D5.5– Intermediate Report on Porting and Tuning of System Software to ARM Architecture Version 1.0

Document Information

Contract Number	288777
Project Website	www.montblanc-project.eu
Contractual Deadline	M24
Dissemintation Level	PU
Nature	Report
Coordinator	Alex Ramirez (BSC)
Contributors	Gábor Dózsa (ARM), Axel Auweter (BADW-LRZ), Daniele Tafani (BADW-LRZ), Vicenç Beltran (BSC), Harald Servat (BSC), Judit Gimenez (BSC), Ramon Nou (BSC), Petar Radojković (BSC)
Reviewers	Chris Adeniji-Jones (ARM)
Keywords	System software, HPC, Energy-efficiency

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^o 288777*

Change Log

Version	Description of Change
v0.1	Initial draft released to the WP5 contributors
v0.2	Version released to Internal Reviewer
v1.0	Final version sent to EU

Contents

Executive Summary

In the past, ARM-based processors were used mainly in the embedded and mobile domains. Applications in these domains typically execute on a single system-on-chip and use customized operating systems (e.g. Android) and libraries (e.g. multimedia libraries). Obviously, the characteristics of large-scale high performance computing (HPC) systems are fundamentally different: HPC applications are comprised of numerous software threads that execute in parallel in hundreds or thousands of interconnected compute nodes. Consequently, specific programming models, scientific and runtime libraries, compilers, and performance analysis tools are essential for the best implementation and execution of such applications. Moreover, economic and efficient operation of large-scale clusters requires dedicated management and monitoring tools to ensure optimal resource utilization at minimal energy consumption.

As the main goal of the Mont-Blanc project is to produce large-scale HPC clusters based on ARM processor architecture, one of its major challenges is to perform porting and tuning of already-existing system software for ARM-based HPC clusters. Deliverable 5.3 [MBD12a] summarizes our initial work in this regard (until month 12 of the project). In this deliverable, we report on the follow-up work in the second year of the project (month 12 - month 24). In particular, we summarize our efforts related to the parallel programming model and compiler, the development tools, and the scientific and runtime libraries. Furthermore, we report on the installation and customization of the operating system, the cluster monitoring and resource management, the performance monitoring and analysis tools, and the parallel distributed filesystem.

Key achievements for the reporting period are the installation of a test cluster of Insignal Arndale boards featuring the same Exynos 5 system-on-chip that will be used in the final Mont-Blanc system, the porting of the Nanos++ runtime system to OpenCL, a study of the new energy aware resource management features of the Slurm resource manager, and the porting of the Lustre parallel file system to ARM.

1 Introduction

With its aim to build a large-scale HPC cluster using embedded mobile technology, a substantial part of the Mont-Blanc project consists of porting and tuning existing system software tools and libraries for the target architecture.

This report follows up on the work reported in Deliverable 5.3 [MBD12a] focussing on the work performed in the months 12 – month 24 period of the project.

The main outcome of our activities is a fully-functional system software tuned for a small cluster comprised of InSignal Arndale boards [arn]. Each Arndale board comprises a Samsung Exynos 5 system-on-chip (SoC), the SoC that will be used in the final prototype of the Mont-Blanc project, which includes an ARM Cortex-A15 Dual Core processor at 1.7 GHz peak frequency and an ARM Mali-T604 GPU. The Arndale cluster has been assembled at the Barcelona Supercomputing Center and is available for experimental purposes to all Mont-Blanc partners.

The rest of this report is organized as follows:

- Section 2 summarizes our enhancements on the Nanos++ runtime environment and the Mercurium compiler. Also, we report on our efforts related to the OpenMPI library tuning for ARM-based architectures.
- Section 3 covers the description of the Linux kernels and distributions that have been installed on the Arndale board cluster.
- In Section 4, we list the runtime and scientific libraries that have been installed on the Arndale boards.
- In Section 5, we briefly report on the installation of the development, performance monitoring and performance analysis tools.
- The installation of system monitoring and resource management tools is discussed in Section 6. Furthermore, we present a detailed analysis of the resource management software features allowing for energy-efficient operation of the system.
- Finally, in Section 7, we analyze the portability of different parallel distributed filesystems for large-scale ARM HPC clusters. Specifically, we summarize the process of porting the Lustre filesystem to Carma boards [car] and we present a preliminary analysis on the benefits introduced by the Gluster filesystem.

2 Parallel Programming Model and Runtime Libraries

2.1 Nanos++ runtime system and Mercurium compiler

In the context of Mont-Blanc project, the Nanos++ runtime system [Nan] has been extended to support OpenCL. Providing OpenCL required implementing the architecture-dependent parts of Nanos++ plugins:

- Implement routines to detect OpenCL devices currently available in the system automatically and use them to execute OmpSs/OpenCL tasks.
- Implement an OpenCL cache so the Nanos++ caching/directory implementation can write, read and move data between/to/from those devices.
- Implement routines which can compile any OpenCL file once and reuse it as many times as needed (cached compiler).

In addition to this, Nanos++ had to be extended in order to give support for the NDRange Clause in OpenCL, which allows the programmer to annotate kernels as if they were regular OmpSs tasks/functions that can be called inside their code. Since the Mercurium compiler [Mer] needs to generate a wrapper function for these kernels, a few basic OpenCL operations had to be exposed by using a set of Nanos++ API calls that allow the following operations:

- Submission of a Nanos++ task to a specific back-end (in this case, OpenCL). This operation is supported by Regular Nanos++ back-end API calls.
- Compilation and setup of the kernel according to its parameters and NDRange values. These operations are supported by extra Nanos++ API calls used by Mercurium when creating an OpenCL task.
- Detection of the devices available for the execution of tasks.

Finally, support for the Fortran programming language was incorporated into the Mercurium compiler [Mer]. The following medium-size kernels were successfully compiled with Mercurium:

- Preconditioning kernel of BigDFT (C/C++)
- Reduced, non-production version of SPECFEM3D (FORTRAN)
- Reduced, non-production version of COSMO (FORTRAN)
- Thermostat routine of MP2C (FORTRAN)
- Hydro kernel (FORTRAN/C)

2.2 Tuning of the OpenMPI library for ARM Architecture

We have measured and optimized MPI point-to-point communications on Arndale boards. As we could use only two boards to evaluate the messaging performance, we have focused on intra-node and nearest neighbor point-to-point communications. We have measured latency and bandwidth by running a ping-pong microbenchmark.

The Arndale Development Boards

The Exynos5 chip does not have integrated Ethernet. There is a 100Mb Ethernet socket on board (that is connected to the cores via USB 2.0) and we used an external Gigabit Ethernet adaptor (from ASIX Ltd) connected to the board as a USB 3.0 device. As we plan to use the Gigabit Ethernet for application (i.e. MPI) messaging in the final prototype, we have focused our measurement to the performance of the Gigabit Ethernet. We use two boards connected by a direct Ethernet link.

The Pingpong Microbenchmark

To measure point-to-point latency and bandwidth we have used the usual pingpong microbenchmark approach. It measures the end-to-end round trip time of a single message between two processes to compute the latency and bandwidth. The measurement is done in a loop several times and the average time is calculated.

We used our custom implementation of the pingpong benchmark because we needed a flexible framework to explore many different scenarios including various messaging APIs and a custom time measurement API. Parameters like number of iterations to skip at the beginning (i.e. warm-up phase), number of iterations to measure, binding of processes to processors, buffer alignments, etc. can be defined as command line arguments for the benchmark.

Measuring the Time: the Timebase and the Timestamp Libraries

The pingpong benchmarks utilizes a custom time measurement library called Timebase. Timebase provides an instrumentation API for exact time measurement. The same API is implemented based on using ARM PMU Cycle Counter, the ARM Architected Timer Counter, the POSIX clock functions, and the PAPI library. The Timebase API also includes a simple timer interface to create, start/stop and dump software timers which also collect simple statistics (sum, max and min) automatically.

The ARM PMU cycle counter provides the highest resolution but it has the drawback of being affected by the actual clock frequency. Even worse, in case the core executes a *wfi*/*wfe* instruction the cycle counter simply stops. On the Arndale boards we have the ARM Architected Timer Counter and it runs at 24MHz that translates to 41ns resolution. That resolution is good enough even for fine grain latency measurements with the pingpong benchmarks.

In addition to collecting simple time statistics with software timers, the fine tuning of kernel parameters for latency optimization required detailed timing information about the distribution of the overhead among the different layers in the messaging software stack (including user and kernel space). For that purpose, we have implemented a Timestamp library that provides an application the ability to collect high resolution timestamps that are coherent between user and kernel space. The Timestamp library also utilizes the ARM Architected Timer Counter.

Test runs

We have measured the end-to-end latency and bandwidth of OpenMPI with different transport layers running at different core frequencies. For the sake of completeness, we have also measured performance of the 100MB Ethernet link - connected to the A15 cores via USB2 - in addition to the (external) Gigabit adapter - connected to the cores via the USB3 bus.

We note that above 1.4GHz we have experienced stability issues on the Arndale boards. This might have been the side effect of using a relatively old version of the Linux kernel (v3.6) but USB3 support seems to be broken in newer kernel versions. We have just received some hint how the USB3 issue could be solved in recent kernel versions.

We have used OpenMPI v1.6.3 and Open-MD v1.5.2 for the test runs.

We have compared the results against the numbers from Tegra2/SECO boards that have integrated

Gigabit Ethernet ports (but less powerful A9 cores).

2.2.1 Baseline Performance Results

Shared Memory

Figure 1 depicts the latency and the bandwidth curves resulted from running the pingpong benchmarks on a single node (i.e. OpenMPI defaults to the shared memory transport layer). As expected, the more powerful A15 core performs better than the A9. At 1.4GHz the latency and bandwidth improvements are 35% and 3X, respectively.

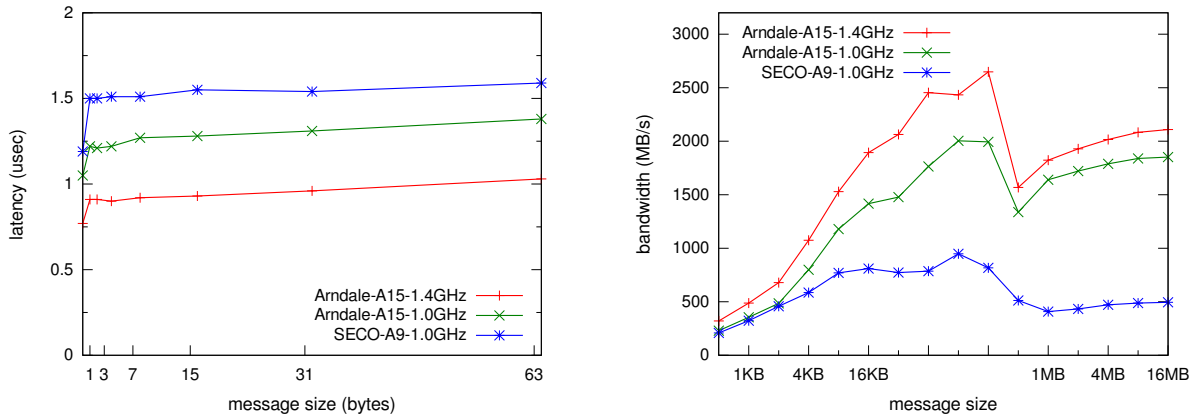


Figure 1: OpenMPI shared memory latency and bandwidth results

Gigabit Ethernet Results

Figure 2 depicts the latency and bandwidth curves we have observed running the vanilla OpenMPI version of the pingpong benchmark.

The better integration of the Gigabit Ethernet connection on the SECO boards results in better latency. Running the A15 at 1.4GHz we can still observe slightly higher latency than running the A9 at 1.0GHz.

The bandwidth curves are rather similar. The plot shows that the bandwidth cannot not apparently benefit from bumping up the frequency from 1.0GHz to 1.4GHz on the Arndale boards (if we use the vanilla OpenMPI library).

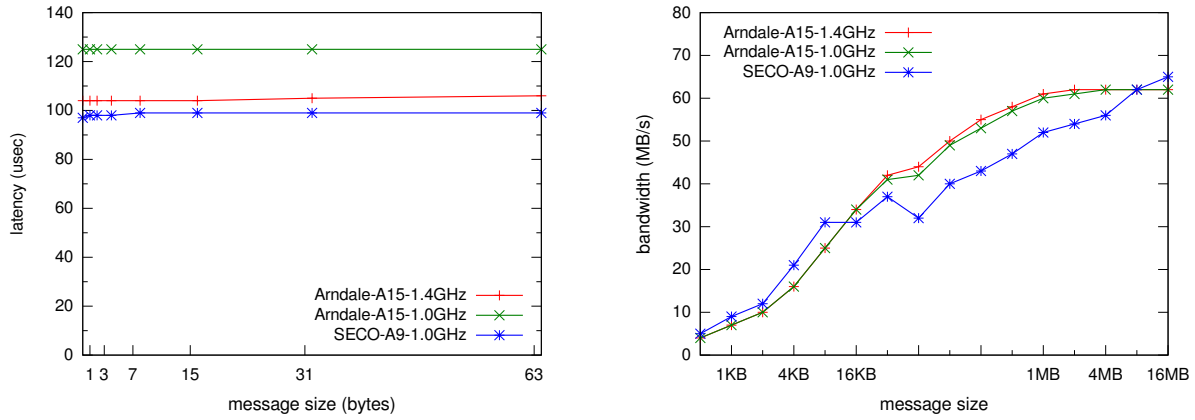


Figure 2: Vanilla OpenMPI over Gigabit Ethernet latency and bandwidth results

Figure 3 depicts the latency and bandwidth curves we have observed running the pingpong benchmark with OpenMPI using Open-MX as transport layer. Open-MX provides direct Ethernet access (i.e. eliminating the TCP/IP overhead).

Open-MX reduces the latency substantially on the SECO boards (compared to the vanilla OpenMPI numbers). We can also see some improvements on the Arndale boards but the effect is not that huge and the latency on the SECO board is roughly half than that of on the Arndale (running the A15 at 1.4GHz). We have found some kernel tuning knobs to alleviate this problem somewhat (which we explain in more detail later in Section 2.2.2).

Similarly to the latency, Open-MX improves substantially the bandwidth on the SECO boards but it only exhibits more modest improvement on the Arndale boards (and the observed peak bandwidth is 2X better on the SECO than on the Arndale). One possible problem with Arndale runs was that the ASIX Gigabit adapter did not work with MTU values larger than 1488 bytes while Jumbo frames were supported on the SECO boards. When a driver update from ASIX fixed the Jumbo frame issue we have re-run the bandwidth test as we explain it later in Section 2.2.2.

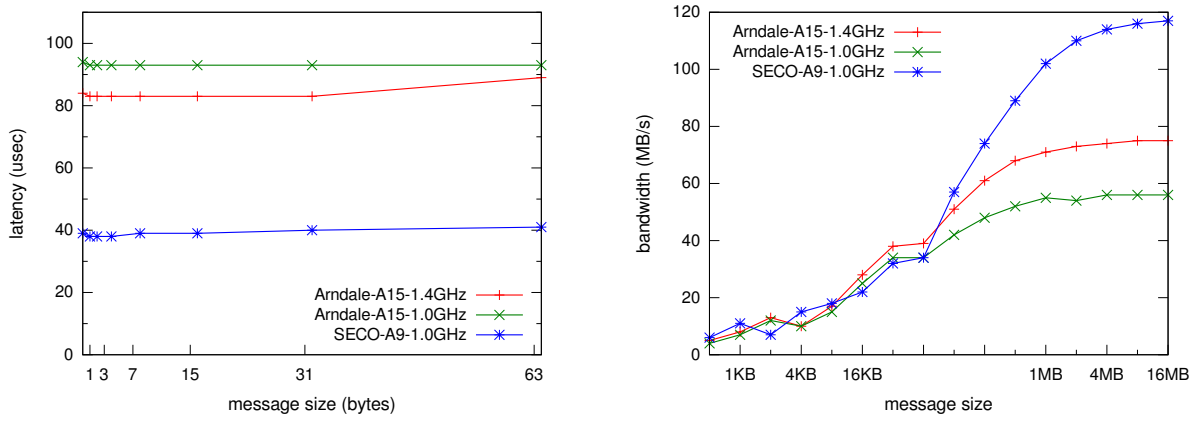


Figure 3: Latency and bandwidth results for OpenMPI using Open-MX transport over Gigabit Ethernet

2.2.2 Optimizations

We have applied various optimizations on the Arndale boards for the GigE/USB3 software stack including large MTU and tuning of USB related kernel and device parameters.

Bandwidth Results for Large MTU

Figure 4 depicts the bandwidth curves we observed running the pingpong test using OpenMPI with Open-MX transport layer and varying the MTU size between 1488 (default) and 4088 bytes (the maximum that the ASIX Gigabit adapter supports).

As the plot shows, the larger MTU does not help with the peak bandwidth we can observe. The only effect is that the curve reaches the max more quickly (i.e. at smaller message sizes). The peak bandwidth is still well below the theoretical limit of 120MB/s.

Tuning USB Kernel and Device Parameters

We have also tried to improve the end-to-end latency by tuning USB3 kernel parameters. We have discovered two parameters that effect latency substantially:

- **IMOD**: host controller (hcd) interrupt moderation value,

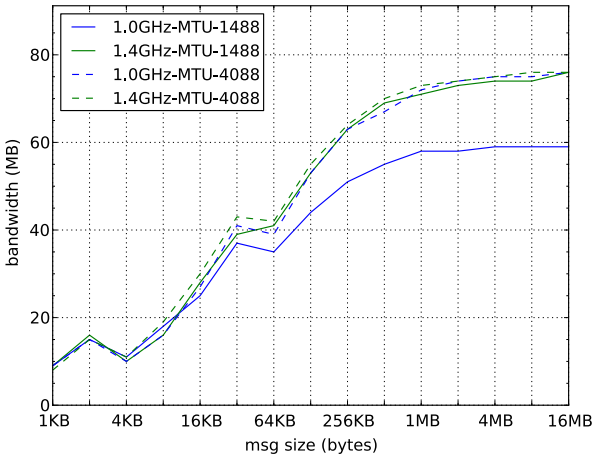
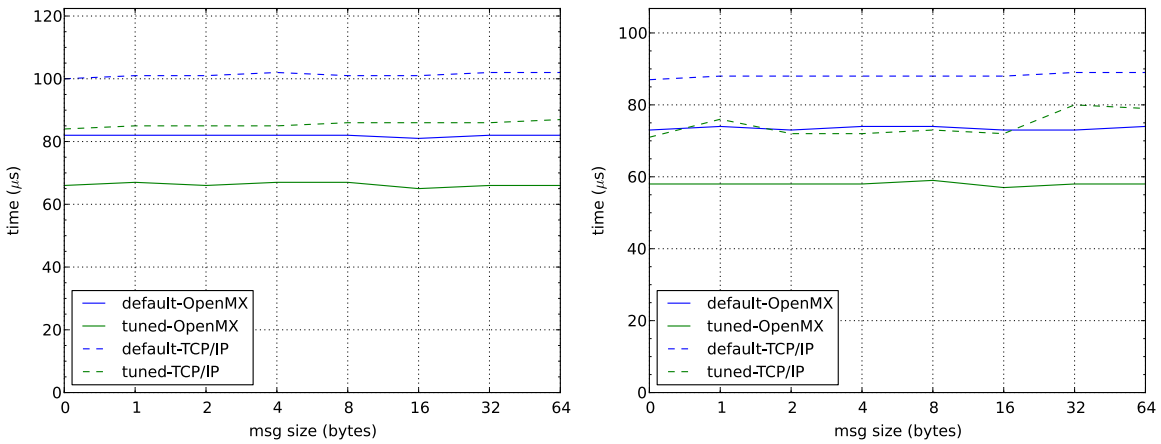


Figure 4: Bandwidth results for OpenMPI using Open-MX transport over Gigabit Ethernet with large (4K) frame size

- **IFG**: ASIX Gigabit adapter inter-frame gap.

We have run parameter sweep type tests with the pingpong benchmark. We used the Timestamp library to collect detailed information about how different values for those two kernel parameters change the execution time along the critical path from the user application down to the xhci-hcd (USB3 host controller) in the Linux kernel. We have concluded that the optimal values to use are IMOD=80 (the default value is 160) and IFG=1 (the default value is 64).



(a) 1.0GHz CPU frequency

(b) 1.4GHz CPU frequency

Figure 5: Improved latency results with optimized USB3 kernel parameters

Figure 5 depicts the latency curves running pingpong benchmarks with vanilla (TCP/IP) and Open-MX transport layer using the default and the tuned USB kernel parameters. In all cases, the optimized USB parameters result in 20% improvement in the end-to-end latency. The plots also show that raising the core frequency to 1.4GHz only slightly improves latency.

3 Operating System

We installed two Linux kernel images on the Arndale boards [arn]: version 3.9.0 and 3.0.31. Version 3.9.0 is one of the latest publically available Linux kernels. This kernel does not include OpenCL [Opeb] drivers for the Mali-T604 GPU, as opposed to version 3.0.31 which, instead, supports OpenCL and is provided by ARM. Both versions are currently installed in the Arndale board cluster.

- The Linux kernel version 3.9.0 has been compiled from the source code obtained from Linaro. Based on this kernel, we deployed a Debian 7 distribution [Deb] to the root filesystem. In this case, the kernel is loaded from the main file server using the Preboot Execution Environment (PXE) and the root filesystem is loaded from the main file server using Network File-System (NFS).
- For kernel version 3.0.31, ARM provided a whole system image, including a root filesystem which contains an Ubuntu 11.10 distribution [Ubu]. The Linux kernel image is loaded directly with U-boot from an SD card inserted at the SD card slot of the Arndale boards. The filesystem is also loaded from the SD card, except the /home folder, that is loaded from the main file server using the NFS. Since the network interface is 100MbE, having the filesystem at the SD card causes no problems and the performance is similar to the one achieved with the network filesystem. This image further includes the drivers needed for executing OpenCL programs on the Mali-T604 GPU.

Since Ubuntu is a Debian-based distribution, there are no incompatibilities between the two distributions. Also, all the software can be installed in both systems.

Besides the Linux kernel image and the root filesystem, the main file server also acts as Dynamic Host Configuration Protocol (DHCP) server to provide each node with an IP address.

Finally, we have experimented with a newer version (3.10) of the Linux kernel on the Arndale board. This enabled us to test advanced features such as the Cross-Memory Attach for shared memory communication between cache-coherent CPUs. Also, we have measured the impact of enabling Transparent Huge Page support in the kernel when running the Mont-Blanc micro benchmarks on the Exynos 5 Dual-powered Arndale board. The kernel patches for THP support are available from public repositories (Deliverable 5.4 [MBD13c]).

4 Scientific libraries

Most applications that are used in the Mont-Blanc project depend on external MPI or scientific libraries. The following libraries were installed on the Arndale boards:

- MPICH2 [MPI] and OpenMPI [Opea]: these are the most widely used open source implementations of the MPI specification. All applications required one of these MPI implementations when running on a cluster environment. MPICH2 has proved to be the most stable implementation in our target system, specifically when interacting with the SLURM job manager. Hence, MPICH2 is currently the default MPI implementation, although the user is still allowed to request OpenMPI.
- FFTW [FFT]: this library provides highly tuned Fast Fourier Transform (FFT) implementations.
- ATLAS [ATL]: is an open source implementation of the Basic Linear Algebra Subprograms (BLAS) interface. ATLAS also provides some routines from the Linear Algebra PACKage.

Also, the OpenCL libraries ViennaCL (C++ linear algebra), clFFT (FFT functions), and clBlas (BLAS functions) were installed and tested on the Arndale boards.

All the libraries have been installed from the standard repositories. In addition to this, we have been compiling the ATLAS library from the sources and were have been exploring tuning of this library to the Arndale boards.

5 Toolchain

5.1 Development Tools

We installed a GCC compiler set version 4.6 [GCC] on the Arndale cluster. This is a standard GCC version that is shipped with Ubuntu 11.10 and Debian 7, the distributions that are in use in the Arndale boards. The GCC compiler set includes a C compiler (gcc), a C++ compiler (g++), and a Fortran 90 compiler (gfortran). Ubuntu and Debian distributions also include the GNU Debugger (gdb) [GDB] versions 7.3 and 7.6 respectively.

5.2 Performance Monitoring Tools

We have installed the Ganglia monitoring system [Gan] on the Arndale boards and tested that the default metrics can be measured and reported via the Ganglia web-frontend. We have also tested the addition of custom metrics, such as CPU temperature and CPU clock frequency. We are planning to present our current efforts on performance monitoring at the Period 2 review (Deliverable D5.6 of the project [MBD13b]).

5.3 Performance Analysis Tools

The following Mont-Blanc performance analysis tools have been successfully ported and extended to support ARM platforms:

- Score-P and Scalasca have been ported and tested on the first Mont-Blanc prototype Tibidabo. Score-P has been further extended for prototypical support of the OmpSs programming model.
- The Extrae instrumentation library has been ported and extended to instrument OpenCL applications. Instrumentation of the OpenCL and hybrid MPI/OpenCL applications using Extrae was successfully tested on the Arndale boards.

The current state of the performance tools in context of the ARM architecture is detailed in the deliverable 5.7 of the project [MBD13a].

6 Cluster Management

System (cluster) management in high performance computing comprises two aspects: system monitoring and resource management. The system monitoring software allows for the status of the system's hardware and software components to be assessed. It is used by system administrators to ensure proper functioning of all system components. The resource management software, on the other hand, is an interface between the end user and the high performance computing cluster. It manages user request queues, monitors available resources, and allocates requested resources to the users.

6.1 System Monitoring

In order to monitor and ensure proper functioning of the Arndale boards, we have installed the system monitoring tool Nagios [Nag]. It allows for remotely monitoring and creating different statistics about the availability and state of various components of a computer cluster. The monitoring on our Arndale cluster includes computational nodes, the file server, and interconnection switch. Nagios is configured to monitor CPU Load and SSH access for each of the computational nodes in the system. In addition to this, Nagios monitors the following statistics and events on the LDAP and file server: CPU Load, Current Users, LDAP status, Delay of ping, Free space at root partition, SSH access, Swap usage, Total Processes. Nagios is configured to notify the system administrator via email in case any of the monitored services fails or it observes values that significantly differ from the expected ones.

6.2 Resource Management

Resource management on Arndale board cluster is performed by Slurm [SLU], a portable and open-source resource manager designed for Linux clusters. Currently, we use the following Slurm modules:

- Resource limits management: This module is used to ensure that the resource requirements of submitted jobs do not exceed the overall system resources.
- Multiple jobs queues with support for Quality of Service (QoS): This module enables different job queues with different priorities.
- Job accounting on per user basis. This module is required to keep track of resources consumed by each user.

6.3 Energy Aware Resource Management

In the previous report [MBD12a], we have shown the lack of energy-awareness in today's resource management software tools. We have also given hints on the necessary technical steps to improve the situation, such as the integration of node-level energy monitoring capabilities into the resource management software to allow for energy-based accounting schemes. Another demand was the exploitation of the dynamic voltage and frequency scaling capabilities of modern processors to tune the system's performance and power consumption according to the application's need. Fortunately, Slurm has recently made substantial improvements in the field of energy awareness and now supports both, per-job energy accounting as well as per-job controlling of CPU frequencies. The new features were introduced with the release of Slurm 2.5 and extended to provide easier access to external sensors in the latest 2.6 release.

6.3.1 Capturing the Energy Consumption of Applications in Slurm

One of the most significant improvements to Slurm for energy-aware operation is energy accounting. For this, Slurm can collect energy usage information from individual compute nodes and attribute the energy consumption to jobs. This enables us to evolve from the current accounting scheme that is solely based on computing time, to a more sophisticated accounting scheme that also takes into account the energy consumption of the hardware.

Many Slurm features are integrated using an easily extensible plugin infrastructure. This is also the case for the energy accounting feature. For some time already, Slurm is able to collect profiling statistics of applications using so called "Job Accounting Gather" plugins. One use case of the "Job Accounting Gather" plugins is to track the memory consumption of jobs. For this, the plugin code runs within the Slurm compute node daemon and monitors the memory footprint of the running applications. The Slurm master queries the plugin for this information at regular intervals and stores the information in the Slurm accounting database.

The "Energy Accounting Gather" plugin interface follows a similar approach for energy consumption information. The Slurm master queries all compute node daemons for the energy consumption information and the compute node daemons are responsible for retrieving the energy consumption from the actual hardware. Since a multitude of interfaces exists to capture energy consumption information in high performance computing systems (of which many are vendor specific), the plugin nature of the implementation enables developers to easily adapt Slurm's energy accounting feature to their platform. As of now, Slurm provides an implementation for two different ways of obtaining energy consumption information:

- RAPL (Running Average Power Limit), a technology built into Intel processors to estimate CPU and DRAM energy consumption.
- IPMI (Intelligent Platform Management Interface), an interface that is widely used for cluster monitoring.

As can be foreseen, the Mont-Blanc prototype will neither support RAPL nor IPMI, so a new plugin will have to be developed to support Slurm's energy accounting feature. Although the low-level interface for acquiring power consumption data is currently under negotiation in WP7 and not yet defined, we have already analyzed the API of Slurm for "Energy Accounting Gather" plugins. Data transfer between Slurm and the platform specific plugin happens via the `acct_gather_energy_t` data structure that contains information about the current power consumption of a node, the total energy consumed by a node, and a time-stamp indicating when the information was updated the last time.

In order to write a new "Energy Accounting Gather" plugin, one has to implement three functions:

1. `int acct_gather_energy_p_update_node_energy(void)`

   This function is called in regular intervals to trigger the platform-specific reading of the hardware sensors and store the obtained values in the local node's `acct_gather_energy_t` data structure. As all hardware energy counters will overflow after some time, this function allows capturing the hardware counters' data before such overflow happens without introducing additional communication overhead between the Slurm master and the compute node daemon.

2. `int acct_gather_energy_p_get_data(enum acct_energy_type data_type, acct_gather_energy_t *energy)`

   Depending on the `data_type` parameter, this function fulfills three different tasks:

   - If `data_type` is set to `ENERGY_DATA_JOULES_TASK`, this function adds the local energy consumption information to a global `acct_gather_energy_t` structure that is held per application.

- If `data_type` is set to `ENERGY_DATA_STRUCT`, this function returns the local node's `acct_gather_energy_t` structure.
- If `data_type` is set to `ENERGY_DATA_LAST_POLL`, this function returns a time-stamp indicating when the hardware sensors where queried for the last time.

3. `int acct_gather_energy_p_set_data(enum acct_energy_type data_type, acct_gather_energy_t *energy)`

   This function exists for transferring data to the plugin but remains unused at the moment except for testing purposes.

This interface acts on the compute nodes only, thus writing an "Energy Accounting Gather" plugin requires accessing the energy or power sensor information from within each node (so called "in-band" access). To facilitate the integration of sensors that can only be acquired externally (so called "out-of-band" access, e.g. through a distinct service network), Slurm 2.6 introduced another plugin API: the ExtSensors API. In contrast to the "Energy Accounting Gather" plugins, the "ExtSensors" plugin is triggered from the Slurm master. As of now, there exists only one implementation of an "ExtSensors" plugin which is capable of reading sensor data from RRD (Round Robin Database) files, a common file format for storing time series data that is used in many system monitoring applications. To develop an own "ExtSensors" plugin, the following five functions need to be implemented:

1. `int ext_sensors_read_conf(void)`

   This function is called to trigger the read of an "ExtSensors" configuration file which controls the behavior of the plugin.

2. `int ext_sensors_free_conf(void)`

   This function should implement routines to free the memory allocated for the data structures of the ExtSensors configuration.

3. `int ext_sensors_p_update_component_data(void)`

   This function is called in regular intervals to trigger the update of the internal data structures. This corresponds to the `acct_gather_energy_p_update_node_energy()` function of the "Energy Accounting Gather" plugin.

4. `int ext_sensors_p_get_stepstartdata(struct step_record *step_rec)`

   This function is called before a job (step) starts to initialize the information about external sensors stored in the job step's data structures.

5. `int ext_sensors_p_get_stependdata(struct step_record *step_rec)`

   This function is called at the end of a job (step) to allow for updating the information about external sensors in the job step's data structures.

Both Slurm APIs for collecting energy consumption information presented above allow for a straightforward implementation of a Mont-Blanc specific plugin that forwards this data to the resource management software. Which of the two plugin APIs will be used in Mont-Blanc largely depends on the low-level hardware interface that will be available to access the sensor information and whether it will be accessible "in-band" or "out-of-band". The information on the low-level sensor access is under preparation in WP7 and will be available within the next weeks.

6.3.2 Setting the CPU Frequency for Applications in Slurm

To start a parallel program, Slurm provides the "srun" command. Using the newly introduced "-cpu-freq" parameter of "srun" or the environment variable "SLURM_CPU_FREQ_REQ", users can request a frequency which will be set for all processors involved in executing the program. If the requested frequency is invalid for the processor, Slurm will select a frequency as close as possible to the requested value. To check, which frequencies are supported and to set the desired processor frequency, Slurm's compute node daemon makes use of the Linux CPUFreq interface [CPU], which is supported on any recent x86 platform and most ARM based system-on-chips.

6.3.3 Testing the Energy Efficiency of DVFS on the Arndale board

As discussed in Section 4.2 of Deliverable 5.3 [MBD12a], Dynamic Voltage and Frequency Scaling (DVFS) can reduce the energy consumption of a computing system substantially by dynamically scaling the CPU clock on the basis of the workload to which the system is subjected. The scientific community has devoted considerable attention to the evaluation of the effectiveness of this strategy, such as in the works described in [HF05] and [ECLV10]. With the introduction of the above mentioned Slurm features, the use of CPU frequency scaling becomes available for even production level machines. In order to have a preliminary assessment of the potential energy gain provided by DVFS for the Mont-Blanc system, we conducted a series of experiments on the InSignal Arndale board.

6.3.4 Experimental Setup

The setup of the experiment is shown in Figure 6. A multimeter is connected in series with the board and measures the electrical current in order to derive the overall power consumption. A heat sink is applied to the SoC in order to prevent system overheating.

We run our tests by selecting two of the Mont-Blanc microbenchmarks, first introduced in Deliverable 4.1 [MBD11], specifically:

1. **2D Convolution:** This benchmark performs a two-dimensional convolution of two input matrices, producing a third matrix by linear combination between each point of the first matrix with the neighboring points of the second one. The 2D Convolution is a fundamental operation for image editing and filtering, where both matrices can be respectively interpreted as an image to be edited and its associated filter.

2. **N-Body**: The N-Body benchmark simulates the gravitational interactions that occur within a group of N different elements, with the objective of predicting parameters such as absolute positions, velocities and force intensities of all the bodies of the system under observation. N-Body simulations are widely used for studying the gravitational interactions between different celestial objects.

In this section we only show results associated with the two benchmarks described above, since their interpretation leads to similar conclusions for the remaining Mont-Blanc microbenchmarks as well. All results have been obtained by running both benchmarks exclusively on the Cortex-A15 dual core with a fixed CPU frequency. Starting from 200 MHz, the frequency has been increased from one benchmark run to another in steps of 100 MHz up to the maximum frequency of 1.7 GHz. The following performance metrics have been obtained during the experiments:

1. **Time-to-Solution** (in seconds): it corresponds to the total execution time of the benchmark.

2. **Average Power Consumption** (in watts): as the name suggests, it corresponds to the average power consumed by the entire board while running the benchmark.

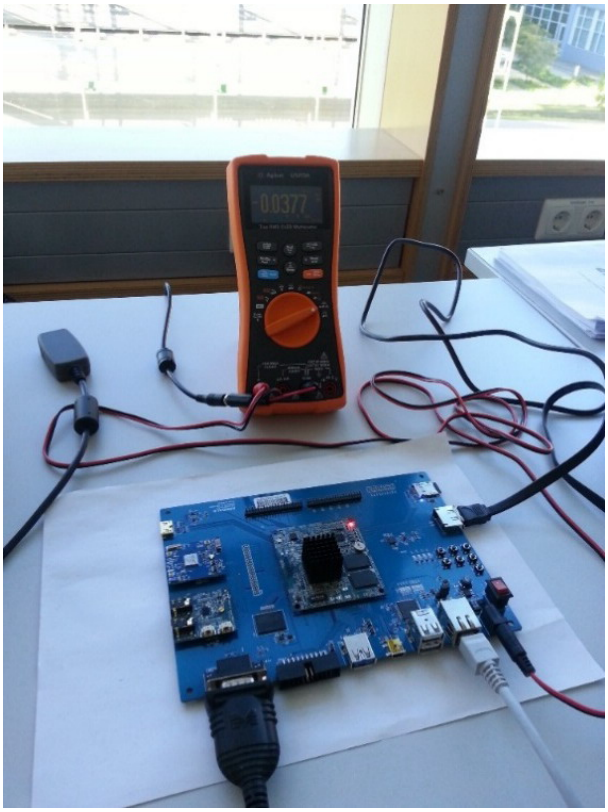


Figure 6: Picture of the experimental setup

3. **Energy-to-Solution** (in joules): the Energy-to-Solution corresponds to the total amount of energy consumed by the board during the execution of the microbenchmark. It is defined as the product of the Time-to-Solution and the respective average power consumption of the board.

The 2D Convolution benchmark consists of 300 iterations with an image width of 2048 pixels and a filter width of 5. The N-Body benchmark, instead, runs on 8192 elements for 20 iterations. All results have been obtained at float and double precision.

6.3.5 Results and Conclusions

Figure 7 and 8 respectively show the Time-to-Solution and the average power consumption of the 2D convolution benchmark. As expected, the total execution time decreases for increasing CPU frequencies, while at the same time the average power consumption grows. Note that the impact of the CPU clock rate to the Time-to-Solution becomes relatively negligible at high frequencies (as opposed to the case of lower frequencies, where setting the CPU clock rate from 200 MHz to 300 MHz allows a time reduction of more than 450 seconds or more than 30%). It can be further observed that, across the entire range of frequencies, the double precision version of the benchmark requires approximately 20% more time to complete than its corresponding single precision version. Both versions lead to similar results in terms of average power consumption, as depicted in Figure 8, which shows the same increasing exponential trend.

Finally, the Energy-to-Solution of the 2D Convolution benchmark is depicted in Figure 9. First of all we note that the total energy consumption progressively decreases for increasing CPU frequency values. This is mainly due to the fact that, at high CPU clock rates, faster completion times compensate for the high power consumed during the execution of the benchmark. At the same time, the impact of

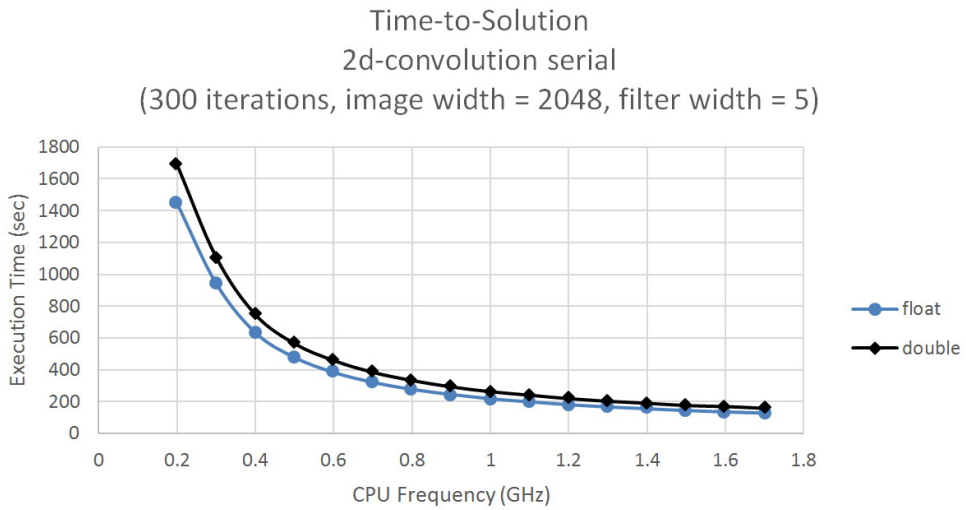


Figure 7: Time-to-Solution of the 2D Convolution benchmark

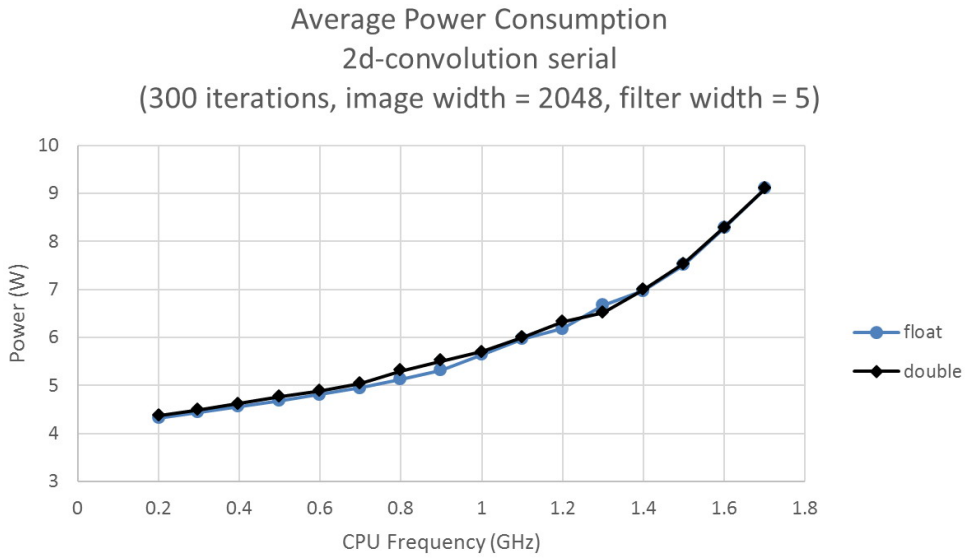


Figure 8: Average power consumption of the 2D Convolution benchmark

higher frequencies to the amount of energy saved is relatively small when compared to low frequency regions (e.g., switching from 200 MHz to 300 MHz allows saving approximately 2000 J in both versions of the benchmark, whereas switching from 1.2 GHz to 1.3 GHz does not introduce substantial benefits in terms of total energy consumption). On the contrary, the energy slowly starts increasing again for CPU frequencies higher than 1.5 GHz; in fact, at these clock rates, the benchmark execution is not fast enough to compensate for the higher power consumption of the board (theoretically indicating a progressive increase of the energy consumption for frequency values that go beyond the Cortex-A15 limit of 1.7 GHz). However, the observed energy consumption increment is negligible, suggesting that the extra cost in terms of energy is still acceptable to maximize performance.

Similar conclusions can be drawn for the N-Body benchmark by looking at the results depicted in Figures 10, 11, and 12. All of the performance metrics under analysis show behavior equivalent to that of the 2D Convolution benchmark, hence the same interpretation of results holds. Moreover, in the case of the N-Body benchmark, we can even observe the highest energy savings in correspondence of the maximum CPU clock rate value of 1.7 GHz, thus making the usage of DVFS essentially ineffective.

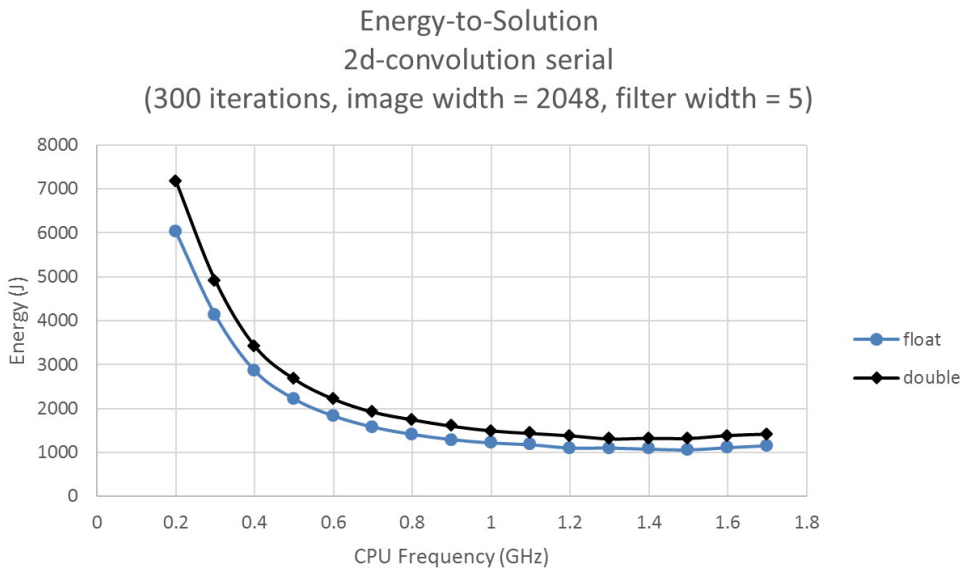


Figure 9: Energy-to-Solution of the 2D Convolution benchmark

As we mentioned in the beginning of this section, similar results have been obtained for the majority of the Mont-Blanc microbenchmarks. Hence we come to the preliminary conclusion that the employment of DVFS to the Cortex-A15 does not generally lead to significant energy savings. For this reason, we believe that running all jobs at high CPU clock frequencies should be the recommended procedure to follow for the final Mont-Blanc system as well. Future investigations will involve the possibility of applying DVFS on the Mali-T604 GPU, in order to assess its effectiveness in terms or energy consumption reduction.

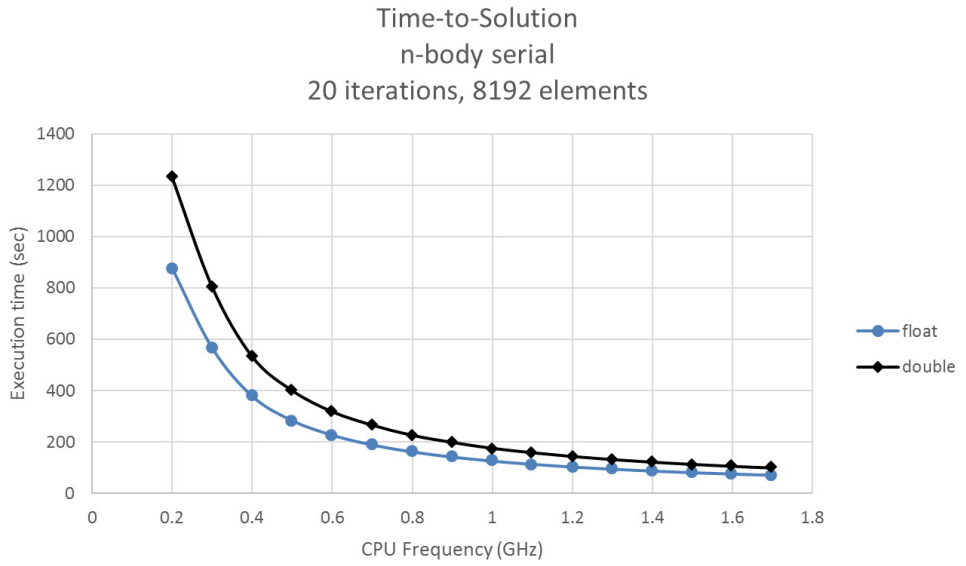


Figure 10: Time-to-Solution of the N-Body benchmark

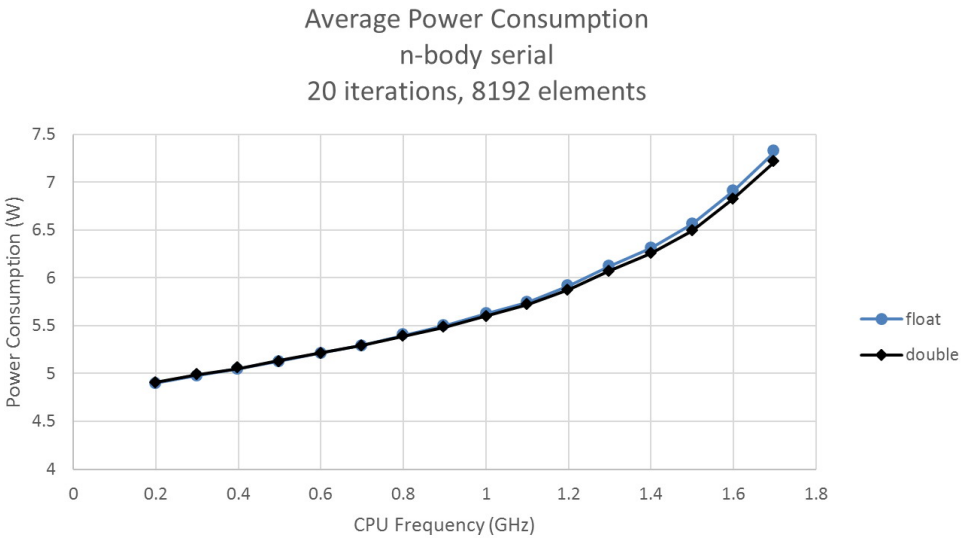


Figure 11: Average power consumption of the N-Body benchmark

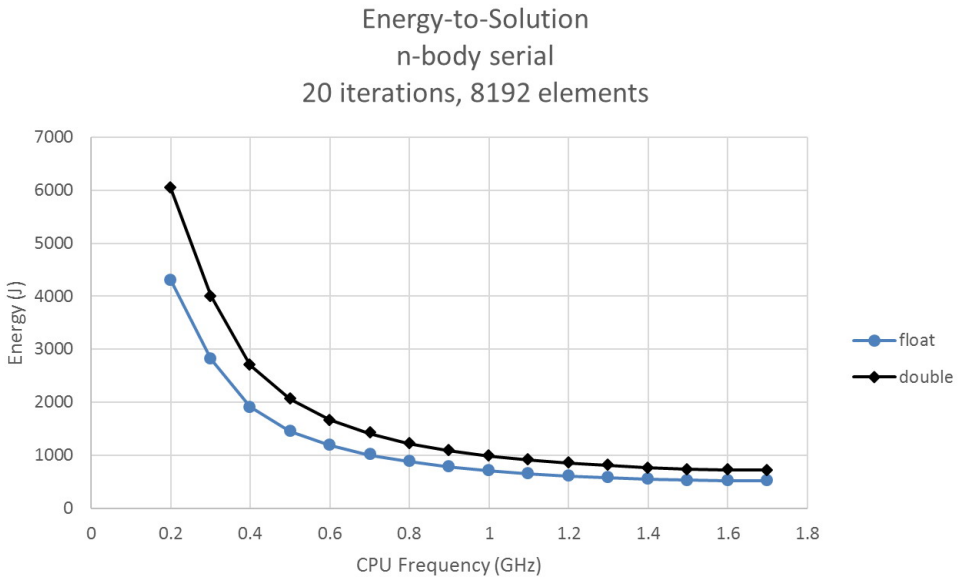


Figure 12: Energy-to-Solution of the N-Body benchmark

7 Parallel distributed filesystem for large-scale HPC clusters

In deliverable 5.2 [MBD12b], we summarized our initial work related to porting the Lustre parallel filesystem to ARM architectures. In particular, we ported Lustre 2.1.X to Linux kernel version 2.6.36. Since this kernel version is not directly supported by Lustre, we needed to make some adjustments in the Lustre client. In general, as the Lustre client uses specific interfaces via kernel modules to provide a high-performance parallel file system, it is required to adapt the Lustre client to each kernel version that is not directly supported.

In this report, we summarize the work that was required to port Lustre to the Carma DevKit [car] running Linux kernel version 3.1.10. After solving numerous problems that required significant effort, we were able to successfully install a fully-functional Lustre client on the Carma DevKit. However, we have detected an important drawback of the Lustre filesystem – it requires significant changes in order to run on Linux kernels that are not directly supported by Lustre. At this point, we are not able to predict the effort required to port Lustre to new versions of the Linux kernel running on the ARM-based HPC prototypes that will be developed in the future. Therefore, in addition to Lustre, we have decided to explore other filesystems and to analyze their suitability for large-scale HPC clusters based on the ARM architecture.

7.1 Environment

The experimental environment for setup and testing of parallel distributed filesystems is similar to the one that has been already described in deliverable 5.2 [MBD12b]. The experimental environment comprises a laptop that is used to configure the basic services (VirtualBox, NFS, DHCP, and TFTP) and a Carma DevKit [car] that is used as a client (see Figure 13). On the server side, we used two versions of Lustre – 2.1.x and 2.3.x. The Lustre 2.1.x server was executed directly on the physical hardware, without VM. The installation of the Lustre 2.3.x server, however, required the setup of a new VM due to kernel version restrictions.

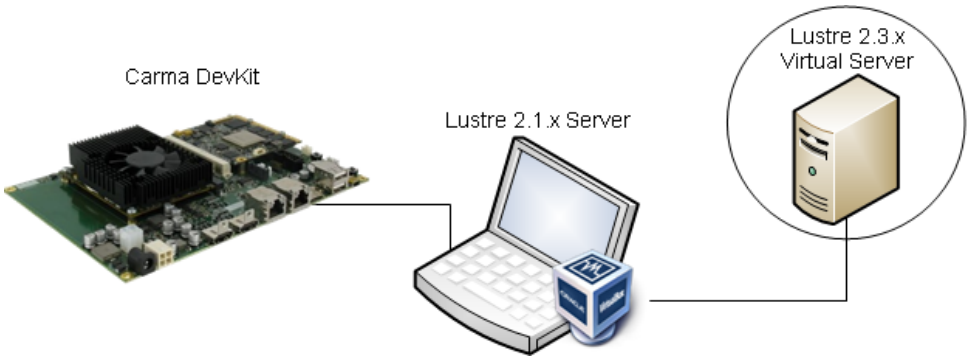


Figure 13: Overview of the experimental environment

7.2 Porting Lustre to ARM architectures

We have ported two versions of Lustre to the Carma DevKit running Linux kernel 3.1.10:

1. Version 2.1.x – previous Lustre version.
2. Version 2.3.x – next maintenance version of Lustre.

7.2.1 Porting of Lustre 2.1.x to the Carma DevKit running Linux kernel 3.1.10

Lustre has been compiled directly on the Carma DevKit board as it required an MPI compiler and cross-compiling appeared quite cumbersome to setup. Still, the compilation of Lustre on the Carma DevKit running Linux kernel 3.1.10 required major patching. We started with the Lustre patches [Whaa] that target 3.0 Linux kernel. At that point, we detected numerous problems that are caused by the changes in the new kernel version. Some of the issues that we detected are listed below:

- autoconf.h no longer exists
- Semaphores and mutexes have changed
- Some variables are atomic: d_count, mnt_count
- Setting attributes to an inode changed
- Sockets and file structures changed
- blkdev functions changed
- sk_sleep structure in socket changed to a function
- dentry locking removed
- ctl_name and strategy fields unused

In order to address these issues, we applied more than 200 Kb of patches, some of them requiring manual changes. We still had some problems with *dentry* that required major changes. At that point, we concluded that we could not provide a fully functional Lustre version 2.1.x running on Linux kernels 3.x. Therefore, we decided to test a recent Lustre code version 2.3.x (next maintenance version) that is prepared to compile on Linux kernels 3.x.

7.2.2 Porting of Lustre 2.3.x to the Carma DevKit running Linux kernel 3.1.10

Porting of Lustre 2.3.x to the Carma DevKit running Linux kernel 3.1.10 required the following steps:

Step 1: We downloaded the latest 2.3.x version from the git repository and configured it using:

```
./configure --with-linux=/home/ubuntu/kernel-source/
--disable-libcfs-cdebug --disable-libcfs-trace
--disable-libcfs-assert --disable-server
```

Step 2: The code is compiled with a make command. Also, we needed to apply some minor changes to the kernel:

```
make headers_prepare
make modules_prepare
ln -s /usr/src/kernel/arch/arm/include/asm /usr/src/kernel/include/asm/
cat kernel_source/include/generated/utsrelease.h >>
kernel_source/include/linux/version.h
```

Step 3: We needed to change some code, this time related to the ARM architecture under study. Some debug functionality is not available on the Carma DevKit, thus we had to modify /libcfs/libcfs/linux/linux-debug.c. The following code is included:

```
#undef HAVE_DUMP_TRACE
#undef HAVE_STACKTRACE_WARNING
```

We also commented the .warning and .warning_symbol lines, since they are not available on the Carma DevKit.

Step 4: Once that all the modules compiled, we did "insmod" in the following order:

- insmod ./libcfs/libcfs/libcfs.ko
- insmod ./lnet/lnet/lnet.ko
- insmod ./lnet/klnds/socklnd/ksocklnd.ko
- insmod ./lustre/lvfs/lvfs.ko
- insmod ./lustre/obdclass/obdclass.ko
- insmod ./lustre/obdclass/llog_test.ko
- insmod ./lustre/ptlrpc/ptlrpc.ko
- insmod ./lustre/osc/osc.ko
- insmod ./lustre/lov/lov.ko
- insmod ./lustre/mgc/mgc.ko
- insmod ./lustre/fld/fld.ko
- insmod ./lustre/fid/fid.ko
- insmod ./lustre/lmv/lmv.ko
- insmod ./lustre/mdc/mdc.ko
- insmod ./lustre/llite/lustre.ko

A hardware-related warning appeared (listed below). However, our tests showed no noticeable drawbacks on these warnings.

```
LNet: 5:0:(linux-cpu.c:1035:cfs_cpu_notify())
Lustre: can't support CPU hotplug well now,
performance and stability could be impacted[CPU 2 notify: 5]
```

Step 5: Finally, once the modules were loaded, the client could mount the filesystem:

```
sudo ./mount.lustre <serverip>:/<lustre-fs-name> /mnt
```

Lustre Version	Server Support	Client Support
1.8.3-1.8.5	RHEL 5, CentOS 5, SLES 10, SLES 11	RHEL 5, CentOS 5, SLES 10, SLES 11
1.8.6 - 1.8.8	RHEL 5, CentOS 5	RHEL 5, CentOS 5, RHEL 6
2.0	RHEL 5, CentOS 5	RHEL 5, CentOS 5, SLES 11
2.1 - 2.1.3	RHEL 5, CentOS 5, RHEL 6, CentOS 6	RHEL 5, CentOS 5, RHEL 6, CentOS 6, SLES 11
2.2	RHEL 6, CentOS 6	RHEL 5, CentOS 5, RHEL 6, CentOS 6, SLES 11
2.3	RHEL 6.3, CentOS 6.3	RHEL 6.3, CentOS 6.3, RHEL 5.8, CentOS 5.8, SLES 11 SP1, 2.6.38

Table 1: Officially supported server and client distributions (for different Lustre versions).

7.2.3 Installing the Lustre server (on x86 architecture)

For the installation of the Lustre server we followed the instructions from the following documents:

1. Walk-thru- Deploying a Lustre pre-built RPMs [Whaf]
2. Lustre Manual [Whad]

A Lustre server can potentially be installed on any version of Linux distribution and kernel on x86 machines. However, only specific combinations or server and client distributions are supported officially [Whae], as can be seen in Table 1. In order to avoid possible problems due to incompatible server/client distributions, we decided to use a CentOS 6.3 distribution on the server side.

Lustre server requires two components: one metadata service (MDS) with a Metadata Target (MDT), and one or more Object Storage Services (OSS) with several Object Storage Targets (Devices storing data). Additional OST increase the storage capacity, additional OSS increase the bandwidth. The installation of a functional Lustre server (with MDS and OSS on the same node) requires the following steps:

1. Install CentOS 6.3
2. Deactivate selinux (/etc/selinux/config set to disabled)
3. Download the kernel precompiled packages (from [Whac]): kernel, kernel-firmware
4. Install the packages: yum localinstall ./kernel*
5. Download e2fsprogs precompiled packages([Whab]): e2fsprogs, e2fsprogs-libs, libcom_err
6. Install libcom_err using rpm tool : rpm –nodeps -U libcom_err-1.42.6.wc2-7.el6.x86_64.rpm
7. Proceed with the installation of e2fsprogs: yum localinstall ./e2fsprogs-*
8. Download Lustre precompiled packages([Whac]): lustre, lustre-ldisks, lustre-modules, lustre-tests
9. Install Lustre packages: yum localinstall ./lustre-*

Once that the server is installed, it is required to reboot into the new kernel and prepare the different filesystems:

1. Create the MGS and MDT:

```
mkfs.lustre --fsname=<fsname> --mgs --mdt --index=0 <block device name>
mount -t lustre <block device name> /mdtnode/
```

2. Create the OST, specifying the ip (NID) of the MGS:

```
mkfs.lustre --fsname=<fsname> --mgsnode=<NID>\
  --ost --index=<OST index> <block device name>
mount -t lustre <block device name> /ostnode/
```

The different devices can be added to fstab in order to run at boot time.

7.2.4 Status

We have a fully-functional Lustre client version 2.3.x working in the Carma DevKit running Linux kernel 3.1.10. POSIX tests were passed in a satisfactory way. The client does not show any performance drawbacks because of running on the ARM hardware and can support up to 85 MB/s of throughput (limited by the standard TCP stack, and the network driver).

Lustre server (2.3.x) has successfully been installed on x86 architecture running CentOS 6.3 Linux distribution.

7.3 Alternative parallel filesystems

Filesystems can have significant impact on the performance and the scalability of large-scale HPC clusters. Therefore, when building a HPC cluster, one of most important design decisions is the choice of the filesystem.

In deliverable 5.2 [MBD12b], we presented an extensive analysis of the filesystem features that are required for the large-scale HPC clusters based on ARM architectures. This analysis motivated us to select the Lustre filesystem as the preferred candidate: it is a well-known, open-source, POSIX-compliant filesystem that provides high performance while scaling to tens of thousands of nodes.

However, an important drawback of the Lustre filesystem is that it requires significant changes in order to run on Linux kernels that are not directly supported by Lustre. In this report, we have briefly described all the problems and demonstrated the effort that was required to adjust Lustre to new versions of the Linux kernel. As we are not able to predict the effort required to port Lustre to future prototypes, we have decided to explore other filesystems and to analyze their suitability for large-scale HPC clusters based on ARM architecture.

7.3.1 Gluster

Gluster [Glu] is a user-level filesystem based on FUSE. Since it is a user-level filesystem, it can be installed on any kind of software/kernel/hardware combination. Though GlusterFS is a File System, it uses already tested disk file systems like ext3, ext4, xfs, etc. According to the developers, it can easily scale up to petabytes of storage. Gluster distributes files (round robin) among the different file server nodes, and it has other distribution scenarios based on stripes and replication which are convenient with big files.

During initial tests we were able to switch from an NFS filesystem to a Gluster filesystem without significant problems and without having to touch any kernel sources. We had no issues when using configure/make operations of Gluster on the Carma DevKit running Linux kernel 3.1.10 and newer versions.

In the initial experiments on the Carma DevKit, the Gluster performance matched the performance of Lustre. Currently, we are performing more extensive tests with different server-client layouts and more applications. Also, we plan to explore a setup of Gluster server on ARM-based hardware to analyze the energy efficiency with respect to the conventionally used x86 filesystem servers.

References

[arn] Arndale board. http://www.arndaleboard.org/.

[ATL] ATLAS: Automatically Tuned Linear Algebra Software. http://math-atlas.sourceforge.net/.

[car] Carma Development Kit. http://www.nvidia.com/object/seco-dev-kit.html.

[CPU] CPU Freq. https://www.kernel.org/doc/Documentation/cpu-freq/index.txt.

[Deb] Debian Testing. http://www.debian.org/devel/testing.

[ECLV10] Maja Etinski, Julita Corbalan, Jesus Labarta, and Mateo Valero. Optimizing job performance under a given power constraint in HPC centers. In *Proceedings of the International Conference on Green Computing*. IEEE Computer Society, 2010.

[FFT] FFTW library. http://fftw.org/.

[Gan] Ganglia monitoring system. http://ganglia.sourceforge.net/.

[GCC] GCC, the GNU Compiler Collection. http://gcc.gnu.org/.

[GDB] GDB: The GNU Project Debugger. https://www.gnu.org/software/gdb/.

[Glu] GlusterFS – Opensource, distributed filesystem. http://www.gluster.org.

[HF05] Chung Hsing Hsu and Wu Chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the ACM/IEEE SC 2005 Conference*, 2005.

[MBD11] Preliminary report of progress about the porting of the full-scale scientific applications, Mont-Blanc Project Deliverable Document 4.1, 2011.

[MBD12a] Preliminary report on porting and tuning of system software to ARM architecture. Mont-Blanc Project Deliverable Document 5.3, 2012.

[MBD12b] Report on Parallel and Distributed Filesystems on the BSC ARM Prototype. Mont-Blanc Project Deliverable Document 5.2, 2012.

[MBD13a] Prototype demonstration of performance analysis tools on a system with multiple ARM boards, Mont-Blanc Project Deliverable Document 5.7, 2013.

[MBD13b] Prototype demonstration of performance monitoring tools on a system with multiple ARM boards, Mont-Blanc Project Deliverable Document 5.6, 2013.

[MBD13c] Report on Tuned Linux-ARM kernel and delivery of kernel patches to the Linux kernel, Mont-Blanc Project Deliverable Document 5.4, 2013.

[Mer] Mercurium compilation infrastructure. http://pm.bsc.es/mcxx.

[MPI] MPICH2: High-performance and Widely Portable MPI. http://www.mcs.anl.gov/research/projects/mpich2staging/goodell/index.php.

[Nag] Nagios - The Industry Standard in IT Infrastructure Monitoring. http://www.nagios.org/.

[Nan] Nanos++ runtime. http://pm.bsc.es/nanox.

[Opea] Open MPI: Open Source High Performance Computing. http://www.open-mpi.org/.

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

[SLU] SLURM: Simple Linux Utility for Resource Management. https://computing.llnl.gov/linux/slurm/.

[Ubu] Ubuntu 11.10 - Oneiric Ocelot. http://old-releases.ubuntu.com/releases/oneiric/.

[Whaa] Whamcloud. 3.0 patches. http://jira.whamcloud.com/browse/LU-812.

[Whab] Whamcloud. e2fs builds. http://build.whamcloud.com/job/e2fsprogs-master.

[Whac] Whamcloud. Lustre builds. http://build.whamcloud.com/job/lustre-master/.

[Whad] Whamcloud. Lustre manual. http://build.whamcloud.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.pdf.

[Whae] Whamcloud. Lustre support matrix. http://wiki.whamcloud.com/display/PUB/Lustre+Support+Matrix.

[Whaf] Whamcloud. Walkthru - lustre server installation. http://wiki.whamcloud.com/display/PUB/Walk-thru-+Deploying+a+Lustre+pre-built+RPMs.