D4.4– Intermediate report on Linux customization for HPC
Version 1.0

Document Information

Contract Number	610402
Project Website	www.montblanc-project.eu
Contractual Deadline	M24
Dissemination Level	PU
Nature	R
Authors	Chris Adeniyi-Jones (ARM)
Contributors	Roxana Rusitoru (ARM), Dani Ruiz (BSC), Nikola Rajovic (BSC), Alejandro Rico (BSC), Filippo Mantovani (BSC). Jesus Labarta (BSC)
Reviewers	Maria Del Carmen Martinez Fernandez (UNICAN), José Luis Bosque Orero(UNICAN)
Keywords	Linux kernel, Scheduling, MPI

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 610402.*

Change Log

Version	Description of Change
v0.1	Initial version of the deliverable
v0.2	Collated version for internal review
v0.3	Incorporated internal review comments
v1.0	Final version for submission

Contents

Executive Summary

This deliverable report describes some of the work that has been done on optimizing the Linux operating system for running HPC applications on ARM. The first section describes problems found in the interconnect hardware/software stack and potential solutions. The second section describes some profiling infrastructure that will be used when looking at how to implement energy-aware scheduling policies at the kernel level.

1 Experiences Configuring the Network of an ARM-based HPC Cluster

The current fastest High Performance Computing (HPC) clusters utilize high bandwidth and low latency networks such as Infiniband. Nevertheless, the commodity solution for intercommunication among compute nodes in HPC clusters is still Ethernet networks. Therefore, analyzing the behaviour of this kind of network is of huge interest to allow the modification of the protocols used to improve the performance.

This summary describes a study of an ARM-based HPC cluster network and the optimizations we applied to better utilize the available hardware. We show performance and scalability results to better understand the impact of our modifications on the performance of scientific applications.

1.1 Motivation

The vision of the EU project "Mont-Blanc", since 2011, is to leverage the fast growing market of mobile technology for scientific computation, HPC and non-HPC workload [1]. The main objective, reached at the beginning of 2015, has been the deployment of a cost-effective prototype based on low-power embedded mobile SoCs. Commercial availability (thus maximising design time and minimizing risk) was a very important criterion when deciding which SoC to use to build the Mont-Blanc prototype. The chosen SoC does not have an Ethernet interface but does have a USB3.0 interface. We use this USB3.0 interface to connect the nodes in the Mont-Blanc prototype with low-end commodity Ethernet. This is implemented using one USB3.0-to-Ethernet (1GigE) chip per SoC. With this 1GigE connection each of the fifteen nodes in the blade is then connected to a blade-level switch. This brings inefficiencies on both the hardware and software sides as communication is going through several layers of hardware interface, drivers and system software stack.

1.2 Experimental setup

1.2.1 Hardware platform

We conduct experiments/evaluation/tuning on the Mont-Blanc prototype cluster [2]. The cluster has 1080 nodes powered by Samsung Exynos 5 mobile SoC [3] which integrates ARM Cortex-A15 at 1.7GHz in a dual-core configuration. Nodes are equipped with 4GB of LPDDR3-1600 RAM memory, and are interconnected with an Ethernet network. Each node has a peak link bandwidth of 1Gbps, provided through a USB3.0-to-Ethernet interface controller. In reality the latency is roughly two orders of magnitude greater and the throughput between one and two orders of magnitude lower than what a contemporary HPC system achieves when using 10GigE Ethernet.

1.2.2 Applications

For this study we employ two well known and established proxy applications used in hardware/software co-design, namely LULESH [4]. LULESH is a proxy application representative of simplified 3D Lagrangian hydrodynamics on an unstructured mesh. It performs a hydrodynamics stencil calculation using both MPI and OpenMP to achieve parallelism. Regarding

communication time, LULESH spends most of the time in MPI_Allreduce. LULESH was compiled with GCC 4.9.2 and OpenMPI 1.8.3, and run on the prototype cluster without particular source code optimizations/modifications.

1.3 Experiences

After performing both weak and strong scaling executions, we noticed that strong scalability was not as expected for LULESH. Looking at the execution traces obtained with EXTRAE [6] we found that some of the MPI processes experienced long timeouts, which introduced load imbalance between processes. After investigating this behaviour we discovered those issues were caused by the TCP protocol and the change of the communication protocol within the MPI implementation, this is, the eager limit. Both parameters were tuned to improve the application performance. In our study, we tweak two TCP configuration parameters: minimum retransmission timeout and selective acknowledgment.

1.3.1 Retransmission TimeOut

Retransmission TimeOut (RTO) refers to the time the system must wait before retransmitting a lost TCP packet. RTO is computed heuristically based on previous transmissions and has a minimum value (RTOmin) the user can set as a parameter. The default RTOmin value is 200 milliseconds. This value is thought for wide area networks but is large for a low latency HPC system.
We performed measurements running LULESH with the default RTOmin value and with the lowest possible RTOmin value (5 ms) allowed by the system. Figure 1 shows the behaviour in the two different cases: 200 ms (top) and 5 ms (bottom).

We can see the MPI calls duration in one iteration of LULESH with different colors. Red indicates that the MPI process is executing an MPI_Wait call, green is for the MPI_Waitall primitive, pink for MPI_Allreduce and blue means computation.
Load imbalance is visible when using an RTOmin of 200 milliseconds. Some MPI processes spend most part of its execution time at the MPI_Wait function, delaying other MPI processes due to the synchronization inherent to the application. Analyzing the call duration in the two tests, we discovered that some MPI processes spent up to 62,34% of its execution time inside MPI_Waitall. After modifying the RTOmin value to 5 milliseconds, the worst case is a 19,17%, resulting in an overall performance improvement of 3,25x.

1.3.2 Selective acknowledgment

Selective Acknowledgment (SACK) is a Linux kernel TCP parameter that can be configured. It is used to retransmit specific TCP segments out of order when one segment is lost during the communication process. Potentially, it may lead to fewer retransmissions, but, since this feature requires more information at the TCP header, each MPI process would need more segmentation when sending a big packet than needed when SACK is disabled. This creates an overhead that, in our case, is not desired because the more TCP packets we send the more we could lose, and every lost packet creates a big delay due to the RTO value.
Additionally, since the architecture we use in our cluster does not provide a network offload hardware, the generation of the TCP headers is done by the CPU, so the smaller the TCP header the less the time spent on network operations (and not compute).

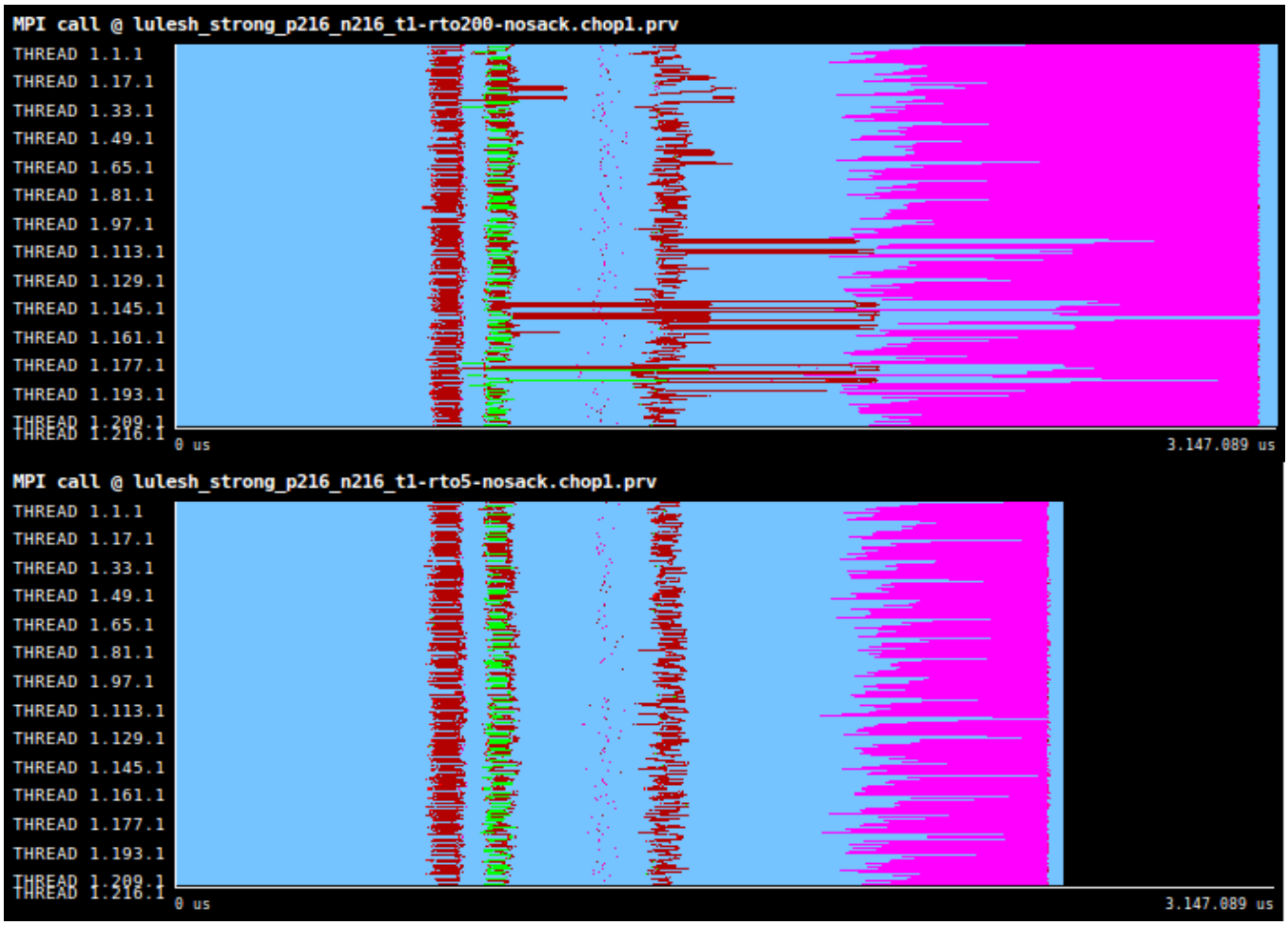

Figure 1: LULESH timeline (one iteration) running on 216 MPI processes for different RTOmin values: 200 ms (top), 5 ms (bottom). Light blue indicates computation periods and other colors show time spent in MPI calls.

1.3.3 MPI

Looking at the applications traces, we noticed as well that most of the communication utilizes a buffer size bigger than the eager size limit value. This means most of the communications use rendezvous protocol. Using rendezvous protocol, each sending process needs to synchronize with the receiver before sending the data. Using eager protocol, which is used when the buffer size is under the value of the eager limit, each send does not need a matching receive before sending the data. This means that, we can avoid synchronization, potentially decreasing the time spent at MPI_Wait and MPI_Waitall primitives. For LULESH, the percentage of the execution time per iteration spent on the MPI_Wait and the MPI_Waitall primitives is 96% and 62,43% for some of the MPI processes, causing a load imbalance for the application. After increasing the eager value limit from 32 KB (default) to 128 KB, the new percentages are 74,58% and 0,65%. This translates into an speedup of 1,28x and 96,05x, and greatly reduced load imbalance between MPI processes.

1.4 Conclusions

Our study demonstrates that modifications of the communication protocols can lead to significant improvements especially in network-bound applications. In our experience, we managed to almost double the performance of some of the applications executed on the Mont-Blanc prototype cluster. We note that the parameters tweaked in our ARM-based cluster may not have the same effects on other clusters with an Ethernet-based network but based on other architectures. For example, we would expect to see less improvement in systems where a significant amount of network processing is being offloaded to specialized hardware within the node.

2 Automatic migration reporting

ARM's big.LITTLE technology is a heterogeneous processing architecture which uses two types of processor. LITTLE processors are designed for maximum power efficiency while big processors are designed to provide maximum compute performance. Both types of processor are coherent and share the same instruction set architecture (ISA). Using big.LITTLE technology, each process can be dynamically allocated to a big or LITTLE core depending on the instantaneous performance requirement of that process. Through this combination, big.LITTLE technology provides a solution that is capable of delivering the high peak performance demanded by parts of moder¡n applications, within the thermal bounds of the system, with maximum energy efficiency. The Linux kernel process scheduler is aware of the differences in compute capability capacity between big and LITTLE cores. Using statistical data and other heuristics, the scheduler tracks the performance requirement for each individual process, and uses that information to decide dynamically which type of processor to use for each process, migrating the process between processors as necessary.

Automatic migration reporting is a lightweight feature added to the Linux kernel that enables the kernel to report on process migration events. For each migration, a source core, destination core, timestamp (in jiffies) and reason are logged. This information will be used to gain insight into when and why the scheduler migrates processes and to assess the correctness and quality of novel scheduling algorithms.

We can also use this information to understand the profile of an application in terms of it's computation, memory and networking requirements. If an application gets migrated onto smaller cores, there is a possibility that it is not as computationally or load intensive as initially believed.

2.1 Motivation

We were motivated to implement this method of capturing process migration behaviour because want to to use a micro-benchmark designed to help us evaluate different policy options when scheduling for big.LITTLE systems (work which is being done in Mont-Blanc 2 WP4). It has several discrete phases that we can use to measure how compute intensive and memory intensive activities run on different core types. Our intention is to observe the interaction between Heterogeneous Multi-Processing (HMP) scheduler policies and the different phases. In addition, we aim to expose different power/performance points with each phase. We will use the following metrics to try and characterize each phase/core type combination: duration, energy to solution, Instructions-per-cycle (IPC), GIPS and GIPS/W (giga-instructions per second, per watt), L1 data cache miss rate, L2 data cache miss rate and data memory accesses. We can control at runtime which phases of the micro-benchmark to run and we can also control the particular set of cores that are available for the scheduler to schedule processes onto.

2.2 Development

The automatic migration reporting was implemented as patches to Linux kernel version 3.15-rc8, at the time of implementation, the officially supported kernel for the Juno ARM Development Platform [8].

There are several ways in which the migration reporting can be enabled. This can be done globally for all process or locally for a single process:

Table 1: Enabling migration reporting

Scope	Source Code Change Required	Implementation
Per-process	Yes	System call
Global	No	sysfs
Per-process	No	procfs via PID

To control the amount of information generated, it is possible to limit the number of migrations that are reported in the output file. This can be done when the kernel is compiled via a kernel configuration parameter or at runtime via sysfs or procfs.

The kernel configuration parameter `CONFIG_SCHEDSTATS` must also be enabled for the migration information to be readable along with the other scheduler statistics. The migration information can be read from `/proc/<PID>/migrations`.

By default, all processes have migration reporting turned off. In addition, child processes inherit the migration reporting status of their parent process.For example, if we enable migration reporting for an application that runs its compute kernels using OpenMP, all the OpenMP threads will have migration reporting enabled upon spawn. This makes it possible to track the migration of the OpenMP processes separately from the parent process.

We show below some example output from running the micro-benchmark with migration reporting enabled:

```
bL_migration_te (3631, #threads: 4)
---------------------------------------------------------------------
se.exec_start                                :       2517511.801480
se.vruntime                                  :        259193.507485
se.sum_exec_runtime                          :         72299.799220
se.nr_migrations                             :                   10
se.statistics.nr_migrations_cold             :                    0
se.statistics.nr_failed_migrations_affine    :                  103
se.statistics.nr_failed_migrations_running   :                   29
se.statistics.nr_failed_migrations_hot       :                   23
se.statistics.nr_forced_migrations           :                    0
migration from 2 to 0 at 4295187232 due to task wake-up
migration from 1 to 2 at 4295185556 due to load balancing
migration from 0 to 1 at 4295185473 due to task wake-up
migration from 1 to 0 at 4295185256 due to load balancing
migration from 2 to 1 at 4295184766 due to load balancing
migration from 1 to 2 at 4295182977 due to task wake-up
migration from 0 to 1 at 4295182110 due to task wake-up
migration from 3 to 0 at 4295181758 due to load balancing
migration from 1 to 3 at 4295181630 due to load balancing
migration from 0 to 1 at 4295181629 due to new task wake-up
policy                                       :                    0
prio                                         :                  120
```

2.3 Conclusion

This feature is now helping us analyze the interaction between different phases of our micro-benchmark, the Linux HMP process scheduler and the big.LITTLE cores on our ARM Juno Development systems. The output from this work will be reported in deliverable report D4.5.

References

[1] Nikola Rajovic Paul M. Carpenter Isaac Gelado Nikola Puzovic Alex Ramirez Mateo Valero. Supercomputing with commodity cpus: Are mobile socs ready for hpc? Technical report, 2013.

[2] The Mont-Blanc prototype. https://www.montblanc-project.eu/arm-based-platforms.

[3] Samsung Exynos. http://www.samsung.com/global/business/semiconductor/minisite/Exynos/w/solution.html#?v=dual.

[4] Ian Karlin et al. Lulesh 2.0 updates and changes. Technical report, 2013.

[5] CoMD Proxy Application. http://www.exmatex.org/comd.html.

[6] Extrae. http://www.bsc.es/computer-sciences/extrae.

[7] ARM big.LITTLE technology. http://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[8] Juno ARM Development Platform. http://www.arm.com/products/tools/development-boards/versatile-express/juno-arm-development-platform.php.