.1.1 *Experiences from Scalable Hybrid*

Both the Xeon Phi and the Tesla cards have drivers and associated software stacks which are certified to operate with a specified set of certified operating systems (OS). The certification process usually is such that there is a slight gap (2-6 months) when a new OS version is released to when the official support for it arrives. Typically, this happens with the next major or minor release of the software stack. In case of urgent kernel updates, for example due to security exploits, it is possible to recompile the driver for a newer kernel but there is a slight risk that unexpected bugs may arise.

At the time of writing, the software stacks support the following OS versions:

- **Kepler/CUDA 6:** RHEL 5.x/6.x, Fedora 19, SLES11, Ubuntu 12/13, L4T, OSX 10.8/10.9, Windows XP/Vista/7/8.1
- **Xeon Phi/MPSS 3.2**: RHEL 6.x, SLES 11, Windows 7SP1/8 Enterprise, Windows Server 2012/2013

There are various unofficial guides available online to use the cards on unsupported operating systems but this often requires manual work and may introduce bugs. There are also other dependencies which can cause issues, especially if integrating accelerators to an existing cluster with an established software stack.

OpenFabrics (OFED) and MPI

Both the Tesla and Xeon Phi have developed facilities to communicate efficiently over the InfiniBand interface with the goal of providing Remote Direct Memory access between cards on different hosts. For Xeon Phi there is the SCIF virtual InfiniBand interface and for Tesla there is GPUDirect RDMA.

The OpenFabrics (OFED) software stack provides a common layer between the upper-level interfaces such as MPI and the low-level drivers for fabrics using Remote Direct Memory Access (RDMA). The OFED stack is used in practically all the production clusters today.

During its evolution, the OFED stack has diverged into the standard OFA (Open Fabrics Alliance) OFED and vendor-enhanced stacks from Mellanox and Intel (formerly QLogic). Furthermore, there are two major versions used widely today, 1.5 and 2.0, which are not compatible. The former is typically the standard version shipped with many OS suppliers.

Typically the accelerators only support a specific subset of the OFED versions. In addition, the MPI libraries (typically OpenMPI, MVAPICH2, and Intel MPI) only support features such as high-performance RDMA mode with even a smaller subset of OFED and MPI libraries.

4.1.2 *Experiences from EURORA*

With the Intel MIC coprocessors being based on x86-compatible multiprocessor architecture, no critical issues concerning their configuration and integration with the operating system arose in the set-up of the production environment of EURORA. The standard Intel software suite provides, in fact, support to configure and use the MIC coprocessors through the MPSS. It was, however, crucial at the start to upgrade the Intel software stack to the 2.1 release in order to improve the stability of the nodes with the MICs.

The development environment for MICs is provided on EURORA by the standard Intel suite Cluster Studio XE 2013. Apart from the MPSS installation, no further package is required for the code development on the MIC cards.

The only, additional, post-installation operation required by the MICs is the building and the installation of the SEP drivers on the MICs for the VTune Amplifier for performance analysis. Though not equipped with MICs, the software stack has been made available on EURORA front-end node as well, in order to allow the compilation of MIC codes without the need to log into a compute node hosting the coprocessors.

The Intel suite allows the compilation of codes to be run with the MIC offload attributes and the cross-compilation of MIC native codes. In both cases, the setup of the proper MIC environment requires users to source the compilervars.sh script provided by the Intel suite. The sourcing of the script is also required before executing an offload program on a node hosting the MIC cards. For the execution of MIC native codes, once logged on the MIC it is instead necessary for the users to manually set the LD_LIBRARY_PATH to include the directory with the Intel MIC libraries (and of any other needed library such as, for instance, the Intel MKL). MPI applications require in addition the setting of the I_MPI_MIC environment variable and the standard use of IntelMPI libraries through the compiler MPI wrappers.

4.1.3 *Experiences from CPU/GPU*

The AMD FirePro series is a product line targeted at the professional community working with 3D graphics or CAD software. The S9000 is the first generation of AMD GPU units that were designed to offload the computational part of the graphics generation from the “normal GPU”. Therefore, it was not meant to work as a stand-alone unit. The software stack provided by AMD reflected this idea. The drivers are part of the Xorg software stack. Even though the cards did not have video output a full X-windows stack is needed to be operational which in turn caused installation and stability issues (for further information please see Chapter 4 of D11.1.3 “Final Prototyping and Technical Evaluation Summary”). The S9000 cards support only OpenCL. Unfortunately, the majority of scientific codes written with GPU acceleration are written using CUDA and are, therefore, NVIDIA GPU specific. Production level OpenCL codes ran well overall.

Problems arose during application development. One reason being that the card does not automatically clean-up left over memory allocations. If an application didn’t release the memory and/or terminated unexpectedly, the card would freeze and force a reboot. At times it took multiple runs of the application, in which memory was not released, before the card would freeze. From the way that the card driver was written there was no way of releasing the memory outside of the application. Handling of such situations was better in each subsequent version of the driver but still the card often needed to be rebooted.

4.1.4 *Porting Example - CRANK*

As an example for an application, we ported `CRANK` to Scalable Hybrid `CRANK` is a pipeline for an automatic solution of protein structures from X-ray crystal diffraction data. To solve the X-ray crystallography phase problem, `CRANK` uses "experimental" phase information from anomalous diffraction signals of heavier atoms. The phase information is extracted from the data and any prior information using advanced maximum likelihood distributions are combined with Fourier recycling between the crystal space and the diffraction reciprocal space. The procedure consists of several steps using different programs employed by `CRANK` (such as `BP3`, `Multicomb`, `and Refmac)` in which similar maximum likelihood functions are used. The speed of the pipeline is heavily dependent on the efficiency of the maximum

likelihood functions evaluation and the speed of the Fourier recycling. As a first step, an attempt was made to improve the speed of the `BP3` function evaluation by CPU parallelization and algorithmic improvements. While this was a successful step, it was not possible to generate a Xeon Phi version due to `CRANK` dependency on the `CCP4` library. While the library source code can be downloaded, it did not compile even for CPU; the compiler generated cryptic compilation errors that were indecipherable. In addition, `CCP4` also does not formally support Xeon Phi and the recommended use of the library is to download binary files which were only provided for x86(-64) target.

Given these issues, it was decided to not proceed any further, and instead make a claim that `CRANK` requires a substantial amount of porting efforts for Xeon Phi due to its dependency. The port for GPU is likely to be a major undertaking due to the low level link to the `CCP4` library which has no GPU version. Nevertheless, with sufficient funding and time, it is possible that this task can be successfully accomplished.

Despite these obstacles, `CRANK` was both algorithmically optimized and the original code parallelized and the results are shown in the following table:

	Application runtime (seconds)	
	1 core	8 cores
CPU (original)	60	---
CPU (tuned)	35	6

Table 1 CRANK Runtimes

4.2 The LUA Module System

The module command (environment-modules package) can be used to manage different software and library versions in a simple way. In the last decade it has become the de facto standard for managing application-specific environment variable settings in HPC systems. However, there are some limitations on how the command can deal with complex dependencies. Also the module command itself had a few outstanding bugs that crashed the user's session if wrong module files were loaded. It has seemed that module environments had become an orphaned project without any active development, which has further encouraged a search for alternatives.

It was decided to include our latest system upgrades the Lmod module system that was developed by Robert McLay at Texas Advanced Computing Center (project home page at TACC: https://www.tacc.utexas.edu/tacc-projects/lmod). TACC module system is a reimplementation of the original Tcl environment modules with several new features and different internal logic to ensure that the dependencies and conflicts between different compilers and MPI libraries are always forced. Lmod is implemented using the Lua scripting language and is under active development. From the end-user's perspective most common module commands look similar, but there are also major differences on the way how commands behave.

One additional feature of the Lmod system, that is useful with a system having a heterogeneous setup with ordinary and accelerator nodes, is "application properties". System administrators can add different properties to module files to mark which modules are compiled with support (for example GPUs or Xeon Phis). These properties can, for example, be used to mark which MPI libraries are compiled with GPUDirect support. We expect that

this will make the Lmod more powerful in a complex accelerator-enabled computing environment.

4.3 Bolt – AMD APU's

Bolt provides an STL (Standard Template Library) compatible library of high level constructs for creating accelerated data parallel applications. Code written using STL or other STL compatible libraries (example: TBB) can be converted to Bolt in minutes. In its open-source debut, Bolt supports C++ AMP in addition to OpenCL™ as underlying supported compute technologies. With Bolt, the kernel code to be accelerated is written in-line in the C++ source file. No OpenCL or C++ AMP API calls are required since all initialization of and communication with the OpenCL or C++ AMP device is handled by the library. Bolt requires significantly fewer lines of code and less developer effort. Bolt is straightforward to use and includes comprehensive documentation for the library routines, memory management, control interfaces, and host/device code sharing.

In addition to adding support for C++ AMP and CPU-only systems, Bolt also includes an array of Bolt capabilities by including support for common compute-optimized routines including sort, scan, transform, and reduce operations.

On an APU machine, the Bolt runtime library (based on the number of iterations of the given routine), tries to put the computation either on the GPU or CPU side of the APU. An implemented stream benchmark, using this library, proved to be very straight-forward, however, due to the dynamic switching between CPU and GPU cores the consistency of the result was very low.

The second test application was a shortest route finder implemented with a genetic algorithm. We used a mixture of library functions (like sorting functions) and self-written chunks of OpenCL code. The code worked but, because of lack of full HSA hNUMA implementation, the performance was not at the expected level.

The 0.1 version of the Bolt library that was tested worked only in a Windows environment and was not stable.

4.4 Heterogeneous System Architecture Standard

The HSA (Heterogeneous System Architecture) Foundation, which is a non-profit industry standards body, seeks to create applications that seamlessly blend scalar processing on the CPU, parallel processing on the GPU, and optimized processing on the DSP via high bandwidth shared memory access (which should enable greater application performance at low power consumption). The HSA Foundation defines key interfaces for parallel computations utilizing CPUs, GPUs, DSPs, and other programmable and fixed-function devices, thus supporting a diverse set of high-level programming languages and creating the next generation in general-purpose computing. The HSA architecture defines a set of hardware properties that have to be supported to be compatible with the HSA model. Being compatible means that the device (e.g. a GPU core) can be treated in the system as a specialized, general purpose core. As a result, HSA compatible compilers (C, C++, C++11, Fortran, Python, etc.) can directly generate a HSAIL code that can be executed on different kinds of devices. Starting from 2012, the AMD GPU units (both APU and discrete GPU) started implementing features required by the HSA standard. These features were utilized by higher level languages (e.g. OpenCL zero copy feature) but the changes were incremental, each generation presented a richer set of features. Until June 2014 the operating system of the HSA software stack was tightly bound to the graphics drivers which did not allow direct access to the HSA API. In June 2014 the first kernel module driver for AMD GPUs and APU units was included in the official Linux kernel source tree making writing a HSA compatible

code using dedicated libraries possible. The compiler support in the GCC compiler is currently under development and the first working version of the compiler, which will allow OpenMP parallelized applications to run on GPU cores, is planned to be available in 2015.

5 Resource Managers and Schedulers

5.1 Scheduler for Scalable Hybrid

The SLURM (Simple Linux Utility for Resource Management) is a free and open-source job scheduler system. SLURM development began in Lawrence Livermore National Laboratory in 2002. In the last decade its popularity has increased steadily and in the latest Top500-list 5 of the top 10 supercomputers used SLURM.

SLURM has a plugin interface which allows for new functionality to be added in a simple way as independent shared libraries. Due to architectural differences, the mechanism to support Xeon Phi and NVIDIA Tesla are different from each other.

The key component of the SLURM support on NVIDIA GPGPUs is the Generic Resource (GRES) –plugin called **gpu**. This controls that the GPUs are allowed for each job and no oversubscription of the GPUs will accidentally happen.

1. User submits a job with a request for 2 GPUs (--gres=gpu:2)
2. SLURM waits until there is a node with 2 GPUs available
3. SLURM starts the job and sets the CUDA_VISIBLE_DEVICES environment variable to point to the 2 free GPUs
4. The 2 GPUs are now reserved

To configure the system, the gres.conf configuration file must be added to the SLURM configuration directory. The file contains a description of each GPU resource and its corresponding device file.

Additionally one can include a CPUs parameter which controls what specific core in the system may use this resource. This is beneficial, for example, on a dual socket system where two GPUs are connected to different CPUs. With the parameter set properly, it is guaranteed that a process bound to a specific core will use the local GPU and not the one in a remote socket.

Below is an example gres.conf configuration:

```
Name=gpu File=/dev/nvidia0 CPUs=0-5
Name=gpu File=/dev/nvidia1 CPUs=6-11
```

One notable caveat in the configuration syntax is that the gres.conf file must be in the configuration path of SLURM, which on many systems is shared to ensure coherence of the main configuration (slurm.conf) –file across cluster nodes.

If all the nodes have an identical GPU configuration it is ok for gres.conf to be identical but for clusters also containing non-GPU nodes, this causes SLURM to halt. It is possible to work around this by having the gres.conf in the shared directory be a symbolic link to a local file that has a node-specific definition of the GPU resources. For example:

```
/share/slurm/gres.conf -> /etc/gres.conf
```

When the initial Scalable Hybrid prototype system from T-Platforms arrived, there was no support for the Xeon Phi in SLURM. In order to ease further application work initial support was developed for SLURM on the Xeon Phi during this project. The support was introduced by us to SLURM 2.5, released in November 2012 and has after that been refined by us as well as other SLURM open source community members.

During the development phase, three different mechanisms were experimented for supporting the Xeon Phi:

- GRES plug-in (offload model)
- Mpirun-mic helper script (native and symmetric model)
- Native SLURM on Xeon Phi (native and symmetric model)

The following sections cover each model in detail.

The **mic** GRES plugin functions in a fairly identical way as the GPU GRES plugin discussed previously.

For example:

1. User submits a job with a request for 2 Phis (--gres=mic:2)
2. SLURM waits until there is a node with 2 Phis available
3. SLURM starts the job and sets the OFFLOAD_DEVICES environment variable to point to the 2 free Phis
4. The 2 Phis are now reserved

The GRES plugin only works with the Xeon Phi offload programming model including LEO, OpenMP4, MKL offload and OpenCL.

The mpirun-mic helper script, found in SLURM's contrib directory, makes it considerably easier to launch serial Xeon Phi jobs or MPI jobs utilizing either Phis only or spanning Phis.

Below is an example of running a job on a system with 2 Phis per host, using:

- 16 MPI tasks per host node
- 2 threads per host task
- 60 MPI tasks per Xeon Phi
- 4 threads per Xeon Phi task
- CPU binary name: ./impi_native_hybrid
- Xeon Phi binary name: ./impi_native_hybrid.mic

```
export MIC_NUM_PER_HOST=2
export OMP_NUM_THREADS=2
export MIC_OMP_NUM_THREADS=4
mpirun-mic -v -x 16 -c ./impi_native_hybrid -z 60 \ –m
./impi_native_hybrid.mic
```

The aforementioned mpirun-mic helper script is run on the host and tasks are launched on the Xeon Phi via ssh. This approach has a number of drawbacks:

This causes the tasks on the Xeon Phi to be "loosely integrated" meaning that they are not under direct supervision and control of the SLURM launcher daemon. Due to this, job accounting does not work and stray processes may be left on the Phis as "zombies" as there is no strict cleanup if the job exits irregularly.

Furthermore, the syntax of the mpirun-mic is still far from being completely user friendly.

The ideal situation would be that the Xeon Phi native MPI tasks could be defined on the Xeon Phi just as the ones on CPU hosts.

5.1.1 *Native SLURM Daemon*

For these two reasons, porting the SLURM daemon natively to the Xeon Phi was also explored. The work itself was fairly time consuming as many dependent libraries were not available for the Xeon Phi and had to be cross-compiled.

The ported daemon worked as expected on the Xeon Phi and each Xeon Phi looked like a node on the cluster, only with a very large amount of cores. This also makes it possible to run different jobs on the host node CPUs and the Xeon Phi. This is potentially a good way to maximize the cluster utilization if there is a mixed workload of CPU and native Xeon Phi jobs running on the cluster. More performance studies are needed how this affects the performance of individual jobs running in such mixed nodes in practice.

Based on our experiences, the native SLURM daemon currently has three major drawbacks which should be addressed:

1. The mic GRES plugin on the hosts is not aware of the native daemon on the Xeon Phi. Thus if offload jobs are running on the host daemon and native jobs sent to the Xeon Phi daemon, it is likely that Phis get oversubscribed with both offload and native workload. Fixing this will likely require patching of the SLURM scheduling system.

   This is not a problem if the cluster runs only native and symmetric jobs or the offload jobs are segregated in separate set of nodes. However, in a cluster where jobs are expected to run with all Xeon Phi usage models on all nodes, this is a critical issue.

2. Being able to enforce affinity between hosts and Phis is not simple. By default one may end up reserving CPU cores and Phis from different hosts across the cluster. While on CPU hosts this is less of an inconvenience, the reduced bandwidth and increased latency of Xeon Phi-to-Xeon Phi communication makes it more critical to have all the tasks inside the minimum amount of hosts.
3. The memory consumption of the daemon is around 200-300MB. Due to the limited (8-16GB) memory on the Xeon Phi, this is already a considerable amount.

We found that it is possible to enforce affinity by setting up the topology plugin and defining each node and it's MICs as being under a single "switch". Example from a topology.conf file:

```
SwitchName=n2 Nodes=node02,node02-mic0,node02-mic1
SwitchName=n3 Nodes=node03,node03-mic0,node03-mic1
```

By default the scheduler will try to place each job into the same node if possible. It is also possible to force the jobs into a limited set of switches (queue if there is none available) using the `--switches` flag in salloc and sbatch.

5.1.2 *Conclusions*

SLURM currently supports both GPUs and technically all the different usage models of the Xeon Phi.

While GPU support is fairly simple to use and implement, Xeon Phi and its several usage modes makes the usage and implementation more complicated. At the moment it is possible to use the GRES plugin for offload jobs and the mpirun-mic wrapper scripts for native and symmetric runs.

A native SLURM daemon is a better solution than the wrapper scripts, but there are several technical issues to solve:

- Reduction of memory consumption
- Awareness of the mic GRES plugin on the host side and vice versa
- Simpler way to enforce host-Xeon Phi affinity

5.2 Scheduler for EURORA

This section will report and discuss the features and limitations of Altair PBSPro in terms of scheduling GPU and MIC jobs on the EURORA prototype.

PBS Professional (PBSPro) is a resource management system developed by Altair. It provides a common way, using jobs and queues, to deliver computing power to applications. Users submit jobs in the form of shell scripts that will run on the allocated resources. PBSPro provides support for standard (CPU, memory, walltime, etc.) and generic resources as well. Administrators can define a new generic resource describing its type (integer, float, Boolean, or string), its kind (consumable/non-consumable, static/dynamic) and its context (server, queue or node). This approach makes it easy to define almost any type of resource but it lacks in their control and usage tracking. As opposed to known typed resources such as cputime or memory for which PBSPro can, for example, enforce resource usage limits, for non-typed resources this must be implemented with other tools and/or custom scripts written ad hoc by system administrators. In this paper, the generic resource features for GPUs and MICs of version 12.2 of Altair PBSPro that is running on EURORA is discussed.

5.2.1 *NVIDIA GPU Scheduling*

On EURORA, a GPU is a consumable integer resource defined at node level: each GPU node has a resource ngpus=2. For scheduling purpose this configuration is acceptable: a job J requesting a GPU will be scheduled to one of the GPU nodes, the node allocated to job J will have its value of ngpus reduced by 1 and will be considered for scheduling only to other jobs requesting a single GPU, until job J is completed. However, because PBSPro does not provide a way to control the usage of the GPUs of the node allocated to job J, some issues arise:

- job J can use any GPUs of the node (0, 1 or both); and
- job J is not guaranteed the exclusive usage of the allocated GPU. Any other single GPU job scheduled to the same node of job J may use the same GPU of job J (or both the GPUs of the node).

From this point of view this configuration provides a very basic support of special resources such as GPUs or MICs.

An advanced configuration can be achieved using PBSPro virtual nodes (vnode). A vnode is an abstract object representing a set of resources which form a usable part of a machine. In a vnode based configuration each natural GPU node (nodeX) has two vnodes (nodeX-gpu0 and nodeX-gpu1) with ngpus=1 each representing the GPU0 and the GPU1 of the natural node. Since each vnode can be managed and scheduled independently, this configuration has the advantage of making it possible to identify which GPUs are assigned to a given job. For example, if the scheduler allocates the vnode nodeX-gpu0 of the nodeX to job J of the previous example, job J should be using GPU0 of nodeX. However, because PBSPro does not provide a way to bind the job to the GPU(s) assigned by the scheduler this configuration still suffers from the issues described above.

An additional possibility, tested at WCSS, is the control of device access via unix rights. By giving the job owner exclusive read/write access during prologue (setting 600 rights to a device) and removing them after job completion (epilogue), it can be guaranteed that one user has exclusive access to a device. But another job belonging to the same user could in fact also access the device. The use of *cgroups* might address this issue but this solution has not yet been tested.

Starting from CUDA version 3.1, NVIDIA provides a solution to address this scheduling issue. It is based on the environment variable CUDA_VISIBLE_DEVICE used for restricting execution to a specific device. Setting this variable to a comma-separated sequence of integers only the devices whose index is present in the sequence are visible to CUDA applications and they are enumerated in the order of the sequence.

Unfortunately this solution cannot be implemented on PBS-like resource management systems, such as PBSPro, because the environment of a PBS job is the same for all the processes launched within the job script. This means that a two processes job scheduled to use nodeX-gpu0 and nodeY-gpu1 could not have different values of this variable for processes running on nodeX and nodeY, where it should be CUDA_VISIBLE_DEVICE=0 and CUDA_VISIBLE_DEVICE=1, respectively. It would only work in the unlikely event where the same GPU index (e.g. GPU0) is selected on all of the nodes assigned to a job.

Not all resource management systems are affected by this issue. For example, the Simple Linux Utility for Resource Management (SLURM) provides a specific generic resource plugin for GPUs that sets the correct environment for each job process to determine which GPUs are available for its use on each node.

An alternative to the variable CUDA_VISIBLE_DEVICE that may work with PBS-like systems was developed by the National Center for Supercomputing Applications (NCSA) in 2009. The CUDA wrapper library is implemented as a preloaded library. As its name suggests, the library intercepts the device allocation calls to CUDA in order to virtualize the natural GPU devices and provide NUMA affinity mapping between CPU cores and GPU devices. By using an appropriate job prologue script, the administrator can set up the library to make only the allocated GPUs visible to the processes of a job. This solution can be implemented with PBSPro and it is in a testing phase on EURORA. However, it seems that this project is no more active or at least its last modification time dates back to 2012.

5.2.2 *Intel MICs Scheduling*

Considering the common features of almost any resource manager system, Intel MIC coprocessors are easier to manage than GPUs because they are based on x86-compatible multiprocessor architecture. The Intel Manycore Platform Software Stack (MPSS) provides the necessary software to configure and run the MIC cards. The MPSS command micctrl is used to configure the MIC kernel, file system, and network as well as boot, reset, shut down, and get the status of the MIC cards. By using this command in the prologue/epilogue script of the resource manager it is possible to switch on/off the MIC cards required to run a job.

On EURORA, each MIC natural node (nodeX) has two vnodes (nodeX-mic0 and nodeX-mic1) representing the two MIC cards. PBSPro is configured to assign a generic resource nmic=1 to each vnode nodeX-micY (Y=0,1), the corresponding natural node nodeX having nmics=2. Jobs requesting MICs are scheduled to MIC vnodes making it possible to identify the MIC cards assigned to a job on each allocated node. Using the micctrl command, it is possible to set up and boot the MIC cards on each node before job execution (prologue) and to reset them after job completion (epilogue).

On EURORA, every MIC card is switched off by default. Those selected for a job are switched on when a job begins its execution and they are switched off again after the job is completed. For example, suppose a user U of group G submits a job requesting two MICs and the scheduler assigns vnodes nodeX-mic0 and nodeY-mic1 to the job. Before the job begins its execution on nodes nodeX and nodeY, the prologue script running on the two natural nodes executes the following operations to set up the assigned MICs (MIC0/MIC1 on nodeX/nodeY):

1. it adds user/group credentials to the MICs filesystem (e.g. on nodeX, micctrl --useradd=U … mic0 & micctrl --groupadd=G … mic0)
2. it boots the MIC cards waiting for completion (e.g. on nodeY, micctrl --boot --wait mic1)

Once the job is completed the epilogue script restores the default configuration:

1. it resets the MIC cards waiting for completion (e.g. on nodeY, micctrl --reset --wait mic1)
2. it removes user/group credentials from the MICs file system (e.g. on nodeX, micctrl --userdel=U mic0 & micctrl --groupdel=G mic0)

This approach ensures that only processes belonging to the job of user U can access the MIC cards assigned to that job. Moreover, it reduces the power consumption of a node by about 210 Watts, when both MIC cards are powered off. However, there is a major drawback. We have measured the time needed for booting (resetting) a MIC card to be approximately one minute.

Since the hook is very simple, its Hook_execution_time depends on the MIC booting/resetting time only. In the presence of hooks, PBSPro schedules jobs in a serial way, which means that, given N pending jobs requesting MICs, that could be assigned free resources at a given time, the N^{th} job will have to wait (N-1)*Hook_execution_time before being dispatched to the selected node/s! This problem has been observed on a cluster with 64 nodes, and it is to be expected to be accentuated on a cluster with a higher number of nodes.

5.3 Scheduler for CPU/GPU

The prototype is using SLURM as a scheduler. It supports both scheduling based on generic resource types (NIC, CPU, and memory) and is also capable of using GPUs as a resource. 40 nodes are configured as a single queue with no distinct GPU queue. A user can however specify that he/she wants to use 0, 1 or 2 GPUs when submitting the job. Currently it is not possible to share GPUs between different jobs. Each job is assumed to have exclusive access to its GPU card or cards.

The scheduler has been configured in a way that unused nodes are put to sleep after a configured period of time and are waken up when additional resources are needed. Nodes with GPU units are not being shut down because in some cases GPU drivers handle the wake-up procedure in an incorrect way causing system freeze.

In addition the queue system has been provided with customized prolog and epilogue scripts that adjust settings of the cooling loop when jobs are starting or finishing.

6 System Management Tools

6.1 Power Data Aggregation Monitor (PowerDAM)

Power Data Aggregation Monitor (PowerDAM) [2][3][4] developed at Leibniz Supercomputing Centre (LRZ) is a unified energy measurement and evaluation toolset. It is aimed towards collecting and correlating energy consumption-relevant data from different aspects of the High Performance Computing (HPC) data center. This collected data includes *environmental* information (e.g. outside temperature, humidity level, etc.), information on *site infrastructure*

[2] Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Petra Piochacz. Towards a unified energy efficiency evaluation toolset: an approach and its implementation at Leibniz Supercomputing Centre (LRZ). ICT4S 2013: Proceedings of the First International Conference on Information and Communication Technologies for Sustainability, pages 276-282, http://dx.doi.org/10.3929/ethz-a-007337628 , 2013.

[3] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Monitoring power data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers. Environmental Modeling & Software, Elsevier, volume 56, pages 13-26, http://www.sciencedirect.com/science/article/pii/S1364815213002934 , 2014.

[4] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. A Path To Energy Efficient HPC Datacenters. HPCWire, http://www.hpcwire.com/2013/10/29/path-energy-efficient-hpc-datacenters/ , 2013

(e.g. cold and warm water cooling loops, etc.), information on *IT systems* (e.g. compute node power, load, temperature, etc.), and information on running *applications* on those IT systems (e.g. starting and ending time stamps, utilized compute nodes, etc.). This data is then correlated in order to better understand the interactions between different components of the data center and to assess the current state of Key Performance Indicators (KPIs) (such as: Energy-to-Solution (EtS)[5][6] of applications; Power Usage Effectiveness (PUE); Energy Reuse Effectiveness (ERE); Water Usage Effectiveness (WUE); etc.[7]) for identifying the improvement areas and verifying the success of applied optimizations. Figure 1 illustrates the overview of the PowerDAM toolset.

6.1.1 *PowerDAM at PSNC*

PowerDAM was deployed as a part of the software monitoring stack installed on the PSNC prototype. It was meant to be used as an information tool for the users who would like to know energy consumption profiles of their applications.

The installation procedure was as simple as unpacking the package into a system folder on the CPU/GPU prototype head node. Since the prototype is running Debian-based Ubuntu Linux, it was necessary to prepare upstart scripts for this distribution.

The configuration was a complicated part of the deployment as, contrary to how such software works, basic programming experience is required as one has to implement a class in Python that feeds the database with the system status data. Installation and configuration of the database was automatic and went flawlessly.

[5] Timo Minartz, JulianM. Kunkel, and Thomas Ludwig. Simulation of power consumption of energy efficient cluster hardware. Computer Science - Research and Development, 25 (3-4):165-175, 2010.

[6] Arndt Bode. Energy to solution: A new mission for parallel computing. In Felix Wolf, Bernd Mohr, and Dieter Mey, editors, Euro-Par 2013 Parallel Processing, volume 8097 of Lecture Notes in Computer Science, pages 1-2. Springer Berlin Heidelberg, 2013.

[7] The Green Grid, http://www.thegreengrid.org , 2014.

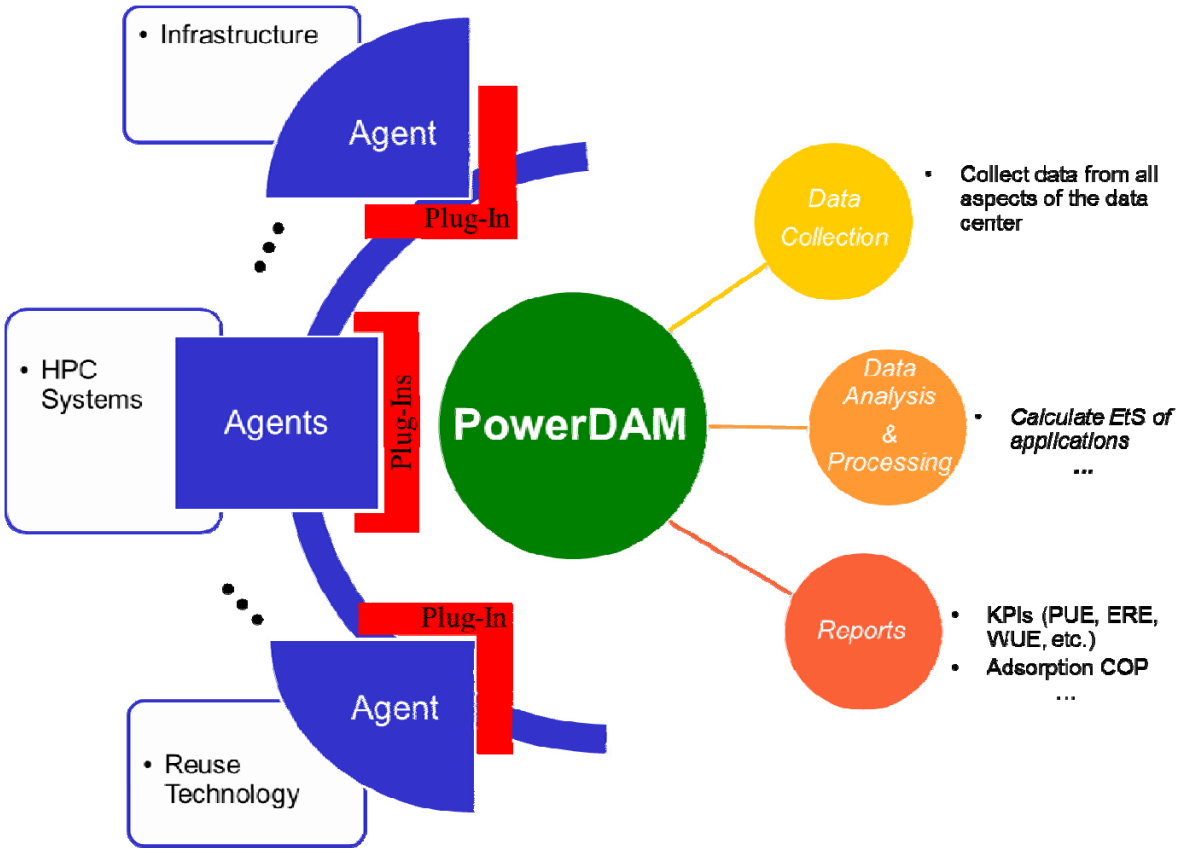

Figure 1 PowerDAM Overview

The toolkit is very flexible and can handle basically any readable form of sensors. In the PSNC case, the majority of the data that may be interesting for PowerDAM was already stored in a MySQL database so it was decided to have PowerDAM read from this rather than read the sensors locally. For some of the data, however, it was more convenient to read directly from the sensors; this turned out to be non-trivial. The data is gathered by running scripts remotely from the head node on the worker nodes. These scripts require root privileges but, by default, no cluster in PSNC supports password-less root command execution, therefore, a workaround using a special user with one command pseudo privileges was created.

In our prototype, PowerDAM monitors per node information about CPU load, temperature, and power consumption (DC). In addition data is being collected about power consumption of the networking part (InfiniBand and Ethernet switches) and cooling loop. Both internal cooling loops (in-rack pumps) and the external one are being monitored. All this data allows for an accurate assessment of the EtS for each SLURM job submitted to the system. It allows for a partial PUE calculation of the prototype.

The documentation provided with the package is fairly complete but was meant to be more as a development document rather than system administrator help. A complete sample set of configuration files would speed up the integration to a great degree.

The toolkit is an interesting software package that is still being actively developed, therefore, all integration issues are directly related to the „non-stable-release” status of the project.

6.1.2 *PowerDAM at CINECA*

The source code of PowerDAM was modified to make it work in push mode instead of the default get mode during the initial integration. However, this integration was dismissed since the data collection framework was directly connected with the monitoring database which PowerDAM now works on top of. PowerDAM was deployed on EURORA without significant change to the source code.

A PBS pro job scheduler plug-in was developed to work properly with PBS jobs and manage invalid jobs. The EURORA sensor reader plug-in was developed to filter and manage the data flow from the monitoring system and to make the sampling rates of a coarser grain so that PowerDAM is fed with sensor values averaged over a 1 minute interval.

The general impression of PowerDAM is that it is still weak in terms of the interaction with jobs and the ability to manage and customize the plugin infrastructure for the sensors could have been made simpler.

Regarding the statistics, PowerDAM works properly with coarse grained samples but, due to the 1 minute sampling interval limitation in the provided version, it can't be used for specific application power profile analysis where data needs to be collected and correlated at a higher sampling rate (milliseconds).

PowerDAM is in production on EURORA and is used to collect data for special purpose projects or benchmarks. However, there is still a minor problem that has not yet been identified that, from time to time, causes PowerDAM to hang, forcing a restart once a day as a workaround.

Finally, CINECA's overall evaluation of PowerDAM is positive and will be used as a tool in the CINECA HPC system software stack

6.1.3 *PowerDAM at CSC*

For the Scalable Hybrid system, integration with PowerDAM was not completed at the time of writing as the system arrived fairly late and there were limited PMs to complete the integration. The initial technical issues encountered were related to the fact that a fairly new version of the Python interpreter is required and a number of site-specific items are hardcoded in the application. This feedback was provided to the development team. The data collection was performed by extending the existing monitoring infrastructure at CSC based on the Graphite and *collectd* tools which was fairly straightforward. This is discussed in Section 6.2.3.

6.1.4 *Importance of Test Installations*

Thanks to PSNC's feedback the installation procedure and requirements will be an important focus area for further PowerDAM development.

Thanks to CINECA's feedback, the execution time of the used data base queries was improved. Also, the current alpha version allows for both a pull and push communication model. Additionally, the one minute sampling interval used in the pre-alpha version of PowerDAM will be removed in the newer version. This is done by allowing each sensor reading to provide its own timestamp to PowerDAM opposite to the PowerDAM defined 1min interval timestamps previously used.

Thanks to the CSC feedback, PowerDAM now supports python v2.6, v2.7, and v3.x. In addition, the improved installer checks for required 3rd party python libraries and for a supported python version before installation of PowerDAM.

6.1.5 *PowerDAM Summary*

The importance and applicability to the real-world power and energy controlling techniques of PowerDAM can already be seen in [8], which presents an adaptive model for application energy and power consumption prediction for a given number of compute nodes based on the PowerDAM monitored and correlated application history energy/power data.

PowerDAM is currently under active development and is available to all PRACE-2IP WP11 partners. Future work includes the addition of a push communication service which will allow the monitored entities for initiation of data collection request; in contrast to currently supported pull-only communication service where the data collection request is initiated by PowerDAM.

The future work also includes database backend improvement for ensuring the scalability for myriads of sensors which are foreseen with the next generation of HPC systems as well as the development of a web-based graphical user interface for easy access and better view of the PowerDAM data.

6.2 Monitoring

6.2.1 *Monitoring – EURORA*

Since the system did not come with an integrated self-protection system against overheating in case of cooling system failure, a set of scripts was created which read the temperature sensors present in the boards and in case of problems instructed the SNMP-capable power supply to cut the power. The protection mechanism takes into account boards and rootcards cold plate temperature and also the number of boards exceeding the temperature threshold or not being reachable. This system, even though still vulnerable to network problems, has proven to be safe enough to prevent overheating damage and as a consequence to guarantee a good quality of service preventing long interruptions in the production

Additionally, sensors to detect water leakage and excessive air temperature/humidity were installed in the system and configured to send alarm through the proprietary RFCode Assett Manager and the local SNMP trap system.

6.2.2 *Monitoring – CPU/GPU*

The state of the system is collected using in-house developed scripts that gather data about the physical state of the servers using IPMI commands. Some of the data (e.g. components temperature) is collected out-of-band using ipmitool command from the head node but others (e.g. per-node power consumption) must be collected by running the ipmicfg command from each node. The data about the GPU (load, temperature) is also collected by periodical execution of scripts on each node. For the GPU the desired sensors are being read using the CLI command aticonfig with --odgc / --odgt parameters.

All data is being read in 1 or 5 seconds intervals. Depending on how fast a given sensor can be queried, a 1 min average value is calculated and inserted into the MySql database. Data for rack power consumption and rack internal pumps is gathered using SNMP queries and also inserted into the same database.

The state of the cooling loop is monitoring using the MODBUS protocol. As MODBUS is a serial protocol transmitted over RS485 cable, a MODBUS to Ethernet gateway was used to

[8] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Predicting the Energy and Power Consumption of Strong and Weak Scaling HPC Applications. Submitted to the Supercomputing Frontiers and Innovations international journal.

allow remote data acquisition. All data gathered from the external cooling loop controller is being saved in the same database as the rest of the system information. Because of the speed limit of the MODBUS interface, the cooling loop may be queried once every 1 minute. After each query, the collected data is being analyzed and, based on the external temperature and current power consumption of the rack, appropriate values for the rack inlet and outlet temperatures are being calculated and sent to the controller. In addition to the above described data collection, a standard Ganglia monitoring system was installed to provide an on-line overview of the state of computing nodes. Standard Ganglia sensors were supplemented by per-node power consumption, CPU and InfiniBand chip temperatures, and GPU statistics.

A web page with on-line overview of the cooling loop state was created. It is publicly available at: http://brassica.man.poznan.pl

Each node is checking its temperature of all components in 1 minute intervals. If the temperature of any component exceeds the threshold of 75C the node is being shut down and corresponding information is being logged.

The threshold temperature, which is 15C below the thermal cut value of the CPUs, was selected basing on empirical experiments. It protects the Novec inside the modules from boiling and at the same time, it allows for 45C inlet cooling. If the Novec is at the correct level, the maximum temperature of any chip inside the node should not exceed 70C. In addition similar scripts are executed from the head node ensuring that the system is protected from overheating also when the OS on the node is not working properly.

6.2.3 *Monitoring (Graphite) – Scalable Hybrid*

The performance and environmental data on the Scalable Hybrid system is collected using the *collectd* (https://collectd.org/) daemon from various sources and analyzed using a graphing system called Graphite (http://graphite.wikidot.com/). Both tools are fairly widely used in the large-scale enterprise and web hosting environments and are now also becoming adopted by the HPC community.

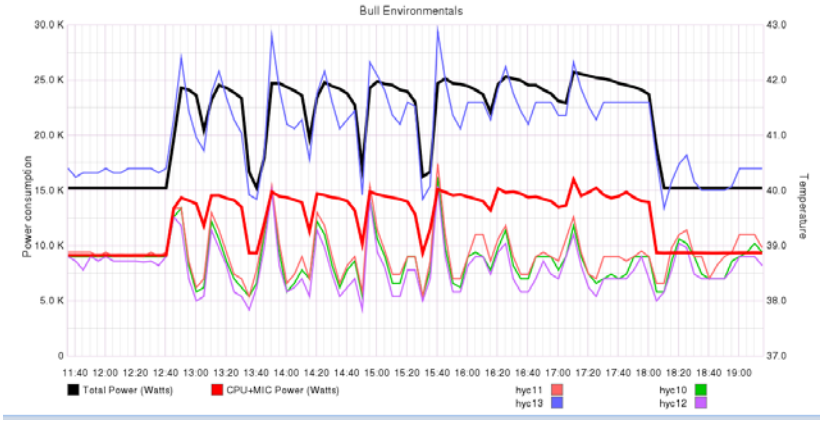

Figure 2 Example Graphite Chart

While Graphite was found extremely capable and useful for real-time monitoring of system metrics, its lack of capability in associating metrics with individual users and jobs limits its applicability. We worked around this problem by creating a set of scripts which, for a given SLURM job id, create an URL for Graphite. Copying this URL to the browser provides graphs (Figure 2) for a given parameter (for example GPU temperatures) on the nodes where the job was running during the time period when the job was running.

Some of the infrastructure data (for example rack inlet temperatures) were only available from the Siemens Desigo building automation system (see D11.1.3 “Prototyping and Technical

Evaluation Summary") which could not be integrated to this system. Thus, for the experiments, this data was exported to an Excel file and collated manually.

PowerDAM and Graphite could complement each other well, ideally, however, there should be only one data collector instance to limit the monitoring overhead. We recommend investigating how these two tools could leverage the same data source.

Furthermore, the infrastructure automation vendors should be encouraged to provide open interfaces to their systems for reading metrics easily and securely with third party tools such as Graphite or PowerDAM.

6.3 Provisioning

6.3.1 *Warewulf – Scalable Hybrid*

The Bull Scalable Hybrid prototype was provisioned using the Warewulf cluster provisioning system which is currently used in all CSC clusters. Warewulf was chosen due to its flexibility in supporting compute nodes with and without local disk and gives us the choice of which upper level tools to use for things such as monitoring the cluster.

Adding the support for GPGPUs only required adding the GPU driver and CUDA library to the standard compute node image.

For full support of the Xeon Phi, there are additional steps to be done during the node bootup, including:

- Loading the mic.ko driver
- Initializing the basic configuration with micctrl --initdefaults
- Setting up the bridge interface for the Ethernet
- Setting the IP addresses on the Xeon Phi cards
- Setting up NFS and/or Lustre shares
- Booting the Xeon Phi

This functionality is included as a set of boot-time scripts in a separate package called warewulf-mic[9]. When setting up the system initially it was found that the then latest version of the package was designed for an older version of MPSS (2.1) and the newer versions (3.x) were not compatible with it anymore. This required us to do extensive changes to the initialization scripts. Since then, the package has been updated to support the newest versions of MPSS.

6.3.2 *Provisioning – CPU/GPU*

The APU units are booted from local hdd drives and no provisioning system is used. In case of the cluster all nodes are booted in a disk-less mode despite being equipped with 60GB local ssd drives.

There is no out-of-the box system installed for provisioning the nodes. An in-house developed environment is being used with some additional scripts prepared to fulfill the requirements of the GPU drivers.

The nodes are booted using PXE protocol and the /boot file system is exposed using TFTP protocol. On both APU nodes and cluster nodes users are authorized using NIS protocol.

[9] http://warewulf.lbl.gov/downloads/releases/warewulf-mic/

6.4 Management System

6.4.1 *xCAT – EURORA*

The cluster has been integrated with the xCAT management system for the installation.

All the boards were provided with PXE network and were working immediately with the pxe installation method provided with xCAT. Additional custom scripts have been developed to install and configure the Intel Xeon Phi software stack due to its heterogeneous composition (rpm packages and setup scripts).

Some difficulties also came from the configuration of the BMC (Board Management Controller) which shared the Ethernet port with the node and required Ethernet Trunking to be configured on both the switch and the nodes.

Also the lack of a backup battery in the nodes caused the loss of BIOS and BMC settings every time the system was electrically powered off and basically required the system to be reinstalled and the BMC to be reconfigured after every power cut.

7 Parallel Programming Environment

7.1 OpenCL

OpenCL[10] (Open Computing Language) is an open standard for parallel programming of heterogeneous computing systems. It is composed of an API and a standard language to write portable code for multi-core CPUs, GPUs, APUs and other architectures, including latest Intel Xeon Phi accelerators. OpenCL kernels are written in a subset of the ISO C99 language that is compiled at runtime to target a particular computing device. The OpenCL standard supports both data- and task-based programming models. This technology allows the code prepared for Intel Xeon processors to be also run on the Intel Xeon Phi after small changes[11]. It might result in sub-optimal performance though. Intel released the new Intel SDK for OpenCL Applications (XE 2013 Beta), which provides a development environment for OpenCL 1.2 applications across both Intel Xeon processor and Intel Xeon Phi coprocessor. This SDK includes code samples, development tools, an optimization guide, support for optimization tools, and OpenCL runtime for Intel CPUs and Intel Xeon Phi coprocessors.

The standard supports a wide range of CPUs, GPUs, DSPs and other processors. It can be used on large-scale systems but only when used with additional communication technique, like MPI. There is a number of tools supporting the code development (e.g. Intel SDK for OpenCL Applications, ARM Mali OpenCL SDK, AMD Accelerated Parallel Processing (APP) SDK, NVIDIA GPU Computing SDK, IBM OpenCL Development Kit).

The latest version of the standard, OpenCL v2.0[12], was announced in 18 March, 2014.

The hardware (NVIDIA K20 GPUs and Intel Xeon Phi accelerators) available on EURORA support OpenCL in version 1.1 and hardware available on CPU/GPU (AMD FirePro S9000 GPUs) support OpenCL in version 1.2.

We ported the SHAKE and RATTLE algorithms from DL_POLY application. The RATTLE OpenCL algorithm shows up to 11x speedup for the test benchmark with 413896 atoms. Some performance bottlenecks of SHAKE OpenCL implementation have been identified -

[10] OpenCL Khronos Group homepage, http://www.khronos.org/opencl/
[11] PRACE Public deliverable, D7.2.1, “A Report on the Survey of HPC Tools and Techniques”, 2013, pdf: http://www.prace-ri.eu/IMG/pdf/d7.2.1.pdf
[12] The Khronos OpenCL Registry with OpenCL 2.0 specification, http://www.khronos.org/registry/cl/

initialization of the OpenCL environment, GPU I/O operations and synchronization between MPI processes in a multi-GPU environment.

Also the DBSCR library from CP2K code has been ported to the OpenCL. The OpenCL results show a better performance to the other methods tested (OpenACC implementation, PGI SMM and GFortran SMM) if bigger data volume is computed on accelerator (i.e. GPU). In comparison with OpenACC developing OpenCL code needs more effort.

7.2 OpenACC

OpenACC [13]is a directive-based open standard designed to simplify parallel programming of heterogeneous CPU/GPU systems. It is supported by NVIDIA, PGI, Cray, and originally CAPS. The developer can annotate C, C++ and Fortran source code to identify the areas to be accelerated using #pragma compiler directives and additional functions. It is portable across operating systems, and multi-core processors such as NVIDIA and AMD GPUs, and Intel Xeon Phi, but the range of target options depends on the compiler used.

The latest version of the standard, OpenACC 2.0a, was announced on August 31, 2013.

Introducing OpenACC into the code requires usage of a compiler which understands the OpenACC pragmas. There are only three such compilers so far, all commercial, delivered by PGI, CAPS and Cray. In some cases, for applications written in Fortran which use Fortran 2008 extensions, it may be not straight-forward to successfully compile with PGI (as reported for CP2K[14]). The PGI 14.1 available on EURORA (and the newest version 14.3) does not fully support the Fortran 2008 extensions, and further problems may occur when compiling with optimization flags. Some disadvantage is also a long compilation time, compering to other compilers (e.g. GNU gfortran), but it is compensated by a very good optimization and short execution time. Thus, how easy is to introduce and use OpenACC depends on the combination of the programming environment and underlying software stack.

At the same time, the CAPS Compiler Suite 3.4.5 for OpenACC is available, but the company is no longer issuing licences. In May 2014 an official statement of CAPS has been announced to all its customers, that the company will be closed due to financial problems, and no support will be provided after the end of the June, 2014.

It must be noticed that the power of the OpenACC standard strongly depends on the compilers' support. The current choice of proprietary compilers seems to be not sufficient, especially in the current landscape of scientific applications which often are open-source and developed using open tools and compilers like GCC. This may lead to compatibility issues, as described for CP2K. Nevertheless, OpenACC once introduced to the code, in connection with the compiler support for different targets, is a powerful technology. The OpenACC directives may be used by a compiler to generate kernels for new emerging architectures, as soon as they become supported.

As an additional conclusion, introducing OpenACC to an existing application is relatively simpler and requires less knowledge and time from the developer than OpenCL. However, it still requires a good understanding of the application, its data and algorithms, and may require refactoring of the original code to gain a performance as expected from the GPU acceleration.

The DBSCR library from CP2K code has been ported to the OpenACC. OpenACC directives are supported by only a few compilers, and the mostly used are Portland Group's PGI Accelerator compilers. A number of different issues were identified when building CP2K

[13] The OpenACC 2.0a Specification (Corrected), pdf: http://www.openacc.org/sites/default/files/OpenACC.2.0a_1.pdf

[14] PRACE Public deliverable, D7.2.1, "A Report on the Survey of HPC Tools and Techniques", 2013, pdf: http://www.prace-ri.eu/IMG/pdf/d7.2.1.pdf

with the pgfortran from the PGI compiler suite, including non-implemented FORTRAN functions and a segmentation fault. Slow compilation when using PGI has been a disadvantage. The results show that PGI compilers provide very good automatic code optimizations, which lead to reducing the computation time. The code compiled with Gfortran and the SMM library for performing matrix operations, runs slower than the code compiled with PGI, without the specialized library Using OpenACC gives very good results, although, the code compiled with PGI and SMM library shows similar performance.

8 Summary

Overall, getting a functional and stable software stack for a prototype is challenging at best. In many cases the software support takes much longer to mature and support new hardware than the procurement and installation of the hardware. Heterogeneous architectures add more complexity to the software stack installation by adding additional software dependencies.

Currently, OFED (OpenFabrics) and MPI support by the accelerators is restricted to specific versions which might interfere with the versions for the HPC system in which the accelerators are installed.

Even though GPU accelerators are nothing new for HPC the integration into the system scheduler is still challenging.

Currently, SLURM supports GPUs but requires a work around if not all nodes look the same (same number of accelerators per node). Intel Xeon Phi support didn’t exist before and was, therefore, developed as part of this task. It is has become part of SLURM package since version 2.5.

The PBSpro scheduler provides only basic support for GPUs. The Xeon Phi support was implemented via prologue and epilogue scripts, however, this solution will not scale.

Overall, current schedulers lag advanced scheduling support for accelerators (being on-par with what’s possible for CPUs).

Support for newer versions of parallel programming environments (like OpenCL, OpenACC, etc.) on new architectures depends strongly on compilers and vendor support. Therefore, it is very hard to judge when such an environment will be stable for software development on new architectures.