D5.11– Final report on porting and tuning of system software to ARM architecture
Version 1.0

Document Information

<table>
<thead>
<tr>
<th>Contract Number</th>
<th>288777</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Website</td>
<td><a href="http://www.montblanc-project.eu">www.montblanc-project.eu</a></td>
</tr>
<tr>
<td>Contractual Deadline</td>
<td>M45</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>PU</td>
</tr>
<tr>
<td>Nature</td>
<td>O – agreed by the WP as written report</td>
</tr>
<tr>
<td>Coordinator</td>
<td>Filippo Mantovani (BSC)</td>
</tr>
<tr>
<td>Contributors</td>
<td>Filippo Mantovani (BSC), Daniel Ruiz (BSC), Oriol Vilarrubi Barri (BSC), Xavier Martorel (BSC), Diego Nieto (BSC), Axel Auweter (LRZ), Daniele Tafani (LRZ), Chris Adeniyi-Jones (ARM), Hervé Gloaguen (Bull/Atos), Gladys Utrera Iglesias (BSC)</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Chris Adeniyi-Jones (ARM)</td>
</tr>
<tr>
<td>Keywords</td>
<td>System software, OmpSs, programming model, cluster management, debugging tools, performance tools, prototype evaluation</td>
</tr>
</tbody>
</table>

**Notices:** The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 288777

©2011 Mont-Blanc Consortium Partners. All rights reserved.
<table>
<thead>
<tr>
<th>Version</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1</td>
<td>Initial version (Filippo)</td>
</tr>
<tr>
<td>v0.2</td>
<td>Added Developer Tools section (Dani)</td>
</tr>
<tr>
<td>v0.3</td>
<td>Added table for describing the platforms (Filippo)</td>
</tr>
<tr>
<td>v0.4</td>
<td>Modified structure of contents in Scientific Libraries section (Filippo)</td>
</tr>
<tr>
<td>v0.5</td>
<td>Added Kernel information (Chris)</td>
</tr>
<tr>
<td>v0.6</td>
<td>Added information about Puppet (Oriol)</td>
</tr>
<tr>
<td>v0.7</td>
<td>Added information about DVFS (Hervé)</td>
</tr>
<tr>
<td>v0.8</td>
<td>Added power monitoring section (Daniele)</td>
</tr>
<tr>
<td>v0.9</td>
<td>Added information about runtimes except for Nanos++ subsection (Dani)</td>
</tr>
<tr>
<td>v0.10</td>
<td>Added information about storage (Dani)</td>
</tr>
<tr>
<td>v0.11</td>
<td>Added information about the Cluster Management applications (Dani)</td>
</tr>
<tr>
<td>v0.12</td>
<td>Added configuration information about several applications (Dani)</td>
</tr>
<tr>
<td>v0.13</td>
<td>Added performance study of OpenMPI (Dani)</td>
</tr>
<tr>
<td>v0.14</td>
<td>Added information about Mercurium (Xavi)</td>
</tr>
<tr>
<td>v0.15</td>
<td>Added information about Nanos++ (Diego)</td>
</tr>
<tr>
<td>v0.16</td>
<td>Added Hardware/Software test and configuration section (Dani)</td>
</tr>
<tr>
<td>v0.17</td>
<td>Added OS noise section (Gladys)</td>
</tr>
<tr>
<td>v0.18</td>
<td>Added Executive Summary and Introduction plus minor fixes (Filippo)</td>
</tr>
<tr>
<td>v0.19</td>
<td>Included comments of Jesus (Xavi+Filippo)</td>
</tr>
<tr>
<td>v0.20</td>
<td>Integrated fixes to Jesus’s comments (Filippo)</td>
</tr>
<tr>
<td>v0.21</td>
<td>Added comment about clBLAS (Diego)</td>
</tr>
<tr>
<td>v1.0</td>
<td>Integrated comments from internal reviewer / ready for submission</td>
</tr>
</tbody>
</table>
Contents

Executive Summary 5

1 Introduction 7

2 Platforms 8

3 Compilers 8
   3.1 GNU compiler suite .................................................. 8
   3.2 Mercurium ............................................................. 8
   3.3 JDK ................................................................. 9

4 Scientific libraries 9
   4.1 ATLAS ................................................................. 9
   4.2 Boost ................................................................. 10
   4.3 clBLAS ................................................................. 10
   4.4 clFFT ................................................................. 10
   4.5 HDF5 ................................................................. 10
   4.6 FFTW ................................................................. 10
   4.7 LAPACK ............................................................. 11
   4.8 PETSc ............................................................... 11

5 Developer tools 11
   5.1 Perf ................................................................. 11
   5.2 Extrae ............................................................... 11
   5.3 Scalasca ............................................................ 11

6 Runtimes 12
   6.1 OpenCL ............................................................ 12
   6.2 CUDA ............................................................... 12
   6.3 MPI ................................................................. 12
      6.3.1 OpenMPI ....................................................... 12
      6.3.2 MPICH ......................................................... 13
   6.4 Nanos++ ........................................................... 13
      6.4.1 OmpSs ......................................................... 15
      6.4.2 OmpSs+OpenCL .............................................. 15
      6.4.3 OmpSs+CUDA ............................................... 16

7 Hardware support 16
   7.1 Kernel 3.11.0 ....................................................... 16
   7.2 DVFS ............................................................... 17
   7.3 Power Monitor ..................................................... 17
      7.3.1 Components of the tool and their configuration .......... 18
      7.3.2 Usage .......................................................... 23
      7.3.3 Verification of the monitoring tool in the Mont-Blanc prototype . 24
8 Storage
8.1 NFS ............................................................... 26
8.2 Lustre ............................................................ 26

9 Cluster management
9.1 OpenLDAP ....................................................... 27
9.2 NTP ................................................................. 27
9.3 Puppet ............................................................. 27
9.4 Nagios ............................................................. 28
9.5 Ganglia ............................................................ 28
9.6 SLURM .............................................................. 28
   9.6.1 Power monitor plugin ..................................... 29

10 Hardware/Software test and configuration
10.1 Thermal experiments ........................................... 29
10.2 OS noise analysis ................................................ 33
Executive Summary

This document describes the status of the system software stack within the Mont-Blanc project. The work of populating a complete software stack for HPC and scientific computing has been performed since the beginning of the Mont-Blanc project (see deliverables D5.3 and D5.5).

In this deliverable we report the work related to the third year and the extension of the project. As during this period the project deployed the Mont-Blanc prototype, based on 1080 SoCs each with dual core CPUs + embedded mobile GPU, the effort has been focused in porting and tuning the Mont-Blanc system software, shown in Figure 1, to our final platform.

Figure 1: Graphical representation of the Mont-Blanc system software stack deployed on the Mont-Blanc prototype and other ARM-based platforms involved in the project

The key achievements reached within the project, making the Mont-Blanc system software unique in the panorama of the HPC software stacks are:

- Full support for the ARM Mali-T604 GPU, including OpenCL 1.1 full-profile under Linux. This not only allows all the partners of the project to execute their standard OpenCL code on the Mont-Blanc heterogeneous computational nodes, but also the testing of new coherent programming paradigms, due to the capability and the software support of sharing memory between CPUs and GPU. The presence of ARM in the consortium has been a key point to achieve this support.

- Porting and deployment of the OmpSs programming model to all ARM-based Mont-Blanc platforms, including the large prototype of 1000+ heterogeneous nodes. This allowed the OmpSs developers to test the programming model in a unique environment composed of commodity heterogeneous technology and parallel HPC applications. The community of OmpSs users also grew as a result of the project, exposing more and more scientists to the task based programming model developed within BSC.
- Porting, testing and deployment of the debugging and productivity tools, Extrae, Paraver and Scalasca on all ARM-based platforms forming part of the prototype ecosystem of Mont-Blanc. This enabled the performance analysis of many scientific applications, consolidating the role of these established tools in the HPC community.

- Particular attention has been dedicated to maintain compatibility whilst optimizing the Mont-Blanc system software not only on the Mont-Blanc prototype, but also on all the ARM-based platforms involved in the project (including Mont-Blanc 2). The idea behind this effort is to maintain the whole system software ecosystem, mostly based on open-source software, in a generic manner, and make it available to the scientific community and in particular to the research groups or companies willing to install an ARM-based system to be devoted to scientific computing.

In order to achieve this idea, the system-administration Mont-Blanc team adopted Puppet, a provisioning software, capable of maintaining complex software configurations over different platforms through a modular system of scripts. Part of this deliverable is therefore also the set of scripts, made publicly available through github, for the automatic configuration of the Mont-Blanc system software on a generic ARM-based platform running Linux and connected to a network.
1 Introduction

This deliverable is organized as follows:

- Section 2 summarizes the platforms that have been used as test platforms for the deployment and tuning of the Mont-Blanc system software.

- Section 3 is dedicated to compilers, with special attention in reporting the activities performed within the project on the Mercurium compiler.

- Section 4 is dedicated to the scientific libraries.

- Section 5 gives an overview of the development and productivities tools ported to the Mont-Blanc prototype.

- Section 6 gives highlights of the runtimes running on the Mont-Blanc prototype, with special attention to Nanos++, forming part of the OmpSs programming model and to OpenMPI, for which we performed a preliminary study of configuration parameters.

- Section 7 summarizes the work that has been done within the project in order to fully support the hardware features of the SoC selected for the prototype (Kernel, DVFS) and the power monitoring tool developed within the project for monitoring the power consumption and other vital parameters of the prototype.

- Section 8 reports about the two distributed filesystems tested on the Mont-Blanc platforms: NFS and Lustre. Concerning Lustre, please refer to D5.10 for more details.

- Section 9 is dedicated to the software for cluster management and monitoring. This part includes the description of the package of script based on Puppet, for smart automatic deployment and maintenance of ARM-based platforms for scientific computing.

- Section 10 finally summarizes the set of tests performed on the final installation at BSC of the Mont-Blanc prototype. This section includes report on extensive thermal tests and a study on operating system noise on the nodes of the Mont-Blanc prototype.
2 Platforms

Table 1 shows the different platforms we have been experimenting in the Mont-Blanc project beside the Mont-Blanc prototype installed at BSC and described in D7.8.

CARMA (PRACE funds) and Arndale are mini-clusters developed for enabling system software porting in the early days of the project. Pedraforca (PRACE funds) and Odroid XU are mid-size clusters developed for early scalability studies. Jetson and Odroid XU-3 are mini-clusters developed within Mont-Blanc2.

<table>
<thead>
<tr>
<th>Prototype Name</th>
<th>CPU</th>
<th>GPU</th>
<th>N. of nodes</th>
<th>Additional info</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARMA</td>
<td>4x ARM Cortex-A9</td>
<td>Nvidia Quadro 1000M</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Arndale</td>
<td>2x ARM Cortex-A15</td>
<td>ARM Mali T-604</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Pedraforca</td>
<td>4x ARM Cortex-A9</td>
<td>Nvidia Tesla K20</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>Odroid XU</td>
<td>4x ARM Cortex-A15 4x ARM Cortex-A7</td>
<td>PowerVR SGX544MP3</td>
<td>24</td>
<td>big.LITTLE IKS [big13]</td>
</tr>
<tr>
<td>Jetson</td>
<td>4-plus-1 ARM Cortex-A15</td>
<td>Kepler GPU</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Odroid XU-3</td>
<td>4x ARM Cortex-A15 4x ARM Cortex-A7</td>
<td>ARM Mali T-628</td>
<td>6</td>
<td>big.LITTLE GTS</td>
</tr>
<tr>
<td>Mont-Blanc prototype</td>
<td>2x ARM Cortex-A15</td>
<td>ARM Mali T-604</td>
<td>1080</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Mont-Blanc platforms

More details concerning Mont-Blanc platforms are available on the Mont-Blanc web page: http://montblanc-project.eu/arm-based-platforms, while a description of the Mont-Blanc prototype is given in D7.8.

3 Compilers

3.1 GNU compiler suite

**URL:** https://gcc.gnu.org/

**Description:** The GNU Compiler Suite includes front ends for C, C++ and FORTRAN as well as libraries for these languages. Support for ARM is included, providing the users with the needed tools to compile their applications.

**Platforms:** On our platforms we installed this software by using the default repositories of the cluster Linux distribution. This is not the case for the Mont-Blanc prototype, where several additional versions of the suite have been compiled and installed, including versions 4.9.0, 4.9.1, 4.9.2 and 5.1.0.

3.2 Mercurium

**URL:** http://pm.bsc.es/mcxx

**Trac:** https://pm.bsc.es/projects/mcxx

**Description:** The Mercurium compiler is the OmpSs compiler developed at BSC. It supports C, C++, and FORTRAN, transforming the OmpSs directives found in the source code
into calls to the Nanos++ runtime system. The compiler also supports targeting CUDA and OpenCL devices.

**Platforms:** On our platforms we installed this software globally accessible by all users in the /apps filesystem. Currently available versions include the stable mcxx-1.99.4 (Sep. 2014), mcxx-1.99.6 (Feb. 2015), and the development mcxx-1.99.8 (May 2015).

The Mercurium compiler has been partially developed and improved in this project. We started with the compiler that resulted from the previous European Project EnCORE (http://www.encore-project.eu), which supported C and C++, and the main development has been the addition of the FORTRAN support. We managed to incorporate a common internal representation supporting the 3 languages. During the project, we have been working on the following main lines:

- Front-End and Parser, to be able to compile the FORTRAN applications provided by the partners of the project. Mercurium now supports fixed and free-form FORTRAN programs.
- Code transformations, to follow the OmpSs directives and transform the code with calls to the Nanos++ runtime system.
- OpenCL support, we provided the transformations needed to allow invoking OpenCL kernels from the FORTRAN language. This included the proper parameter passing to the Nanos++ runtime system to create the memory buffers for implementing the data copies to the Mali GPU.
- ARM environment, the compiler has a component that describes the data types supported by the target architecture. The ARM 32-bit environment was added during this development.

3.3 JDK

**URL:** [http://www.oracle.com/technetwork/java/javase/overview/index.html](http://www.oracle.com/technetwork/java/javase/overview/index.html)

**Description:** The Java Development Kit (JDK) includes tools useful for developing, testing and running programs written in Java programming language.

**Platforms:** Version 1.7.0_60 is installed at the Mont-Blanc prototype. It has been tested with the Hadoop experiments – reported in D5.12.

4 Scientific libraries

4.1 ATLAS


**Description:** The Automatically Tuned Linear Algebra Software (ATLAS) provides with C and Fortran77 interfaces.

**Platforms:** Version 3.11.30 with Lapack support installed on all the platforms.

**Configuration:** Architecture discovery from ATLAS has been done to improve the performance of the library, achieving between 90% and 95% of the peak performance of each core on all the platforms when using this library.
4.2 Boost

**URL:** [http://www.boost.org/](http://www.boost.org/)

**Description:** Boost C++ Libraries are meant to provide the user with a set of widely useful functionality and be and usable across a broad spectrum of applications.

**Platforms:** Version 1.58.0 has been compiled and installed from sources on all of our clusters.

4.3 clBLAS

**URL:** [https://github.com/clMathLibraries/clBLAS](https://github.com/clMathLibraries/clBLAS)

**Description:** clBLAS, which is a portion of clMath, provides an OpenCL implementation of BLAS. The complete set of BLAS level 1, 2 and 3 routines are implemented. Some of the kernels are automatically generated based on the target architecture, reaching 8 GFLOPS in single precision and 2 GFLOPS in double precision for e.g. GEMM routines.

**Platforms:** It is currently installed on the Mont-Blanc prototype.

4.4 clFFT

**URL:** [https://github.com/clMathLibraries/clFFT](https://github.com/clMathLibraries/clFFT)

**Description:** This library includes OpenCL implementations of FFT functions. It also support running on CPU devices to facilitate debugging and heterogeneous programming.

**Platforms:** It is currently installed on the Mont-Blanc prototype.

4.5 HDF5

**URL:** [https://www.hdfgroup.org/HDF5/](https://www.hdfgroup.org/HDF5/)

**Description:** HDF5 is a data model, library and file format for storing and managing data. It is designed for flexible and efficient I/O and for high volume and complex data.

**Platforms:** The version 1.8.13 of HDF5 with MPI support is currently installed in all of our platforms.

4.6 FFTW

**URL:** [http://www.fftw.org/](http://www.fftw.org/)

**Description:** The Fastest Fourier Transform in the West (FFTW) C library provides functions to compute the discrete Fourier Transform in one or more dimensions.

**Platforms:** Versions 3.3.4 and 2.1.5 have been installed on our platforms with both double and single precision support.
4.7 LAPACK

**URL:** [http://www.netlib.org/lapack/](http://www.netlib.org/lapack/)

**Description:** The Linear Algebra PACKage (LAPACK) is written in FORTRAN 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.

**Platforms:** Version 3.5.0, which is the current version, is installed on all our platforms.

4.8 PETSc

**URL:** [http://www.mcs.anl.gov/petsc/](http://www.mcs.anl.gov/petsc/)

**Description:** This suite contains data structures and routines for a scalable and parallel solution of scientific applications modeled by partial differential applications.

**Platforms:** Version 3.5.3 with MPI and OpenCL support is installed at the Mont-Blanc prototype. It was also configured to use our FFTW and Boost installation.

5 Developer tools

Please refer to D5.9 for more details.

5.1 Perf

**URL:** [https://perf.wiki.kernel.org/index.php/Main](https://perf.wiki.kernel.org/index.php/Main)

**Description:** The Linux perf command is a tool to instrument CPU performance counters, tracepoints, kprobes and uprobes (dynamic tracing).

**Platforms:** Is it currently installed on all the platforms where the kernel sources are available: CARMA, Odroid XU, Odroid XU-3, Jetson and the Mont-Blanc prototype.

5.2 Extrae

**URL:** [http://www.bsc.es/computer-sciences/extrae](http://www.bsc.es/computer-sciences/extrae)

**Description:** Extrae is a package devoted to generating Paraver trace-files for post execution analysis. It uses different interposition mechanisms to inject probes into the target application.

**Platforms:** Latest version of this package (3.1.0) is currently installed on all of our platforms, with support for MPI, OpenCL, OmpSs and/or CUDA depending on the available runtimes on each cluster.

5.3 Scalasca

**URL:** [http://www.scalasca.org/](http://www.scalasca.org/)

**Description:** Scalasca is a software tool that supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior.
Platforms: Version 2.2.1 is installed at the Mont-Blanc prototype.

6 Runtimes

6.1 OpenCL

URL: https://www.khronos.org/opencl/

Description: Open Computing Language (OpenCL) is a framework for executing across heterogeneous platforms consisting of CPUs or GPUs.

Platforms: OpenCL is only installed at the prototypes that are fully IEEE 754 compliant in both CPU and GPU. This means only Arndale and the Mont-Blanc prototype. Version 1.1.0 of OpenCL is installed on both.

6.2 CUDA

URL: https://developer.nvidia.com/about-cuda

Description: CUDA is a framework for executing across single or multiple GPUs. It provides also the needed compiler for generating binaries from the source code.

Platforms: Only Nvidia platforms support CUDA, so only CARMA, Pedraforca and Jetson support it. Different versions are installed on each of them, starting from 5.0, which does not include an ARM-native compiler since it was first included at version 5.5, to 6.0, which is currently installed only on Jetson.

6.3 MPI

The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum, which has over 40 participating organizations [mpi15]. Several implementations are available, in our case we provide only two of them at our platforms.

6.3.1 OpenMPI

URL: http://www.open-mpi.org/

Description: OpenMPI is an open-source MPI implementation developed and maintained by a consortium of academic, research and industry partners. It implements MPI-1, MPI-2 and MPI-3 standards.

Platforms: Version 1.8.3 is installed from sources on all of our platforms.

OpenMPI performance study: For OpenMPI, the Intel MPI Benchmarks (IMB) [imb13] have been executed on the Mont-Blanc prototype in order to study network performance varying different OpenMPI and TCP parameters. As network topology could in principle also affect performance we executed the benchmark by using three different network topologies:

- Nodes located in the same blade: in this case nodes are connected through 1 on-board switch.
Nodes located in different blades but within the same rack: in this case two level of switches are involved, the ones within the blade (previous case) plus the ones among blades (10 GbE).

- Nodes located in different blades and different racks: in this last case a third level of switching is involved, as we have to cross rack boundaries.

Apart from the network topology, the other monitored parameters have been:

- **RTO** indicates the TCP Retransmission Time Out in milliseconds.
- **PPN** indicates the number of MPI processes mapped to each node.

Figures 2a, 2b and 2c show the result obtained by running the Alltoall benchmark from IMB. Figures 2d, 2e and 2f show results for the Exchange benchmark. The first one executes the MPI_Alltoall function on all the MPI processes while the Exchange benchmark from IMB performs a bidirectional ring communication. Each plot contains several lines, each one shows a different TCP and process binding configuration.

General observations are that:

i) we observe low bandwidth (≈ 30% less than theoretical 1 GbE peak),

ii) we observe high latency (not plotted here),

iii) we also detect an overall high variability among executions,

iv) we do not see strong dependency on the physical position/distance of the nodes.

Looking at the TCP parameters, it seems that lowering the RTO has generally negligible effect on all the topology configurations for large message size, even if in some cases this lowers the variability of the results: it is interesting to note for example that lowering the RTO results reduces the fluctuations around 16 and 32 KB in the Alltoall benchmarks.

At a second stage, we modified the buffer size that OpenMPI uses for sending and receiving data. Figures 3a and 3b show the result by using an RTO of 200 and 5 milliseconds respectively. At this time all the nodes were located at the same blade since the previous test showed that the topology variation does not make any significant. We can see that large message sizes, 64 and 128 KB seem to work slightly better, while worst option in both cases of RTO is represented by the buffer size of 16 KB.

Even if based on synthetic benchmarks, these preliminary tests showed that careful tuning of environment parameters can have beneficial effects on the overall performance of the system and can alleviate the technological limitation hit by the prototype, due to the use of commodity, embedded and low-cost technology. However, we finally abandoned OpenMPI due to it’s high variability and bad scalability and moved to using MPICH.

### 6.3.2 MPICH

**URL:** [https://www.mpich.org/](https://www.mpich.org/)

**Description:** MPICH is a high performance and widely portable implementation of the MPI standard. It too implements MPI-1, MPI-2 and MPI-3 standards.

**Platforms:** Versions 3.1.3 and 3.1.4 are installed on all our platforms.

### 6.4 Nanos++

**URL:** [http://pm.bsc.es/nanox](http://pm.bsc.es/nanox)

**Trac:** [https://pm.bsc.es/projects/nanox](https://pm.bsc.es/projects/nanox)
(a) Alltoall benchmark with all the nodes on the same blade

(b) Alltoall benchmark with nodes on different blades but same rack

(c) Alltoall benchmark with nodes on different blades and rack

(d) Exchange benchmark with all the nodes on the same blade

(e) Exchange benchmark with nodes on different blades but same rack

(f) Exchange benchmark with nodes on different blades and rack

Figure 2: Alltoall and Exchange benchmarks with different RTO and MPI process mapping.
(a) Exchange benchmark with RTO equal to 200ms and one MPI process mapped to each node. (b) Exchange benchmark with RTO equal to 5ms and one MPI process mapped to each node.

Figure 3: Exchange benchmark with different buffer sizes.

**Description:** Nanos++ is the OmpSs runtime developed at BSC, designed to support the OmpSs programming model. It allows experimentation with new program transformations that can be implemented using the Mercurium compiler (see section 3.2). As such it is designed to be extensible by means of plugins. Currently, runtime plugins can be added (and selected for each execution) for:

- Task scheduling policy
- Thread barrier
- Device support
- Instrumentation formats
- Dependences approach
- Throttling policies

In this project, we have worked on the device support aspect, adding the plugin supporting OpenCL tasks. The execution of the OpenCL code and the data transfers are triggered automatically from Nanos++ based on the *target* directives used in the source code of the applications.

Nanos++ was continually updated on the prototype. Currently we have installed the latest versions: Nanos++ 15.04 stable version (0.7) and development version (0.9).

### 6.4.1 OmpSs

OmpSs is the effort to integrate features from the StarSs programming model developed by BSC, and OpenMP into a single programming model. In particular, our objective is to extend OpenMP with new directives to support asynchronous parallelism and heterogeneity (devices like GPUs). However, it can also be understood as new directives extending other accelerator based APIs like CUDA or OpenCL. Our OmpSs environment is built on top of our Mercurium compiler and Nanos++ runtime system.

### 6.4.2 OmpSs+OpenCL

OmpSs+OpenCL is an OmpSs plugin to use OpenCL-based heterogeneous devices on the platforms. OmpSs manages all operations related with the devices (context, buffers, data transfers,
memory coherency,...) allowing the application developers to focus on the development of kernels and increasing their productivity. We have improved the support for OpenCL in architectures with a reduced number of cores, and in particular in the nodes of the Mont-Blanc prototype. This improvement consists of the blocking of the OpenCL thread manager when idle, to allow another thread to execute SMP tasks. This solution avoids contention for the cores by the application threads, thus improving the performance achieved in applications using two SMP threads and the GPU.

We have also adapted the Nanos++ configuration and compilation mechanisms to better detect the ARM OpenCL library for the Mali GPU.

6.4.3 OmpSs_CUDA

OmpSs_CUDA is an OmpSs plugin which works similarly to OpenCL plugin. OmpSs is able to manage all operations related with the NVIDIA GPUs: devices, buffers, data transfers, etc. In addition, OmpSs was designed to let the user work with both OpenCL and CUDA devices with few changes in their code.

7 Hardware support

As hardware support depends on the platforms, we report here the effort done by the project for having added-value hardware support on the Mont-Blanc prototype.

7.1 Kernel 3.11.0

The 3.11.0 Linux kernel used in the Mont-Blanc prototype is based on the 3.11.0 kernel available for the Arndale board [Arn12] from the Samsung Landing team at Linaro [Ker14].

The kernel was configured to allow for differences between the Arndale board and the Mont-Blanc SDB card:

- Removing the dependence on MMC card detection (no MMC slot is provided on the SDB)
- Changing the default low-level UART
- Adding support for the USB3-to-Gigabit Ethernet chip on the SDB as a loadable kernel module
- Changes to the memory map. The SDB has more memory (4GB versus 2GB) than the Arndale board for which the kernel was originally developed

A number of patches are required for the correct integration of support for running OpenCL on the Mali-T604 GPU in the Samsung Exynos 5250. These patches were provided under a Limited Use License agreement.

The driver for the AX88179 USB3-to-Gigabit Ethernet chip was downloaded from the ASIX website and built as a kernel module against the kernel source.
7.2 DVFS

Dynamic Voltage Frequency Scaling (DVFS) has been implemented with several modifications in both the kernel and the hardware. The DVFS features allow us to change the frequency of the processor as well as the frequency of the GPU. If these frequencies are reduced, then it is possible to decrease the SoC voltages to reduce the power consumption. In order to do so, the Exynos SoC requires some programming action on the PMIC (Power Management Integrated Circuit) which controls the voltages of the SoC.

The PMIC is on the EMB card and can be accessed either by the BMC (Board Management Controller) or by the SoCs. In order to allow each node modify its current frequency, and therefore its voltage, the FPGA on the EMB was modified to allow each SoC to access the PMIC once the board has been initialized since, during the initialization process, it is the BMC which must access the PMIC.

At kernel level, we have made other changes in order to ensure that the frequency and the voltage is set every time a request is made. The issue here is that for this particular Linux kernel version, the CPU frequency driver modifies directly the registers in the SoC with no care of the sysfs bookkeeping done by the OS. This translates into CPU frequency changes not being reflected at the corresponding sysfs entries. This could result in some cases, a change of frequency was enforced and applied to the CPU, but not accordingly updated in the OS. The OS state reflected the previous CPU frequency, causing a mismatch and making it possible that a subsequent change of frequency request via the OS could be ignored.

In order to resolve this issue we modified the kernel sources so every time a new CPU frequency is requested (via the OS), the registers are always updated. This way we added some overhead to each change, but at least we ensure that, for example, after decreasing the frequency due to high temperature, it will return to its highest value after the thermal issue is solved.

This last issue has been solved on newer version of the Linux Kernel by using the CCF (Common Clock Framework) API to set the clock rates.

7.3 Power Monitor

In deliverable D5.8 [Del13b] we provided a detailed description of the power sampling process of the Mont-Blanc prototype and of the main features of the tool implemented for monitoring its power consumption. The software architecture of the tool is illustrated in Figure 4 and is briefly summarized as follows.

The tool consists of three major components: a Pusher responsible for polling out-of-band power consumption data from the Board Management Controller (BMC) of the monitored device according to a specific protocol (IPMI, BACNet, SNMP, etc.). Once retrieved, the power data is sent to the Collect Agent, which, as the name implies, collects all monitored information from one or multiple instances of the same or different Pushers, potentially spread amongst different chassis, racks or even data centers. The Collect Agent is finally responsible for transmitting the collected data via MQTT messages [mqt14, Del13b] to the Apache Cassandra key-value store [cas14, Del13b], where all monitored information is finally stored. On top of standard monitoring features and specific customizations due to the “first-of-a-kind” nature of the Mont-Blanc prototype, the software design of the monitoring tool also allows for minimizing transport messaging overhead by employing the MQTT protocol while extreme scalability and configurable redundancy are offered with the employment of the Cassandra key-value store. For a more detailed explanation of the main features of the tool, the reader may refer to deliverable D5.8 [Del13b].

In this section of the document, we illustrate how we expanded the conceived monitoring
tool not only to get power consumption information, but also to acquire other important sensor data of the system such as Samsung Daughter Board (SDB) and Ethernet Mother Board (EMB) temperatures, fan speeds, Power Supply Unit (PSU) voltages and so on. Particularly, in Section 7.3.1 we describe the installation process of the tool, while in Section 7.3.2 we explain its usage, providing monitored data query examples. Finally, in Section 7.3.3 we verify the tool collecting sensor readings and presenting first findings on the power profile of the Mont-Blanc prototype.

7.3.1 Components of the tool and their configuration

The source code of the monitoring tool is written in C++ and consists of many software components. In addition to the Apache Cassandra key-value store, the software tools that are of interest for the Mont-Blanc prototype are the following:

- **CollectAgent**, used to collect the monitored data from all Pushers and push it to the Cassandra key-value store.

- **MontBlancPusher**, used to pull exclusively power consumption data at SDB level.

- **IPMIPusher**, used to pull all other sensor data available through the IPMI [ipm14] interface like EMB temperature and EMB power consumption, fan speeds, PSU voltages and electrical currents and so on.
• **FilePusher**, used to push temperature readings at SDB level. Each SDB runs an instance of this Pusher.

To run the tool it is sufficient to first run Cassandra by typing:

```
>> cd DCDBDEPLOYPATH/cassandra/
>> ./bin/cassandra
```

Once Cassandra is running, we may start the Collect Agent by typing in the command line:

```
>> cd DCDBDEPLOYPATH/bin/
>> ./collectagent
```

The first run of the Collect Agent will automatically create two Cassandra keyspaces [cas14, Del13b], respectively the dcdb and the dcdb\_config keyspace. The first one contains the Cassandra column family **sensordata**, which organizes all sensor information through their IDs, values and relative timestamps. We remind the reader that sensor IDs are structured as 128-bit MQTT message topics in the Mont-Blanc prototype and we invite him to refer to Section 3.3.2 of Deliverable 5.8 [Del13b] for a detailed explanation of the MQTT message layout. Each Sensor ID is then published in the column family **published\_sensors** of the dcdb\_config keyspace with a corresponding, unique human-readable **sensor name**. All published sensor names are based on the concatenation \(<\text{hostname}>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\>\&num
Table 3: List of sensor suffixes and MQTT identifiers at EMB level.

<table>
<thead>
<tr>
<th>SUFFIX</th>
<th>MQTT IDENTIFIER</th>
<th>SENSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWR</td>
<td>0000</td>
<td>EMB power consumption (W)</td>
</tr>
<tr>
<td>TMP</td>
<td>0001</td>
<td>EMB temperature (C)</td>
</tr>
<tr>
<td>PQII_TMP</td>
<td>0002</td>
<td>PQII temperature (C)</td>
</tr>
<tr>
<td>FAN1A_BMC</td>
<td>0003</td>
<td>Fan 1A speed (rpm)</td>
</tr>
<tr>
<td>FAN1B_BMC</td>
<td>0004</td>
<td>Fan 1B speed (rpm)</td>
</tr>
<tr>
<td>FAN2A_BMC</td>
<td>0005</td>
<td>Fan 2A speed (rpm)</td>
</tr>
<tr>
<td>FAN2B_BMC</td>
<td>0006</td>
<td>Fan 2B speed (rpm)</td>
</tr>
<tr>
<td>FAN3A_BMC</td>
<td>0007</td>
<td>Fan 3A speed (rpm)</td>
</tr>
<tr>
<td>FAN3B_BMC</td>
<td>0008</td>
<td>Fan 3B speed (rpm)</td>
</tr>
<tr>
<td>FAN4A_BMC</td>
<td>0009</td>
<td>Fan 4A speed (rpm)</td>
</tr>
<tr>
<td>FAN5B_BMC</td>
<td>000A</td>
<td>Fan 4B speed (rpm)</td>
</tr>
</tbody>
</table>

Mont-Blanc Pusher

The Mont-Blanc Pusher retrieves power consumption data at SDB level and sends it to the Collect Agent via MQTT. To run the Mont-Blanc Pusher simply use

montblancpusher [-v] [-s] [-h <host>] [-l] [-D] <config file name>

where:

- -v enables verbose mode, printing all information related to SDB power consumption with relative timestamp.
- -s enables statistics mode, printing the rate of MQTT messages sent to the Cassandra DB.
- -h <host> is used to indicate a specific host to connect to for retrieving the sensor data. If not used, the tool considers the hostname in the provided config file or is localhost.
- -l enables RCMP+ for improved authentication and data integrity checks, as well as encryption and the ability to carry multiple types of payloads (uses IPMIv2 instead of IPMIv1.5).
- -D starts the tool in daemon mode.

The provided config file contains all the information related to the hosts (like hostnames, MAC addresses, IPMI ports) and sensors (like type of sensors, readout frequencies, sensor IDs) and must be written in an INFO layout. The MontBlancPusher source code provides a template for the related config file.

IPMI Pusher

The IPMI Pusher retrieves all IPMI sensor data at EMB and chassis level and sends it to
Table 4: List of sensor suffixes and MQTT identifiers at chassis level.

<table>
<thead>
<tr>
<th>SUFFIX</th>
<th>MQTT IDENTIFIER</th>
<th>SENSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAWERPOWER</td>
<td>0000</td>
<td>Total chassis power consumption (W)</td>
</tr>
<tr>
<td>CMBTEMP</td>
<td>0001</td>
<td>Chassis Management Board temperature (C)</td>
</tr>
<tr>
<td>FPTEMP</td>
<td>0002</td>
<td>FP Temperature (C)</td>
</tr>
<tr>
<td>FAN1A_CMC</td>
<td>0003</td>
<td>Fan 1A speed (rpm)</td>
</tr>
<tr>
<td>FAN1B_CMC</td>
<td>0004</td>
<td>Fan 1B speed (rpm)</td>
</tr>
<tr>
<td>FAN2A_CMC</td>
<td>0005</td>
<td>Fan 2A speed (rpm)</td>
</tr>
<tr>
<td>FAN2B_CMC</td>
<td>0006</td>
<td>Fan 2B speed (rpm)</td>
</tr>
<tr>
<td>PSU1_VIN</td>
<td>0007</td>
<td>Power Supply 1 Voltage In (V)</td>
</tr>
<tr>
<td>PSU2_VIN</td>
<td>0008</td>
<td>Power Supply 2 Voltage In (V)</td>
</tr>
<tr>
<td>PSU3_VIN</td>
<td>0009</td>
<td>Power Supply 3 Voltage In (V)</td>
</tr>
<tr>
<td>PSU4_VIN</td>
<td>000A</td>
<td>Power Supply 4 Voltage In (V)</td>
</tr>
<tr>
<td>PSU1_IIN</td>
<td>000B</td>
<td>Power Supply 1 Current In (Amp)</td>
</tr>
<tr>
<td>PSU2_IIN</td>
<td>000C</td>
<td>Power Supply 2 Current In (Amp)</td>
</tr>
<tr>
<td>PSU3_IIN</td>
<td>000D</td>
<td>Power Supply 3 Current In (Amp)</td>
</tr>
<tr>
<td>PSU4_IIN</td>
<td>000E</td>
<td>Power Supply 4 Current In (Amp)</td>
</tr>
<tr>
<td>PSU1_12VOUT</td>
<td>000F</td>
<td>12 Volt Power Supply 1 (V)</td>
</tr>
<tr>
<td>PSU2_12VOUT</td>
<td>0010</td>
<td>12 Volt Power Supply 2 (V)</td>
</tr>
<tr>
<td>PSU3_12VOUT</td>
<td>0011</td>
<td>12 Volt Power Supply 3 (V)</td>
</tr>
<tr>
<td>PSU4_12VOUT</td>
<td>0012</td>
<td>12 Volt Power Supply 4 (V)</td>
</tr>
<tr>
<td>PSU1_3v3VOUT</td>
<td>0013</td>
<td>3.3 Volt Power Supply 1 (V)</td>
</tr>
<tr>
<td>PSU2_3v3VOUT</td>
<td>0014</td>
<td>3.3 Volt Power Supply 2 (V)</td>
</tr>
<tr>
<td>PSU3_3v3VOUT</td>
<td>0015</td>
<td>3.3 Volt Power Supply 3 (V)</td>
</tr>
<tr>
<td>PSU4_3v3VOUT</td>
<td>0016</td>
<td>3.3 Volt Power Supply 4 (V)</td>
</tr>
<tr>
<td>PSU1_12IOUT</td>
<td>0017</td>
<td>12 Volt Power Supply 1 Current (Amp)</td>
</tr>
<tr>
<td>PSU2_12IOUT</td>
<td>0018</td>
<td>12 Volt Power Supply 2 Current (Amp)</td>
</tr>
<tr>
<td>PSU3_12IOUT</td>
<td>0019</td>
<td>12 Volt Power Supply 3 Current (Amp)</td>
</tr>
<tr>
<td>PSU4_12IOUT</td>
<td>001A</td>
<td>12 Volt Power Supply 4 Current (Amp)</td>
</tr>
<tr>
<td>PSU1_3v3IOUT</td>
<td>001B</td>
<td>3.3 Power Supply 1 Current (Amp)</td>
</tr>
<tr>
<td>PSU2_3v3IOUT</td>
<td>001C</td>
<td>3.3 Power Supply 2 Current (Amp)</td>
</tr>
<tr>
<td>PSU3_3v3IOUT</td>
<td>001D</td>
<td>3.3 Power Supply 3 Current (Amp)</td>
</tr>
<tr>
<td>PSU4_3v3IOUT</td>
<td>001E</td>
<td>3.3 Power Supply 4 Current (Amp)</td>
</tr>
<tr>
<td>PSU1_PWRIN</td>
<td>001F</td>
<td>Power Supply 1 Power In (W)</td>
</tr>
<tr>
<td>PSU2_PWRIN</td>
<td>0020</td>
<td>Power Supply 2 Power In (W)</td>
</tr>
<tr>
<td>PSU3_PWRIN</td>
<td>0021</td>
<td>Power Supply 3 Power In (W)</td>
</tr>
<tr>
<td>PSU4_PWRIN</td>
<td>0022</td>
<td>Power Supply 4 Power In (W)</td>
</tr>
</tbody>
</table>

the Collect Agent via MQTT. To run the IPMI Pusher simply type

```
ipmipusher [-v] [-s] [-h <host>] [-l] [-D] <config file name>
```

where all options have the exact same function of the ones of the Mont-Blanc Pusher. Similarly, the config file of the IPMI Pusher contains all the information related to the hosts and sensors and must be written in an INI layout.
**File Pusher**

The File Pusher reads SDB temperatures and, like the other Pushers, send it to the Collect Agent via MQTT. The difference with the other Pushers is that the File Pusher follows a “more genuine” push behavior and it must be launched from the device that is being monitored (in our case the SDB). For this reason, the folder containing the binary must be visible to all SDBs in the Mont-Blanc prototype. To run the File Pusher simply type

```
```

where:

- `-v` enables verbose mode, printing all information related to SDB temperature with relative timestamp.
- `-s` enables statistics mode, printing the rate of MQTT messages sent to the Cassandra DB.
- `-b` enables reading of binary files. By default, the File Pusher reads text files.
- `-a`, automatically generates the MQTT topic of the SDB where an instance of the File Pusher is running.
- `-I`, enables the `inotify` mode. With this option, the File Pusher pushes readings if and only if the monitored file is modified. The implementation of this feature is done with the employment of the Linux `inotify` set of functions. This allows for a less invasive monitoring feature, lowering the overhead on the compute node by pushing data only when necessary (that is, when it changes). Unfortunately, the file containing SDB temperature information is located in `sysfs`, hence does not change unless it is open, making this feature de facto unusable for the Mont-Blanc prototype.
- `-D` starts the tool in daemon mode.
- `-i <interval>`, specifies the readout frequency in milliseconds. The default values is 1000 milliseconds.
- `-h <host>` indicates a specific host to connect to for retrieving the sensor data. If not used, the default host is localhost.
- `-t <topic>` is used to specify an MQTT topic to associate with the monitored file. If not used, the default topic will be the file absolute path. The name of the file to be monitored is `<filename>`.

Due to its simplicity and to the fact that it is used only to monitor SDB temperatures, the File Pusher does not require a config file and all configuration parameters can be set as command line options as shown above.
7.3.2 Usage

The acquisition of monitored sensor data from the Cassandra key-value store is provided by the `dcdbquery` tool that can be found in the folder DCDBDEPLOYPATH/bin. The usage of the tool is as follows:

```
dcdbquery [-r] [-l] [-h <hostname>] <Sensor 1> [<Sensor 2>...] <Start> <End>
```

where:

- `<hostname>` is the name of the database server. For the Mont-Blanc case it is gw.mont.blanc.
- `<Sensor n>` represent the name of one or more sensors.
- `<Start>` is the start of the time series.
- `<End>` is the end of the time series.

`<Start>` and `<End>` times can be supplied in two formats:

- Human readable: In this case the date should be supplied as ‘yyyy-mm-dd hh:mm:ss’ (with quotes), e.g. ‘2015-04-16 15:38:29’.
- Unix epoch: corresponding to the output of ‘date +%s’

By default, times are interpreted to be in UTC. Using the -l option allows to switch interpretation of `<Start>` and `<End>` as well as the generated output to the local timezone of the user. Furthermore, if the -r option is specified, the generated output contains the raw internal timestamps (nanoseconds since UNIX Epoch) instead of the human readable ISO format.

Using the `dcdbquery` tool for monitoring sensor data during the execution of an application requires the knowledge of the nodes where the job is submitted and its execution time. This information is easily accessible in SLURM, for example by augmenting the job submission script with the following lines:

```bash
# Print the node list
echo "Running on nodes: " scontrol show hostname $SLURM_JOB_NODELIST"

# Print the start time
echo -n "Execution starts at: "
date +%s

# Run my application
srun my_application

# Print the end time echo -n "Execution stops at: "
date +%s
```

For example, let us assume we want to acquire the power consumption of SDB mb-11-10 and the temperature of SDB mb-11-11. Then, the query will be

```
dcdbquery -h gw.mont.blanc -r mb-11-10-PWR mb-11-11-TMP 1429178521 1429178527
```
where we chose <START> to be 1429178521 and <STOP> to be 1429178527. An example of the output of the query is as follows

Sensor, Time, Value
mb-11-10-PWR, 1429178521580000000, 8458  
mb-11-10-PWR, 1429178522600000000, 9462  
mb-11-10-PWR, 1429178523620000000, 8994  
mb-11-10-PWR, 1429178525650000000, 7475  
mb-11-10-PWR, 1429178526670000000, 7472  
mb-11-11-TMP, 1429178521580000000, 5700  
mb-11-11-TMP, 1429178522600000000, 3600  
mb-11-11-TMP, 1429178523620000000, 4200  
mb-11-11-TMP, 1429178525650000000, 3400  
mb-11-11-TMP, 1429178526670000000, 4000

The output file is formatted in comma-separated-values, allowing for easy data processing in most spreadsheets and database management systems. The Energy-to-Solution (EtS) of a job is also more easily accessible in the Consumed Energy field of Slurm through a specific implementation of its ext_sensors plugin (as explained in deliverable [Del13a]). The integration of the tool with Slurm is briefly discussed in Section 9.6.1.

7.3.3 Verification of the monitoring tool in the Mont-Blanc prototype

Before the power measurement tools were made available to the application developers of the Mont-Blanc project, thorough tests were conducted to verify the proper collection of data. For this, a small test tool was written that performs 5 cycles of CPU-intense calculations for 30 seconds, followed by 30 seconds of idle. If the data acquisition and storage works well, the 5 application cycles are clearly visible in the applications power trace (see Figure 5a) obtained from the database. It should be noted that the CPU-intense phases run fully out of the CPUs L1 cache meaning there is no stress on the memory or the GPU resulting in lower power consumption than other benchmarks.

Despite its simplicity, the tool is helpful in finding the most common sources of errors in the power measurement setup, such as: inaccuracies in the time synchronization between the database and the board management controllers, too high or too little indicated power drawn due to broken components or sensors as well as other irregularities. To qualify the Mont-Blanc prototypes power measurement, the test application was run on all nodes and the resulting graphs have undergone visual inspection.

This analysis revealed a couple of blades in which for some of the nodes, the power and performance was not stable across iterations (as shown in Figure 5b). Further experiments showed that the drop in power consumption always came along with a drop in compute performance at which it became clear that this rather exhibits a performance problem instead of a problem in the power measurement setup. Since other users of the machine also experienced issues related to performance variations, the problem was subsequently narrowed down to a cooling issue which could be mitigated by moving towards a more aggressive fan speed policy and disabling of peculiar nodes. After these changes, subsequent runs of the test application longer exhibited this power and performance variation (see Section 10.1 for more details about thermal tests).
(a) Node behaves as expected  
(b) Node with power and performance issues.

Figure 5: Power trace of the power cycle test application.

To ensure that the prototype was in good shape, histograms of the temperature and power distribution were generated from the data obtained before and after stabilizing the system. Figure 6 shows that the new fan speed policy was successful in lowering SoC temperatures (with the aim to avoid any temperature related throttling).

(a) Initial system  
(b) Reworked system

Figure 6: Temperature histogram of the prototype’s SoC temperatures

Figure 7 shows the power distribution of SDBs under idle and load. It illustrates that the stabilization also reduced the number of outliers in the power distribution allowing for better reproducible application benchmark results. Power variations among different nodes under the same work load is a known behaviour and the extent of it can be explained by the absence of CPU binning in the mobile marked as opposed to the desktop and server CPU markets.

Finally, combining temperature and power data, we can analyze in Figure 8 that - contrary to what is typically observed on x86 based systems - no significant impact of operating temperature on power consumption is being observed in the Mont-Blanc prototype. Yet, this effect mainly depends on the semiconductor manufacturing process. It is therefore not possible to make any claims about the temperature-power dependency of ARM based SoCs in general.
### Figure 7: Power histogram of SDBs in case of idle and load

(a) Initial system

![Idle/Load Power Histogram](image1)

(b) Reworked system

![Idle/Load Power Histogram](image2)

### Figure 8: Analysis of the temperature-power dependency in the Mont-Blanc prototype

(a) Initial system

![Power vs. Temperature](image3)

(b) Reworked system

![Power vs. Temperature](image4)

### 8 Storage

Please refer to D5.2 and D5.10 for further information concerning Mont-Blanc experience related to distributed file system on ARM-based platforms.

#### 8.1 NFS


**Description:** Network File System (NFS) is a distributed file system protocol which allows the user on a client computer to access files over a network.

**Platforms:** We use NFS to share the home user folders as well as the applications described across all the clusters except for the Mont-Blanc prototype, which uses Lustre file system.

#### 8.2 Lustre

**URL:** [http://lustre.org/](http://lustre.org/)
Description: The Lustre file system is an open-source, parallel file system that supports many requirements of HPC environments.

Platforms: We use Lustre file system on the Mont-Blanc prototype to share a partition among all the compute nodes, containing application binaries as well as the home folders of each user.

9 Cluster management

9.1 OpenLDAP

URL: http://www.openldap.org/

Description: OpenLDAP is an open-source implementation of the Lightweight Directory Access Protocol (LDAP), providing an LDAP server which is used for authentication purposes.

Platforms: This software has been installed from repositories on all our servers. Regarding the ARM-based platforms, an LDAP authentication client is installed and configured among all of them.

Configuration: The LDAP database is indexed and cached to improve the latency of each query.

9.2 NTP

URL: http://www.pool.ntp.org/en/

Description: Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks.

Platforms: All our platforms synchronize their clocks with the NTP server implemented at our servers.

9.3 Puppet

URL: https://puppetlabs.com/puppet/puppet-open-source

Description: Puppet is a configuration management solution that allows to define the state of an IT infrastructure, enforcing automatically this state. It uses a set of configuration files to describe the modules and the manifests that Puppet uses to specify the states. In those files it is where it is specified the state of the machine using the Puppet language.

Usage: In the Mont-Blanc project Puppet is used to automate the deployment of the software stack to the computational nodes and to maintain their configurations. It is useful not only because it does not require configuration of each node separately, but also because the same Puppet configuration can be used for maintaining the different platforms of the project due to the flexibility of the Puppet language.

All Puppet manifests and modules files as well as more information about the functioning of Puppet are available in the following git repository: https://github.com/Mont-Blanc-Project/puppet.git
9.4 Nagios

URL: https://www.nagios.org/

Description: Nagios is a monitoring tool designed to monitor the status and integrity of a whole IT infrastructure.

Platforms: We installed version 4.0.8 from sources on our servers. No clients are installed on ARM-based node because the status is checked by external x86 server.

Configuration: Nagios server connects to the platforms via SNMP (Simple Network Management Protocol) or SSH (Secure Shell) to query each node in order to get status data. Not all the nodes of each platform are checked by Nagios since it will not scale due to the huge number of nodes we have as well the fact that each check translates to an active connection and the consequently write to disk. Considering this scalability issue, we decided that only the nodes of each platform designated as login nodes are checked by Nagios. Performance data from these nodes is stored in a RRD database to generate status and performance plots. For the rest of the nodes we use Ganglia, described at section 9.5. Ganglia allows to create a hierarchy of nodes which forward messages from others, so a smaller number of connections is made against the server and fewer performance data messages are written to disk since the data is aggregated at the forwarder nodes.

9.5 Ganglia

URL: http://ganglia.sourceforge.net/

Description: Ganglia is another monitoring system but with a different approach to Nagios. In this case the idea is to monitor performance metrics of the platforms as, in example, CPU load, memory usage, etc.

Platforms: The version installed on our servers and platforms is the 3.7.1.

Configuration: Ganglia is able to scale by using intermediate nodes to transfer data from other nodes. This way, we built a tree in our platforms so each node can actually sent its performance data with two hops at most. This decreases exponentially the number of connections established with the server.

9.6 SLURM

URL: http://slurm.schedmd.com/

Description: SLURM is an open-source workload manager designed for any size of Linux cluster. It supports heterogeneous architectures.

Platforms: Version 14.11.3 is installed in our prototypes from sources since we need to have control of all the plugins installed due to the fact that we are not using state of the art architectures.

Configuration: By using different plugins provided by SLURM we are able to specify the network topology so that a network-aware selection of the compute nodes for each job is possible during the scheduling of the jobs. Apart from this, more configuration has been done to make the several queues on our prototypes more fair by using Quality of Service policies as well as setting job priorities considering the age of the job, the number of nodes
and predicted execution time. Finally, performance data of each job as memory and CPU usage, execution time, power consumption and several others are stored in a database so the user can consult it afterwards to know how the application is using the available resources.

9.6.1 Power monitor plugin

The Slurm resource manager (www.schedmd.com) provides full support for energy accounting since version 2.6. Using the energy accounting features, the Energy-to-Solution (EtS) of all applications running on a cluster is stored along the standard accounting fields (i.e. user, node list, runtime,...) in the accounting database. This provides to both users and system administrators an easy-to-use interface to gather statistics about the energy-usage of their applications without having to analyze the detailed power traces obtained from the dcbquery tool (see Section 7.3 for details regarding the usage of dcbquery on how to retrieve sensor data in the Mont-Blanc prototype). In D5.5 [Del13a], we have described the APIs that Slurm provides for integrating new systems with the existing energy accounting infrastructure: an in-band interface called acct.gather.energy and an out-of-band interface called ext.sensors.

The power monitoring solution in Mont-Blanc works out-of-band through the board management controllers. Thus, the ext.sensors interface was the first choice to integrate the Mont-Blanc power monitoring solution with Slurm. The Slurm plugin is very simplistic as it only provides a hook function that is called upon completion of each job. During this hook function, it queries the monitoring backend library for the energy consumed by each node involved in the job during the job’s timeframe, sums up the results and writes the information into the job’s accounting data structure. To enable compatibility between the Slurm plugin and the backend library, the C++ library had to be extended with a C API so that the Slurm plugin can link against it.

Deriving the energy from the power trace is done within the backend library by implementing a simple trapezoidal integration scheme. According to the Slurm conventions, the consumed energy is stored in the accounting database as Joules. Users can query the accounting information using the standard Slurm sacct command.

10 Hardware/Software test and configuration

10.1 Thermal experiments

After bringing up the Mont-Blanc prototype and starting to execute applications as well as testing the stability of the cluster, we noticed variability across the executions. After a deeper study we saw some instability appearing on some of the nodes.

This instability was appeared to be a random effect, although though it was more frequent in case of large simulations with MPI applications where a huge number of nodes were used at the same time. We have been forced to study this misbehaviour at both node and blade level.

Figure 9 shows an example of the data that we systematically measured on each blade and node of the Mont-Blanc prototype (data for one blade). For this study we executed the High Performance Linpack benchmark (HPL) [lin08] separately on all the nodes of the blade, collecting the following data on each one (in order as displayed in Figure 9)

- SoC temperature at node level
- CPU frequency at node level
• Power consumption at node level
• Blade motherboard temperature
• Blade switch temperature
• Power consumption of the blade
• Blade fans speed

It is important to notice how all plots shown in Figure 9 are related. In the first line we can see the behaviour of the temperature of all nodes within the blade. The saw behaviour of temperature of the nodes is strongly related to last line of the Figure 9, where the revolutions per minute (RPM) of the fans are shown: the correlation is clearly visible between the fan RPM, the temperature of the board and the overall power consumption of the blade. In particular, when the temperature of the motherboard passes a threshold, the fans are activated at a higher RPM and therefore the overall power consumption increases. After some time of stronger air flow, the overall temperature becomes lower, resulting in the fans also lowering their frequency and so power consumption decreases etc. This is in general a normal behaviour, but in our case some nodes were suffering from thermal throttling of CPU frequency: looking at the frequency box in Figure 9 it is clearly visible that nodes 11 and 12 after a certain point of the run start running at half frequency. These two nodes are in fact suffering a behaviour of the hardware that is automatically limiting the frequency in case of reaching of a temperature threshold (85 degrees C) in order to avoid damage to the hardware.

As we can see at the frequency plots, the frequency of these nodes is decreased to avoid damaging the CPU, but is not going back to its maximum after the temperature issue is solved. This behaviour is also reflected at the power consumption of those nodes: the orange line (node 12) and the blue line (node 11) are clearly indicating the use of half power starting from about one third of the run.

So we conclude the solutions were the following:
• Increase the speed of the fans with a firmware update at blade level
• Power off the nodes presenting the thermal throttling
  – Node number 12 is more prompt to show the issue due to design
• Modify the kernel to ensure the frequency is restored once the thermal issue is solved

All of these solutions have been implemented, but only the first and the second one were deployed in order to have as small interference as possible with the users (updating the kernel implies long time outage).

After the deployment of the fixes we executed again the same sanity checks on all the blades. At Figure 10 we can see the results.

You can see that fan speed is increased with obvious benefits for the temperature of the nodes. Of course, as the fans are working on a higher speed, the power consumption of the blades is increased between 7 and 20% depending on the load.

We also powered off all the nodes number 12 since, even with the fixes, the still presented the thermal issue sometimes.

Table 5 shows the time line for the process that has been described in this section: overall the project needed to dedicate a huge effort for about a month for fixing this instability issue. In this process, besides the constant work of Mont-Blanc system administrators and project collaborators at BSC, we should point out the fast reactivity of Bull/Atos in producing new functional firmware in short time, avoiding further delay in the project.
Figure 9: Thermal study
Figure 10: Thermal study after the deployment of the new blade firmware
### 10.2 OS noise analysis

Operating system noise involves any external activity that consumes resources such as CPU and was launched independently of the currently running program. In general, this kind of activity is an operating system service or a local daemon launched periodically with the objective of providing statistical information or ensure the correctness of the system. Usually, these programs have durations of the order of microseconds and the periodicity can vary from milliseconds to seconds or minutes. They mostly appear on every node in the system and are not usually attached to one CPU. The total amount of CPU time spent by this activity is below 1% of the total time. Additionally, it runs independently of the currently running user program, interrupting it whenever the period time accomplishes or some event occurs.

Figure 11 shows a trace-file execution of an OpenMPI program. This program consists on a loop, where each iteration is composed by fixed work quanta (just calculation without cache misses) and an user event at the end of it. There is no explicit coordination between the processes of the program during the loop. Each iteration is supposed to run for 4.8 milliseconds. The trace just shows the iterations whose duration exceeded a threshold (5.0 milliseconds, that is approximately 1.04X larger than the ideal case). As it can be seen, abnormal iterations are generated at unpredictable times and without any coordination between CPUs and nodes.

Even though the total amount of time spent in abnormal activity is below 1% of the total time, the impact can be more serious depending on the periodicity and type of synchronization that the user program has.

Figure 12 shows the execution trace-file of the same program of Figure 11, but with global synchronization at the end of each iteration: a call to the `MPI_Allreduce` operation. The duration variation between intervals is greater than when considering asynchronous iterations. A zoom of one of the irregular iterations is shown in figure 13. Figure 14 shows the original program trace. The orange region represents MPI_allreduce execution operation and the blue region represents calculation part. It can be seen how a delay on one CPU can unbalance the whole application as all the processes have to wait for the slower one generating the irregular iteration with an abnormal duration.

In order to quantify the impact of OS noise on programs with regular synchronization an evaluation of a microbenchmark where granularity is varied (amount of calculation or fixed time...
work quanta between global synchronizations) was done.

Figures 15 and 16 show the average slowdowns of such microbenchmark. The results correspond to evaluations carried on December 2014 and on June 2015 respectively.

In general, it can be seen that as the granularity diminishes, the performance of the program is degraded. The probability of some OS noise activity falling in exactly the same interval across all the node diminishes, so that they interrupt at different moments, unbalancing the processes that participate in the global synchronization and consequently degrading performance.

As can be seen in figure 15, in December 2014, applications showed an average slowdown of 8× in the worst case (128 cpus with a grain of 600 microseconds). However, after installing on the Mont-Blanc prototype the support for DVFS and after fixing the thermal issues described in 10.1 OS noise was reduced and consequently performance improved. The re-evaluation of the microbenchmark is shown in figure 16. Currently the average slowdown in the worst case is below 1.4×. Network latency in the form of just latency and noise can affect synchronization. However, in this case it showed to not be the source of slowdown. When doubling the number of nodes, in fact, one would expect to have more latency. If latency had been the unique or main source of slowdown, performance would have degraded when augmenting the number of nodes. However, performance improved (the micro-benchmark has no cache misses). Figure 17 shows a comparison slowdown for 128 cpus of the same program as Figure 12, with fixed work quanta and MPI_allreduce at the end of each iteration. It can be seen that even doubling the number of nodes, the slowdown doesn’t get worse. In fact performance increases even for fine
grain granularity. One would expect to have no slowdown in the case of 1 process per node because daemons should be migrated to other (idle) cpu and therefore all the noise should go to the idle cpu. This seems not to happen due to the fact that some daemons run on specific cpus (pinned) so they are not migrated. In any case, this slowdown behaviour requires further study.
Figure 15: Slowdown of regularly synchronized applications (Dec/2014)

Figure 16: Slowdown of regularly synchronized applications (June/2015)
Figure 17: Slowdown of regularly synchronized applications

References


[Del13b] Mont-Blanc deliverable 5.8, Prototype demonstration of energy monitoring tools on a system with multiple ARM boards, 2013.


