D5.3– Preliminary 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	M12
Dissemintation Level	PU
Nature	Report
Coordinator	Alex Ramirez (BSC)
Contributors	Axel Auweter (BADW-LRZ), Gabor Dozsa (ARM), Isaac Gelado (BSC), Nikola Puzovic (BSC), Nikola Rajovic (BSC), Daniele Tafani (BADW-LRZ)
Reviewers	Chris Adeniyi-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

The Mont-Blanc project will produce the first large-scale supercomputer based on ARM cores. The ARM architecture has been succesfully used in the past in embedded and mobile platforms. However, the requirements and constrains of those platforms greatly differ from the needs of a High Performance Computing (HPC) system. One of these major differences is the system software used in each environment.

Embedded and mobile computing programmers typically use Operating Systems and libraries customized for their target application (e.g., Android). Moreover, such platforms typically target applications that run in a single MPSoC chip. This is in contrast to a typicall HPC environment, where general purpose operating systems (e.g., Linux) and scientic libraries (e.g., BLAS) are used to run applications in hundreds or thousands of compute nodes in parallel.

This document describes initial work done to create a functional HPC system based on ARM cores, from the operating system, to the scientific libraries, and parallel execution. Such work does not only involve the port of system software to the ARM architecbure, but also tuning these software components to fully exploit the characteristics of ARM cores. Similarly, the cluster management system also needs to be adapted to the characteristics of ARM-based nodes and to the goal of achieving very high energy efficiency.

1 Introduction

The Mont-Blanc project aims to build the first supercomputer based on low-power processors based on the ARM architecture. Such processors have traditionally been used to build mobile and embedded platforms, where an embedded operating system (or no operating system at all) is used. Moreover, the libraries used in mobile and embedded platforms (e.g., multimedia, signal processing, etc.) are not used in high performance computing (HPC) environments at all. Furthermore, those libraries which are common to both HPC and embedded systems (e.g., standard C library) require different optimization strategies; e.g., performance vs. size.

In this document we describe the preliminary porting and tuning efforts in the operating system and system libraries within the Mont-Blanc project. A major difficulty in this work is the lack of the final Mont-Blanc prototype, so the initial work has been done using close-enough system: the Tibidabo ARM cluster built in the PRACE project [PRA12] being implemented in the Barcelona Supercomputing Center (BSC). Tibidabo is a cluster formed of 128 nodes distributed in two racks. Each node is formed by an ARMv7-compliant carrier board, containing an NVIDIA Tegra 2 with a dual-core ARM Cortex A9 processor, 1GB of main memory and two network interfaces (1GbE and 100MbE).

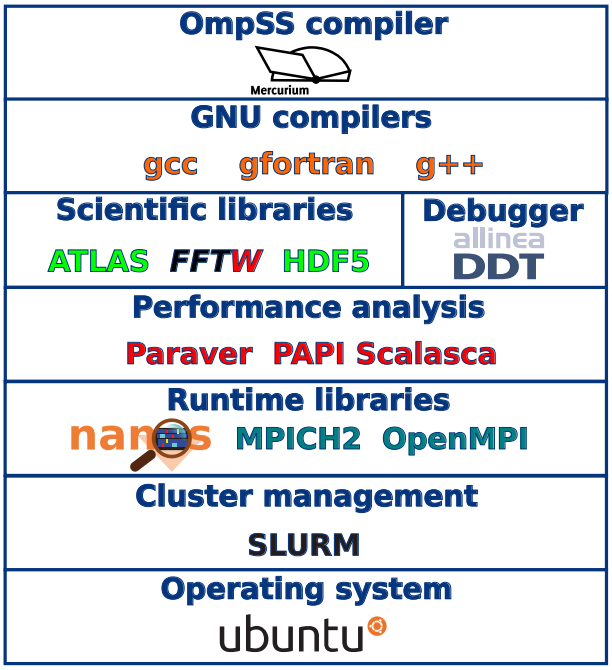

Figure 1: Software stack installed on the Tibidabo Prototype

The work described in this document encompasses the system software layers within the Mont-Blanc software stack. Figure 1 shows the Mont-Blanc software stack, which has also been deployed in the Tibidabo prototype.

This document mostly describes the work done to have a fully functional system software stack in our ARM/HPC system, which enables developers to compile, run, debug, and profile applications. We also describe the initial setup to enable profiling of the system software stack, which is a required step towards being able to tune the system software for HPC applications. Finally, we describe the planned future software development, porting and profiling efforts.

2 Node System Software

In this section we describe the initial effort done to have a functional ARM node, which forms part of the HPC system made available to the other Mont-Blanc Work-Packages. Because of the need of other Mont-Blanc WPs to have access to a functional ARM/HPC environment suitable for developing, debugging, and profiling tasks, we have prioritize functionality over the tuning of the system software stack. We also describe the planned work in this WP for future node system software development.

2.1 Linux Operating System

The Tibidabo hardware was delivered by the manufacturer with a Linux kernel version 2.6.32 (in SMP Preempt configuration). Based on this Linux kernel we have deployed a Ubuntu Linux 10.10 [Ubu10] distribution, which conforms the root filesystem of all the nodes in the cluster.

The Linux kernel image is loaded at boot time by each cluster node using the Trivial File Transfer Protocol (TFTP) from a main file server. This server also exports the root file system to all the nodes as a Network File-System (NFS). The root filesystem is being regularly updated with the latests compilers and development tools to provide users with the most recent versions of all the base software.

Besides the Linux kernel image, and the root filesystem, the file server also acts as Dynamic Host Configuration Protocol (DHCP) server to provide each node with two different IP addresses, one for each network interface present in the node. The 1GbE interface gets an IP address from the communication network, while the 100MbE gets an address from the filesystem network. This network configuration isolates MPI messages from regular file access operations, so the effect of large I/O operations (e.g., dumping profiling traces to disk) do not disturb the regular application execution.

The currently installed Linux Kernel image is quite outdated (as September, 2012 the current Linux Kernel image is 3.5.3 [Ker12]). We have not invested any time to port the most updated image to our target platform because most source code modifications are not related to the ARM architecture, but to the specifics of the chip (i.e., NVIDIA Tegra2 [Teg12]) used in the cluster. Hence, any effort invested in porting recent kernel codes to the current environment are not likely to be useful for the final Mont-Blanc prototype. We do not plan to perform any further Linux kernel development effort until the target chip for the Mont-Blanc prototype is selected by WP7.

The file server has been also configured to allow a set of nodes to load a different Linux Kernel image and/or a different root filesystem. This functionality allows testing new compilations of the Linux Kernel and/or new root filesystems (e.g., Ubuntu 12.10) without interrupting the normal operation of the cluster. Further development and work on the filesystem server is described in the D5.2.

2.2 Development Tools

The standard GCC compiler set [GCC12] shipped with Ubuntu 10.10 is version 4.4. This toolchain includes a C compiler (i.e., gcc), a C++ compiler (i.e., g++), and a Fortran 90 compiler (i.e., gfortran). Besides the this standard compilers, we have compiled and installed the latest GCC toolchain versions (4.6.2, and 4.7.0), which the user can select at runtime.

Ubuntu also ships the GNU Debugger [GDB12] (i.e., gdb) version 7.2, which is available in all nodes. We have also installed the Allinea DDT debugger [All12], which is widely use to debug MPI applications on cluster environments.

We have automatize the compilation and deployment of new GCC tool-chains. New versions of GCC are being installed on the Tibidabo system as they are released. Although we plan to continue using GCC to generate ARM binaries, we plan to augment the compiler toolchain with the necessary components to generate binary code for the accelerator included in the MPSoC selected by WP7 (e.g., CUDA compiler).

2.3 System and Scientific Libraries

Most target applications in the Mont-Blanc project depend on external MPI and/or scientific libraries. We analyzed the library dependencies of the each target applications in WP4 when compiled for the x86 architecture. After this analysis we found the following dependencies which where installed from the standard Ubuntu repositories:

- MPICH2 [MPI12] and OpenMPI [Ope12a]: 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 MPI implementation in our target system, specifically when interacting with the SLURM [SLU08] job manager. Hence, MPICH2 is currently the default MPI implementation, although the user is still allowed to request OpenMPI.
- FFTW [FFT12]: this library provides highly tuned Fast Fourier Transform (FFT) implementations.
- ATLAS [ATL12]: is an open source implementation of the Basic Linear Algebra Subprograms (BLAS) interface. ATLAS also provides some routines from the Linear Algebra PACKage).

Currently, scientific libraries are being executed in the ARM cores. However, we expect the MPSoC in the Mont-Blanc prototype to include a compute accelerator which, most likely, will have an ISA and architecture quite different from regular ARM cores. After the Mont-Blanc prototype MPSoC is selected by WP7, we plan to port FFTW and ATLAS to be executed by the compute accelerator. Depending on the target accelerator architecture, this work might involve the porting of existing libraries (e.g., CUBLAS and CUFFT), or the development of new libraries from scratch.

A potential porting work is the porting of an MPI library, or development of MPI call wrappers to allow the code running in the compute accelerators to perform inter-node communication. However, the development of this work highly depends on the profiling and analysis of the applications developed in WP4; only if MPI calls used within computation kernels introduce large overheads we foresee the need of in-accelerator MPI support.

3 Performance Analysis and Tuning of Node System Software

After deploying the Tibidabo prototype, access to the Mont-Blanc partners was granted for them to start developing and running the Mont-Blanc applications and kernels. Based on the initial feedback from WP3 and WP4 two major performance issues were identified by the users: floating-point operations and MPI communication. Based on this feedback, we have analysed these two components and start exploring potential tuning approaches to alleviate the current performance issues. This section describes this analysis and tuning work.

3.1 Performance of Floating Point Applications

The older ARM cores may not have floating point unit, so they require floating point support to be implemented in software. Since target for ARM is relatively new in Gnu Compiler Collection (unlike x86 which was present from the beginning), compiler always needs to assume the worst-case scenario and to compile for software floating point in default configuration. However, this problem can be resolved easily using correct compiler flags to indicate the proper version of the ARM architecture, and to instruct the compiler to use available hardware units for floating point operations. If these options are not passed correctly, users can see a 30x performance degradation.

Another problem that is related to floating point computations can not be resolved as easily. Due to the absence of floating point functional units and registers, all legacy code was compiled to use floating point emulation, and the so called *soft-float ABI (Application Binary Interface)*. With this ABI, on each function call, all floating-point arguments are passed in integer registers, requiring values to be first moved to integer registers and then moved back to floating-point registers inside the body of the function. This mode is still in use to support older ARM CPUs that do not have floating point registers. Note that, even when soft-float ABI is used, compiler can still use hardware to perform operations.

On the other hand, *hard-float ABI* uses floating point registers to pass arguments to functions, and eliminates the overhead of moving data to and from integer registers. Problem is that these two modes are not compatible, and a program compiled using *hard-float ABI* can not execute on an OS (and a corresponding set of libraries) that was compiled using *soft-float ABI*, and to ensure compatibility with older ARM cores all binary distributions of the Linux OS are compiled with *soft-float ABI*.

Currently, in our prototype we must compile all applications using *soft-float ABI* to match the underlying OS. This overhead affects applications that have many function calls with floating-point arguments[1]. This problem can be resolved, but it takes time, and there are indications that Debian GNU/Linux OS will be released for *hard-float ABI* for ARM targets. Expected improvement in performance is around 30%.

3.2 Performance of the MPI Library

The performance of MPI library is crucial for the overall performance of scientific applications in the Mont-Blanc project. OpenMPI had already been ported to ARM architecture by the time the Mont-Blanc project started so our objective in the project is to analyse and tune MPI performance.

[1]Most of the scientific applications that are of our interest use floating point operations heavily

3.2.1 Performance Measurement Methodology

As the final prototype system will only be available at a later phase of the project, we have been focusing first on the most basic performance metrics of the MPI library namely, point-to-point nearest neighbour latency and bandwidth. More complex metrics like performance of various collective communication operations will be analysed in detail and optimised if necessary when the final prototype configuration is known since they depend heavily on the actual network hardware and topology. Furthermore, unless the network hardware provides native support for such collective communications, their implementation rely on the point-to-point routines so optimising point-to-point latency and bandwidth is usually an important initial optimisation step for the collectives, too. Nevertheless, we also have run the Intel MPI Benchmark suite - that includes a comprehensive set of collective benchmarks, too - to get some baseline numbers (see later in Section 3.2.3).

The PingPong Microbenchmark To measure point-to-point latency and bandwidth we have used the usual pingpong microbenchmark approach. It measures the 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.

In addition to the pingpong benchmark that comes with the IMB suite, we have also implemented our own custom version because we wanted to use the same microbenchmark to evaluate different messaging APIs (e.g., MPI, TCP/IP, Open-MX, etc.) This is useful for instance, if we want to measure how much the underlying TCP/IP stack contributes to the overall MPI latency. Parameters of the measurements 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.

Test hardware There are three systems that we have used for MPI performance measurements so far. We have selected them based on availability. Each system contains a different processor implementation. The ARM processors chosen are Texas Instruments OMAP4430 and NVIDIA Tegra 2, with an Intel processor for comparison.

Both ARM processors use a dual-core Cortex A9 design running at 1GHz with 32 Kb of L1 cache (separate data and instruction caches) and 1 MB of L2 cache. The OMAP platform is a widely available PandaBoard, featuring an ARM NEON data engine and 1 GB RAM in the form of a ULPDDR2 Package on Package (POP). The Tegra platform - a SECO board - is based on a SECOQ7-MXM carrier board with a Quadmo747-X/T20 processing module. Table 1 shows the key configuration parameters for each of the Cortex A-9 chips.

Processor	Clock (GHz)	Cores	L1 Cache (I$/D$ KB)	L2 Cache (KB)	Memory (GB)	SP Peak (GFLOPS)
OMAP 4430	1	2	32/32	64	1	2
Tegra 2 T290	1	2	32/32	64	1	2

Table 1: Cortex-A9 Implementations used in performance measurements

The Intel system is a Lenovo Thinkcenter M91p. It features a quad-core Intel Core i7-2600 processor running at 3.3 GHz.

Measuring the Time We have started measuring execution time in our pingpong benchmark by means of the standard POSIX timer API, i.e.,`clock_gettime()`. However, we have realised that values showed a large random variations if the measured time was in the micro second or sub microsecond range especially, on the Pandaboard. The reason behind the variation turned out to be the low resolution external clocksource on the boards. The default clocksource frequencies and the corresponding effective clock resolutions are listed in Table 2.

	External clocksource frequency	Effective clock resolution
SECO board	1MHz	1 μs
Panda board	32kHz	31 μs

Table 2: Default clocksource frequencies and corresponding clock resolutions on the Panda and SECO boards

Interestingly, `clock_getres()` reports 1 nanosecond clock resolution on both Panda and SECO boards. Looking at the Linux kernel source we found out that the kernel will always report 1 nano second resolution by default if is built with the high resolution timers option enabled regardless of the actual clocksource frequency.

In order to enable very fine grain instrumentation and precise time measurements - which are necessary for instance, to break down shared memory latency overhead among the different messaging components - we have decided to rely on the ARM PMU (Performance Monitoring Unit) hardware cycle counter. A portable way to access hardware counters is to use the PAPI library ([PAP12]). However, PAPI is not always available. In particular, PAPI failed to build on our Panda boards. Moreover, PAPI has a relatively high overhead when very fine grain measurements are required.

To overcome these limitations, we implemented a light instrumentation API an library called `Timebase` that provides an arbitrary number of predefined software counters to measure elapsed times. These software counters can be individually activated/deactivated and they record automatically simple statistics including number of activations, total/minimum/maximum time measured. The activation/deactivation methods are inlined and very lightweight as they are used in the critical path. We also provided three different implementations for the `Timebase` API : direct ARM PMU access, PAPI and POSIX clock functions. In this way, we can run the same instrumented benchmark on different platforms and architectures. We note that the direct PMU access version of `Timebase` has 40 cycles overhead for each timer activate/deactivate event whereas the overhead in case of PAPI is around 1600 cycles on the SECO boards.

3.2.2 Baseline Latency and Bandwidth Performance Results

We measured latency and bandwidth using three different message transport paths within Open-MPI.

- **Shared memory**: MPI processes run on the same node and communicate via shared memory. Results are depicted in Fig. 2.
- **TCP/IP network**: This is the default MPI protocol over Ethernet networks. On the SECO boards (i.e. Tibidabo prototype) we measured performance both on the 100Mb and 1GigE networks. Results are depicted in Fig. 4.
- **TCP/IP loopback**: The loopback mode enables to run a TCP/IP test on a single node. It is useful to compare TCP/IP overhead of different platforms excluding the effect of the physical network layer. Results are depicted in Fig. 3

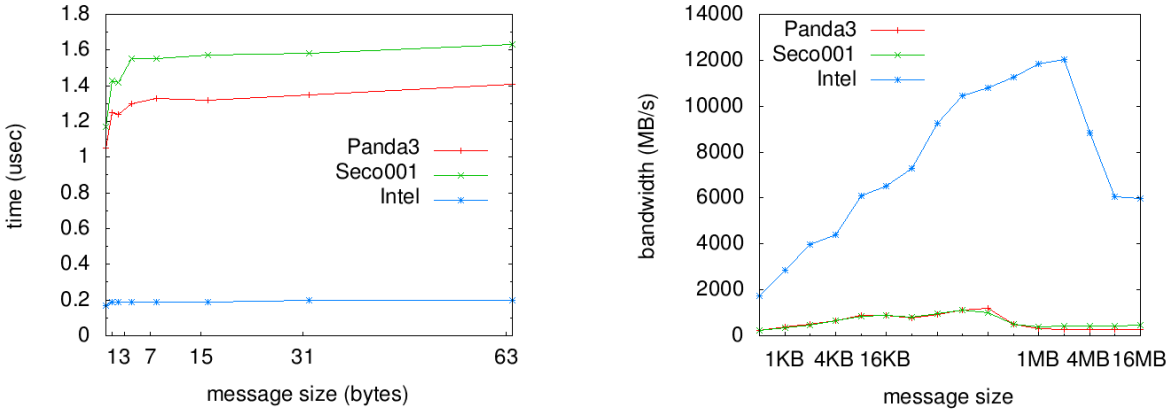

Figure 2: OpenMPI shared memory latency and bandwidth results

The plots show clearly that Intel outperforms the ARM systems in every tests which is not a surprise taking into account the significant architectural differences like CPU frequency. However, we have identified two areas where the difference is extreme : network latency and shared memory bandwidth. In Sect. 3.2.4, we summarise our preliminary results about optimising network latency. Our plans to improve shared memory bandwidth is described briefly in Sect. 3.2.5.

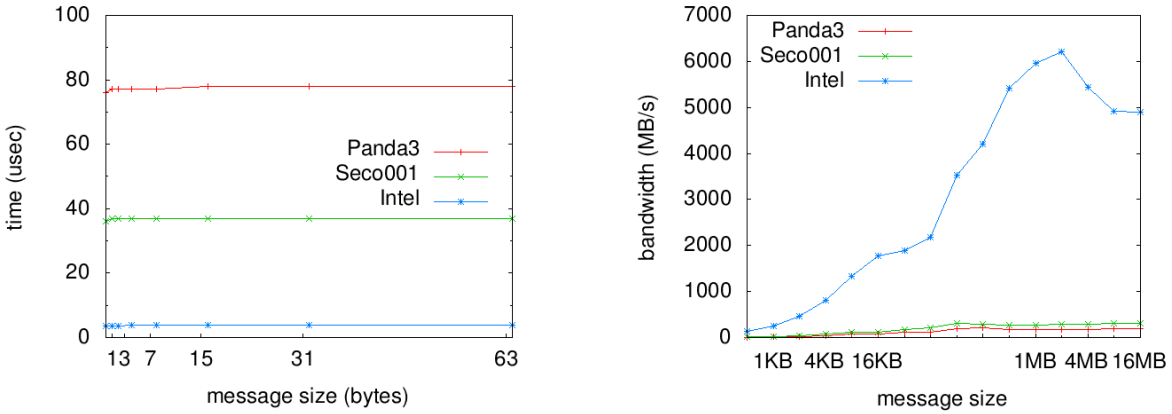

Figure 3: OpenMPI TCP/IP loopback latency and bandwidth results

3.2.3 Intel MPI Benchmark Suite Results

In addition to our customizable pingpong benchmark, we have also executed a commonly used benchmark suite, called Intel MPI Benchmark [IMB12]. This benchmark provides a battery of tests to assess the performance of both point-to-point and collective MPI primitives. In order to have a baseline for comparison, we have executed the same test on an Intel-based cluster too, with a similar network configuration (1GbE network cards and commodity switches).

We report the results from the "Exchange" benchmark that performs parallel transfers among a number of involved nodes (2 and 4 nodes in our case). The processes that are involved communicate in a chain-like fashion, sending and receiving data from both left and right neighbor varying message size from 0B to 4MB. The observed delays are reported in Table 3. In comparison with Intel-based cluster, Tibidabo shows larger delays with all messages sizes.

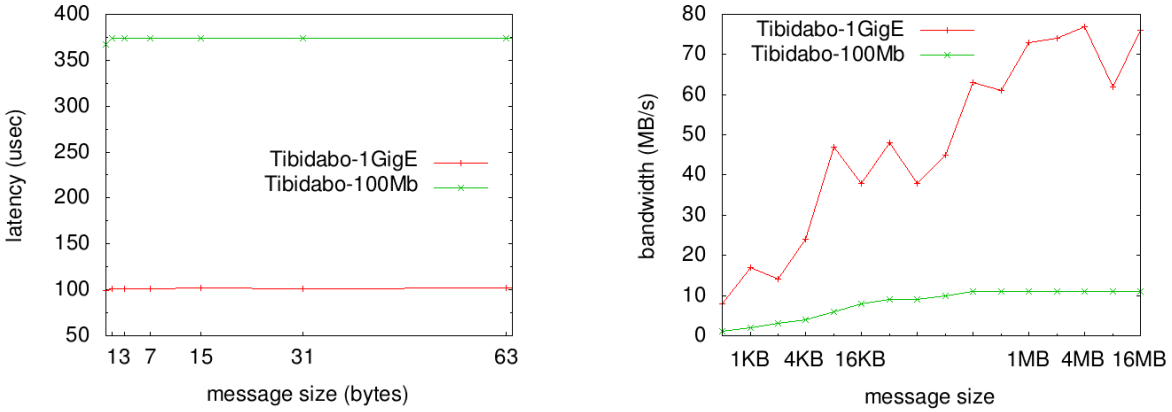

Figure 4: OpenMPI TCP/IP network latency and bandwidth results on Tibidabo using the 1GigE and the 100MB Ethernet connections

	Tibidabo		x86	
Packet Length	2 processes	4 processes	2 processes	4 processes
0	143.93	970.28	56.78	70.64
8	150.37	983.04	55.52	56.65
256	165.82	1255.24	81.32	54.4
1024	239.11	403.34	398.86	529.91
16384	865.75	26096.22	677.65	751
4194304	162495.7	787591.93	84536.85	72843.02

Table 3: Delay measured with "exchange" benchmark from *Intel MPI Benchmarks* suite. Benchmark is executed both on Tibidabo and on an x86 cluster with 1GbE network. All delays are in μs

Given that the MPI library depends on Ethernet drivers, and that both of them run in software then this result is expected given the obvious difference in processing power.

The biggest problem that we discovered so far concerns collective and parallel communication. Using Intel MPI benchmark, we discovered that current network that we have does not behave well neither with alltoall nor with parallel communication.

When only two MPI processes are involved in the exchange we observe delays starting from 150μs and bandwidth of around 100MB/s, which is near the peak bandwidth that our network can provide. However, when four MPI processes are involved delays unexpectedly increase to 970μ seconds and bandwidth drops to around 20MB/s. This behavior is present only on Tibidabo, and Intel-based cluster shows very little sensitivity to the number of involved MPI processes. As a consequence, applications that utilize all-to-all communication (or sending messages in parallel) do not scale well on our system. We have identified two major problems that lead to this type of behavior:

- *MPI Library* – The current implementation of MPI on ARM platforms utilizes TCP/IP stack in order to send and receive messages and involves costly system calls. This issues could be tackled with user-level MPI library that would use Ethernet protocol directly which should decrease delays. In fact, we already have some preliminary results using direct Ethernet with OpenMPI, see Sect. 3.2.4.

- *Working with unreliable hardware* – We have discovered that after a reboot of the cluster,

delays and bandwidth in parallel communication becomes much better (behavior similar to the one of the Intel cluster). Since the switches that we use are not optimized of HPC and for heavy communication, their performance starts to degrade when they are used for long time, and overall performance of the systems decreases. In case when commodity switches are used, monitoring software should be able to recognize this situation and either to report it, or to fix it (e.g. by rebooting the switches).

3.2.4 Improving network latency

We have measured around 100 μs end-to-end OpenMPI latency over the 1GigE network on Tibidabo (see Fig.4). This number is very high comparing to contemporary HPC systems. Even taking into account that 1GigE port is far from the latest network technology, there is likely a lot of room for performance improvement. We had the suspicion that the processing overhead of the messaging software stack and especially the TCP/IP stack was the main reason behind the high latency.

We note that SECO boards have two Ethernet ports : a 1GigE one (connected via PCIe) and a 100Mb one (connected via USB 2.0). We also measured the latency over the 100Mb ports of the SECO boards (i.e., Tibidabo) and got 350 μs. However, we focused on the 1GigE results since MPI library is supposed to use the 1GigE port for application messages while the 100Mb network can serve administrative and control system messages.

In addition to running the pingpong benchmark, we have conducted some further experiments to confirm that TCP/IP processing is indeed a major cause of high latency.

Bypassing OpenMPI In the first experiment, we have ported our handcrafted version of pingpong to use the TCP/IP socket API directly instead of calling OpenMPI functions. The aim was to prove that majority of the latency overhead does not come from the OpenMPI layer. Indeed, the TCP/IP version of pingpong delivered practically the same latency numbers as the original version proving that the bottleneck is definitely below OpenMPI in the messaging stack.

Lowering CPU Frequency In the second experiment, we lowered the CPU frequency on a 2-node SECO board cluster and rerun the pingpong benchmark. The aim was to show that the latency was compute bound. The A9 cores on the SECO boards run at 1GHz by default. In this experiment we lowered core frequency to 456MHz and got 144 us latency. That shows near linear relation between core frequency and latency : lowering frequency by 54% increases the latency by 52%, so latency seemed to be compute bound.

As a first attempt to eliminate the TCP/IP overhead we have looked at two existing direct Ethernet communication solutions : Open-MX ([Ope12b]) and Intel DET ([Int12]). Both of them provide basically a Linux kernel driver and a user space communication library to speed up communication over commodity Ethernet fabric. The Intel DET kernel driver turned out to be out of date and failed to build properly with the SECO board Linux kernel. Open-MX on the other hand seems to be a well supported on-going project and we could build the driver and the user space library without any major issue. Moreover, OpenMPI has built-in support for the Open-MX transport layer so the integration of the two system could be done smoothly, too. The first results we got are rather encouraging: our pingpong benchmark reported 65 μs latency which translates into a 32% improvement over the TCP/IP value. Fig. 5 depicts OpenMPI latency and bandwidth curves using TCP/IP and Open-MX message transport layers. Open-MX also substantially improved the bandwidth for large message sizes. For example, the bandwidth for 16MB messages is 80% higher. However, more thorough testing is needed

regarding robustness of Open-MX. Right now we can see sporadic Ethernet driver hangs when running Open-MX benchmarks.

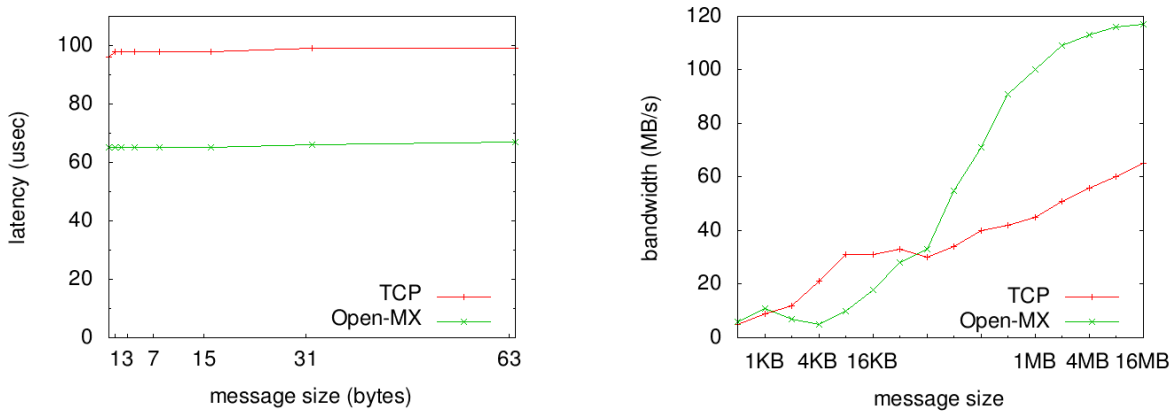

Figure 5: OpenMPI latency and bandwidth over the 1GigE Ethernet connection using TCP/IP and Open-MX message transport layers

3.2.5 Future plans

In the near future, we will continue the network latency optimisation work and focus on testing robustness and scalability of Open-MX.

We also plan to start working on the shared memory bandwidth optimisations. For large messages, bandwidth can be increased by kernel assisted memory copy where the kernel maps the source buffer into the address space of the target process temporarily so the data can be copied from source buffer to the target directly. An potential solution to look at is KNEM [KNE12].

As soon as the final prototype and network hardware is selected and available, we will start analysing and if necessary, optimising various MPI collective operations. We will focus on those that are more critical for the performance of Mont-Blanc applications.

4 Cluster System Software

The Mont-Blanc project is based on low per-core power consumption at the cost of reduced per-core performance. To compensate for the performance loss, higher degrees of parallelism are necessary. In addition to the increased absolute core count, current ARM based System-on-chips provide only up to four CPU cores. Thus, despite the potential of dense physical packaging, the number of logical compute nodes will be much higher than in comparable x86 based clusters. Since Cluster Management tools typically operate on logical nodes, scalability of the Cluster Management tools is crucial for Mont-Blanc. Cluster Management in high performance computing comprises of two aspects:

- **System Monitoring** software is used to ensure proper functioning of all cluster components. It is used to monitor both hardware and software components in clusters and notifies system administrators in case of detected anomalies. It is therefore the major point of interaction between the system administrators and the cluster system.
- **Resource Management** software, in return, is the interface between the system's users and the system. It keeps track of all available computing resources in the cluster and manages a queue of users' requests for the resources. Once enough resources are available to run a compute job in the queue, the resource management software allocates the required resources for the user and executes the compute job.

4.1 Tibidabo Job Scheduler and Monitoring Software

On Tibidabo, we have installed the SLURM [SLU08] system to manage MPI jobs in the cluster. It is an open-source resource manager designed for Linux clusters of all sizes, and it has been compiled and installed on Tibidabo since its portable C implementation made it a good candidate for Tibidabo's resource manager. SLURM has many modules that can be configured and installed, and we currently use the following:

- Resource limits management – to ensure that jobs do not try to use more resources than those that are available.
- Multiple job queues with support for Quality of Service (QoS) – to enable different types of users to have different priority of jobs (e.g., queues for benchmarking, regular queues and superuser queues).
- Job accounting on per user basis – to keep track of resources consumed by users.
- Per job power consumption job epilogue – this module provides samples taken from power meter during the execution of jobs. Samples are given every 5 seconds for the entire machine, and they have granularity of 0.1kW. Samples are organized in such a way that a user can easily calculate the required energy for the job.
- Cooling-intensity-aware per job node scheduling – Scheduler is capable of avoiding nodes in zones of the cluster that tend to overheat.

The Ganglia [Gan00] system monitoring has been also installed from sources. Ganglia is a distributed system monitor for clusters, and it allows for remote view of statistics (either in real-time, or values from history). The system is highly customizable, and its current installation on Tibidabo supports the following statistics:

- Global memory usage
- Power consumption
- NFS Server load
- NFS Server memory usage
- Node CPU usage
- Job distribution per CPU (distributing jobs according to number of requested CPUs)
- Filesystem and communication network utilization

4.2 Concepts for Energy-optimal System Operation

In the area of resource management systems, today's systems already support high performance computers with millions of CPU cores. However, for improved energy efficiency of future supercomputers, such as the final Mont-Blanc prototype system, resource management systems could still be improved. As of today, high performance resources are mostly accounted for using CPU hours. With energy costs making up an increasing fraction of the total cost of ownership of supercomputers, accounting for resources based on consumed energy becomes a viable alternative. Current resource management systems, however, lack the support for this.

In order to realise this feature it is first necessary to determine a mechanism to predict the energy/power consumption associated with the execution of a job. This would require to jointly monitor the energy consumed at node level (e.g., energy usage of CPU, memory, etc.) and at rack level (e.g., energy usage for cooling, power supply, etc.); hence, on the basis of these measurements, it is possible to estimate the energy-to-solution of a job and consequently define a scheduling strategy based on the job energy/power consumption. Current technology provides several ways of accessing and extracting energy-related information for different architectures, such as the Running Average Power Limit interface counters for Intel Sandybridge chips [int09] or the "Processor Power in TDP" MSR for the AMD Family 15h processors [amd11]. A viable solution for the ARM-based Mont-Blanc prototype architecture could be the PAPI energy and power readings [PAP12]. PAPI has the advantage of being platform-independent, providing a common user interface for various performance metrics; however, since it may suffer from fine-grained measurement overhead as briefly discussed in Section 3.2.1, the exact impact of its usage has to be quantified yet. Once this information is available, the resource management system (e.g., SLURM) should be accordingly modified in order to support energy-based accounting schemes.

Also, modern hardware supports a variety of power saving operation modes. In high performance computing, where performance was prioritized higher than energy efficiency for a long time, such low-power modes are not widely used. Yet, the resource management software is the ideal component to include support for the different hardware power-saving modes. This way, idle nodes can be powered off and through the use of dynamic voltage and frequency scaling, applications that are incapable of exploiting high processor frequencies can be run on lower CPU frequencies.

The simplest implementation of aforementioned mechanism can be interpreted in the following general way: if an idle node does not receive a job submission request within a pre-specified interval of time, then it can enter suspension mode or be completely powered off in order to save energy. Upon receival of a new job submission request, the system re-enters normal operational mode. Furthermore, dynamic voltage and frequency scaling can be used for codes that can

not fully exploit high system processor frequencies by running them at lower voltage and CPU frequencies [DVF11]. In this way, considerable energy savings can be potentially achieved as demonstrated in several research works such as [hHcF05] and [ECLV10].

The adoption of these energy-aware mechanisms becomes particularly relevant when moving towards the Exascale. As illustrated from preliminary results included in the Mont-Blanc Project deliverable document 3.1 [D3111], the overall performance of codes (e.g., Flops/sec, time-to-completion of a job, etc.) may not scale well when increasing the number of cores of the system. This behaviour can already be clearly observed for a relatively small number of cores (e.g., note the scalability of `BigDFT` at Figure 7 of [D3111]) and it predictably gets worse when incrementing the number of nodes, even when using backfill scheduling solutions. Therefore, it may be beneficial to limit the number of nodes requested by these kind of jobs in order to reduce the otherwise inevitable waste of computing resources. This means that in an Exascale system, in addition to "large" jobs requiring a considerable amount of resources to run (e.g., in the order of hundreds or thousands of nodes), we could further expect the presence of a high number of "small" jobs associated with fewer computing nodes, particularly due to their poor performance in terms of scalability. As a direct consequence, this may have an impact on the energy consumption, as illustrated in the example of Figure 6.

The graph shows the number of busy nodes of a high performance computing system over time. In case 1, the execution of a large job (job A) is terminated at time t, freeing at once a high number of computing nodes (in the example 500 nodes) that can be immediately reserved for running a second large job (job B). On the other hand, in case 2, computing resources are gradually released due to multiple terminations of "small" jobs. In this way, we observe a lower resource utilization (represented by the area delimited between the curves of case 1 and 2) with a consequent waste of energy determined by nodes being in an idle state and not being assigned to any job. Hence, this occurrence may further justify the need of the above-discussed power saving mode strategies in order to limit the energy consumption of an Exascale high performance computing system.

SLURM [SLU08] implements power saving mode strategies with an integrated mechanism that allows to slow down the performance of computing nodes or completely power them off. The CPU underclocking is achieved by using a `cpufreq` governor (that has to be enabled by appropriately configuring the Linux kernel) which can modify the CPU voltage and frequency;

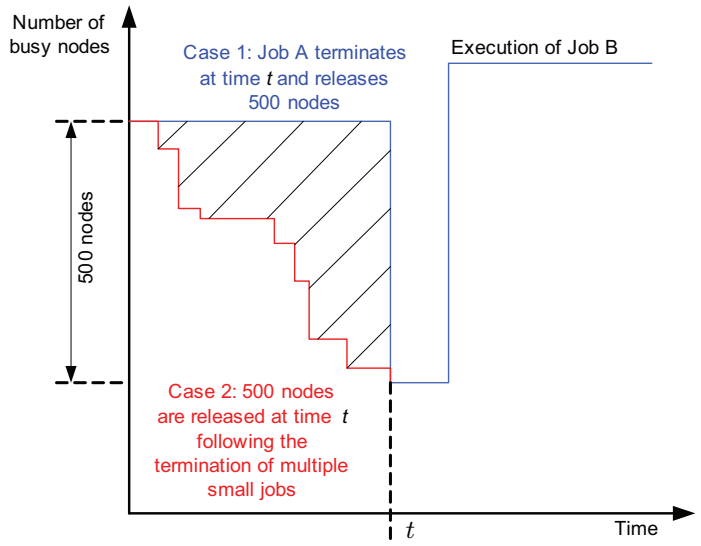

Figure 6: Example of energy consumption due to multiple computing nodes in idle state.

however, note that the DVFS feature is not integrated directly in SLURM. In this regard, we believe efforts should be made in order to include it in the resource management tool for the intended objectives of the project. SLURM further provides a comprehensive set of parameters to efficiently manage the sleep/awake state of computing nodes. Specifically, the SLURM administrator can specify values such as:

- `SuspendTime`, that is the interval of time after which an idle node is allowed to enter power saving mode;
- `SuspendRate`, which represents the maximum number of idle nodes that can be put into a sleep state per minute;
- `ResumeRate`, that is the maximum number of nodes that can re-enter normal operational mode per minute.

The parameters `SuspendRate` and `ResumeRate` are intended to respectively avoid drastic drops and raises of energy consumption. On top of these essential configuration settings, the SLURM administrator can additionally define his own programs to trigger the sleep state of a computing node. The execution of these programs is performed by configuring the dedicated `SuspendProgram` and `ResumeProgram` parameters. SLURM offers a variety of other configuration settings that allow to further refine the suspension/resume mode of computing nodes. We are currently analyzing all the features of the power saving mechanism supported by SLURM in order to better quantify the energy savings achieved when powering down idle nodes of the cluster.

4.3 Scalability of Monitoring Software

New features such as energy-to-solution based accounting and other energy-efficiency related tuning efforts rely on the availability of fine-grained sensor data. In systems with tighter integration of data centre infrastructures, e.g. when waste heat is reused, infrastructure data (temperatures, flow rates, humidity, etc.) also needs to be monitored.

Traditionally, two approaches exist to store sensor data:

- `Round-robin database files` use pre-allocated disk space to store a predefined number of measurements. Automatic aggregation, such as computing the average of older values, helps reducing the required disk space. Due to powerful visualization features, round-robin database files are often used as the basis for system monitoring solutions such as Ganglia.
- `Relational database management systems` such as MySQL, PostgreSQL, etc. can also be used to store sensor data. Their advantage is the ability to correlate different types of data easily. Thus, relational database management systems are ideal candidates for the relation of application performance data to power consumption for power modelling purposes or the assessment of the energy-to-solution of applications.

Although well established in current high performance computing installations, both solutions will likely struggle to scale to needs of future Exascale systems. With their pre-allocated files, round robin database systems restrict flexibility in the measurement granularity. In return, relational database management systems struggle with their attempt to ensure atomicity during high data insertion rates. Also, relational database management systems are difficult to design if scalability is a key requirement.

Through their use in large-scale web applications, a new type of database systems has recently gained popularity. These systems are often referred to as NoSQL database systems as they typically do not provide a standard interface based on the Simple Query Language (SQL). A specific implementation of NoSQL database systems are key-value stores where every database entry is only accessible through a single primary key. Based on this key, a hashing function can be used to distribute each database entry to multiple database backend servers. This hash can be computed in constant time, meaning that such database systems scale to higher speed and capacity by simply adding more database backend servers.

We have started to test different existing implementations of NoSQL database systems with respect to their suitability to be used as system monitoring database. The main criteria for the assessment are easy scalability, high insert rates, and efficient data retrieval for a given time span. Based on this evaluation, a system monitoring solution will be implemented and tested on both, artificial test data, as well as existing real clusters.

References

[All12] Allinea DDT - the debugging tool for parallel computing. http://www.allinea.com/products/ddt/, 2012.

[amd11] Amd family 15h processor bios and kernel developer guide. *AMD*, 2011.

[ATL12] Automatically Tuned Linear Algebra Software (ATLAS). http://math-atlas.sourceforge.net/, 2012.

[D3111] Kernel selection criteria and assessment report. *Mont-blanc Project Deliverable Document 3.1*, 2011.

[DVF11] Dynamic Voltage and Frequency Scaling. http://en.wikipedia.org/wiki/Voltage_and_frequency_scaling, 2011.

[ECLV10] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Optimizing job performance under a given power constraint in hpc centers. In *Green Computing Conference, 2010 International*, pages 257 –267, aug. 2010.

[FFT12] FFTW Home Page. http://www.fftw.org/, 2012.

[Gan00] Ganglia Monitoring System. http://ganglia.sourceforge.net/, 2000.

[GCC12] GCC, the GNU Compiler Collection. http://gcc.gnu.org/, 2012.

[GDB12] GDB: The GNU Project Debugger. http://www.gnu.org/software/gdb/, 2012.

[hHcF05] Chung hsing Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 1, nov. 2005.

[IMB12] Intel MPI Benchmarks 3.2.3. http://software.intel.com/en-us/articles/intel-mpi-benchmarks/, 2012.

[int09] Intel architecture software developers manual, volume 3: System programming guide. *Intel*, 2009.

[Int12] Intel Direct Ethernet Transport. http://software.intel.com/en-us/articles/intel-direct-ethernet-transport/, 2012.

[Ker12] The Linux Kernel Archives. http://www.kernel.org/, 2012.

[KNE12] High-Performance Intra-Node MPI Communication. http://runtime.bordeaux.inria.fr/knem/, 2012.

[MPI12] MPICH2. http://www.mcs.anl.gov/research/projects/mpich2/, 2012.

[Ope12a] Open MPI: Open Source High Performance Computing. http://www.open-mpi.org/, 2012.

[Ope12b] Myrinet Express over Generic Ethernet Hardware. http://open-mx.gforge.inria.fr/, 2012.

[PAP12] Performance Application Programming Interface. http://icl.cs.utk.edu/papi/, 2012.

[PRA12] PRACE-1IP Details. http://www.prace-project.eu/PRACE-First-Implementation-Phase, 2012.

[SLU08] Simple Linux Utility for Resource Management. https://computing.llnl.gov/linux/slurm/slurm.html, 2008.

[Teg12] Tegra2 and Tegra 3 Superprocessors. http://www.nvidia.com/object/tegra-superchip.html, 2012.

[Ubu10] Ubuntu 10.10 (Maverick Meerkat). http://releases.ubuntu.com/10.10/, 2010.