# Enabling the Use of Embedded and Mobile Technologies for High-Performance Computing

*Author:*
Nikola Rajović

*Advisor:*
Alex Ramirez


A THESIS SUBMITTED IN FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
**Doctor per la Universitat Politècnica de Catalunya**
Departament d'Arquitectura de Computadors

Barcelona, Spring 2017

*To my family ...*

*Mojoj porodici ...*

# Acknowledgements

During my PhD studies I received help and support from many people - it would be strange if I did not. Thus I would like to thank professors Mateo Valero and Veljko Milutinović for recognizing my interest in HPC and introducing me to my advisor, professor Alex Ramirez. I would like to thank him for trusting in me and giving me an opportunity to be a pioneer of HPC on mobile ARM platforms, for guiding and shaping my work last five years, for helping me understand what real priorities are, and for being pushy when it was really needed.

In addition, thank you Carlos and Puzo for helping me to have a smooth start of my PhD and for filling the gap when Alex was too busy.

Last two years of my PhD I have been working with Alex remotely, and I would like to thank the local crew who helped me: professor Eduard Ayguade for providing me with some very hard-to-get manuscripts and accepting to be my "ponente" at the University; professor Jesus Labarta for thorough discussions about parallel performance issues and know-how lessons on demand; Alex Rico for helping me deliver work for the Mont-Blanc project and for friendly advices every so little; Filippo Mantovani for making sure I could use the prototypes without outages and for filtering-out internal bureaucracy issues.

In addition, I would like to acknowledge support from BSC Tools department, especially Harald Servat and Judit Gimenez, who always managed to find a free slot for me in their busy agenda.

Gabriele Carteni and Lluis Vilanova shared with me their Linux Guru experience, which was extremely helpful during deployment of my first prototype - Tibidabo. Thank you guys! In addition, thank you Dani for bypassing ticketing system and fixing issues related to our internal prototypes as soon as I trigger them.

Thanks to prelectura committee and the constructive comments I received, the quality of my PhD thesis manuscript was significantly improved.

Professor Nacho Navarro used to cheer me up many times with his down-to-earth attitude, ideas and advices - I am sad that he cannot see me defending my thesis.

I shared my office with a lot of people, and I would like to thank them for all the good and bad times.

I would like to thank all my Serbian friends I lived, used to hang around, played basketball, or shared office with - thank you Браћала, Боки, Бране, Радуловача, Уги, Зоки, Зуки, Влајко, Мрџи, Павле, Вујке, Перо, Пузо ...

Thanks to Rajović headquarters and their unconditional support for my ideas, my PhD adventure finally ends with the writing of this Acknowledgement. Knowing you are with me helped a lot!

Last but not least I would like to thank my darling, my Ivana, for the patience, understanding and endless support. You are the true hero ... and thank you Uglješa, my son, for inspiring me with your smile ....

*Author*

My graduate work leading to these results was supported by the PRACE project (European Community funding under grants RI-261557 and RI-283493), Mont-Blanc project (European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement $n^o$ 288777), the Spanish Ministry of Science and Technology through *Computacion de Altas Prestaciones (CICYT) VI* (TIN2012-34557), and the Spanish Government through *Programa Severo Ochoa* (SEV-2011-0067).
Author has been financially supported throughout his studies with a grant from Barcelona Supercomputing Center.

# Abstract

In the late 1990s, powerful economic forces led to the adoption of commodity desktop processors in High-Performance Computing (HPC). This transformation has been so effective that the November 2016 TOP500 list is still dominated by x86 architecture. In 2016, the largest commodity market in computing is not PCs or servers, but mobile computing, comprising smartphones and tablets, most of which are built with ARM-based Systems on Chips (SoC). This suggests that once mobile SoCs deliver sufficient performance, mobile SoCs can help reduce the cost of HPC.

This thesis addresses this question in detail. We analyze the trend in mobile SoC performance, comparing it with the similar trend in the 1990s. Through development of real system prototypes and their performance analysis we assess the feasibility of building an HPC system based on mobile SoCs. Through simulation of the future mobile SoC, we identify the missing features and suggest improvements that would enable the use of future mobile SoCs in HPC environment. Thus, we present design guidelines for future generations mobile SoCs, and HPC systems built around them, enabling the new class of cheap supercomputers.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

In the domain of High-Performance Computing there is a continued need for higher computational performance. Scientific grand challenges in engineering, geophysics, bioinformatics, and other types of compute-intensive applications require increasing of computing capabilities of supercomputers in order to support growing complexity of problems and models. Over the time, there were different approaches in increasing the required level of performance due to new requirements, such as energy efficiency and economical market conditions.

## 1.1 Microprocessors in Supercomputing

During the early 1990s, the supercomputing landscape was dominated by special-purpose vector and Single Instruction Multiple Data (SIMD) architectures. Vendors such as Cray (vector, 41%), MasPar (SIMD,[1] 11%), and Convex/HP (vector, 5%)[2] designed and built their own HPC computer architectures for maximum performance on HPC applications. During the mid to late 1990s, microprocessors used in the workstations of the day, like DEC Alpha, SPARC and MIPS, began to take over high-performance computing. About ten years later, these RISC (Reduced Instruction Set Computing) CPUs (Central Processing Units) were, in turn, displaced by the x86 CISC

[1] SIMD: Single-Instruction Multiple Data

[2] All figures are for vendor system share in the June 1993 TOP500 list [122].

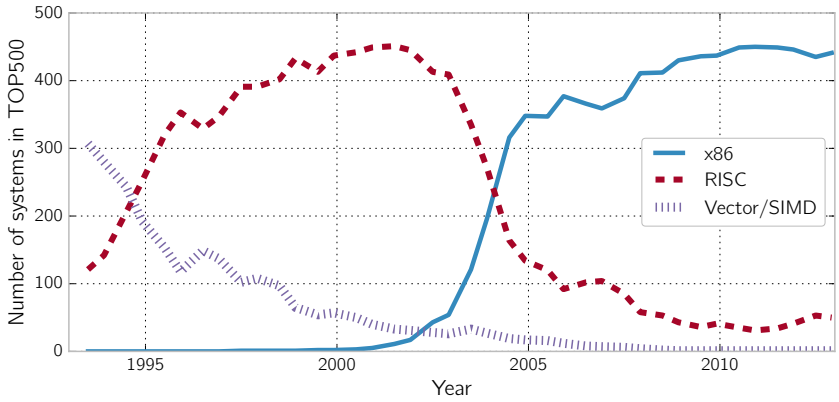



**Figure 1.1:** Development of CPU architectures share in supercomputers from TOP500 list. Special-purpose HPC replaced by RISC microprocessors, in turn displaced by x86. *Data source: TOP500*

(Complex Instruction Set Computing) architecture used in commodity PCs. Figure 1.1 shows how the number of systems, of each of these types, has evolved since the first publication of the TOP500 list in 1993 [122].

Building an HPC chip is very expensive in terms of research, design, verification, and creation of photo-masks. This cost needs to be amortized over the maximum number of units to minimize their final price. This is the reason for the trend in Figure 1.1. The highest-volume commodity market, which was until the mid-2000s the desktop market, tends to drive lower-volume higher-performance markets such as servers and HPC.

The above argument requires, of course, that lower-end commodity parts are able to attain a sufficient level of performance, connectivity and reliability. To shed some light on the timing of transitions in the HPC world, we look at the levels of CPU performance during the move from vector to commodity microprocessors. Figure 1.2 shows the peak floating point performance of HPC-class vector processors from Cray and NEC, compared with floating-point-capable commodity microprocessors. The chart shows that commodity microprocessors, targeted at personal computers, workstations, and servers were around ten times slower, for floating-point operations, than vector processors, in the period 1990 to 2000 as the transition in HPC from vector to microprocessors gathered pace.

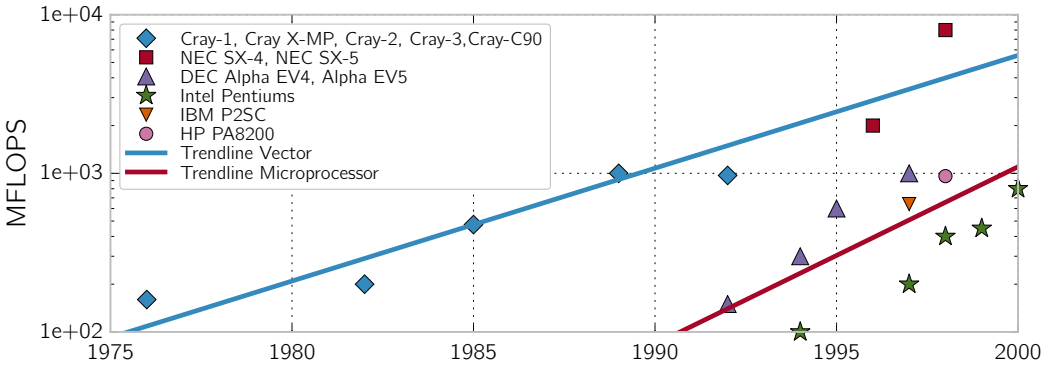



**Figure 1.2:** Vector and commodity processors peak floating-point performance development. *Data source: TOP500 list and various WWW sources.*

The lower per-processor performance meant that an application had to exploit ten parallel microprocessors to achieve the performance of a single vector CPU, and this required new programming techniques, including message-passing programming models such as Message Passing Interface (MPI). Commodity components, however, did eventually replace special-purpose HPC parts, simply because they were many times cheaper. Even though a system may have required ten times as many microprocessors, it was still cheaper overall.

As a consequence, a new class of parallel computers built on commodity microprocessors and distributed memories, gained momentum. In 1997, the ASCI Red supercomputer [89] became the first system to achieve 1 TFLOPS performance in the High-Performance Linpack (HPL) benchmark by exploiting 7,246 parallel Intel Pentium Pro processors [122]. Most of today's HPC systems in the TOP500 are still built on the same principle: exploit a massive number of microprocessors, based on the same technology used for commodity PCs. These systems represent 81% of the total systems in the June 2016 TOP500 list. Vector processors are almost extinct, although their technology is now present in most HPC processors in the form of widening SIMD extensions.

## 1.2 Energy Efficiency

Performance is not the single important metric for HPC systems. Supercomputers are large scale machines and as such have high power requirements [53]. Energy is increasingly becoming one of the most expensive resources and it substantially contributes to the total cost of running a large supercomputing facility. In some cases, the total cost over a few years of opera-

**Table 1.1:** Energy efficiency of several supercomputing systems from Green500 list. *Data source: Green500 list.*

| System | Heterogeneous | Architecture | GFLOPS/W |
|---|---|---|---|
| Shoubu | ✔ | PEZY-SC | 6.6 |
| Sunway TaihuLight | × | Sunway SW26010 260Cores Manycore | 6 |
| Sugon | ✔ | NVIDIA Tesla K80 Kepler GPU | 4.8 |
| Inspur TS1000 | ✔ | NVIDIA Tesla K20 Kepler GPU | 3 |
| SANAM | ✔ | AMD FirePro S10000 GPU | 2.97 |
| Piz Dora | × | Intel Xeon E5-2695 v4 | 2.7 |
| Shadow | ✔ | Intel Xeon Phi 5110 | 2.4 |
| Sequoia | × | IBM BlueGene/Q | 2.2 |

tion can exceed the cost of the hardware infrastructure acquisition [66, 70, 71]. Let us quantitatively describe the magnitude of energy expenditure, assuming the market electricity cost for an industry consumer in Spain, with a list price of ∼0.1 €/KWh [51]. Running a supercomputer with the same performance (93 PFLOPS) and power requirements (15.4 MW) as current June 2016 TOP500 list #1, Sunway TaihuLight supercomputer, in Spain would cost ∼13.5 M€ without VAT per year only in electricity costs.

Following performance development and power requirements, we can estimate that next milestone in supercomputers' performance, 1 EFLOP (1000 PFLOP) should be reached by the year 2020, but the required power for such a system will be up to ∼150 MW if the current energy efficiency is not improved[3]. Such a requirement is not realistic because it would demand setting a supercomputer facility next to an electricity production plant and the electricity bill of such a machine would not be sustainable. A more realistic power budget for an exascale machine is 20 MW [29], which would require an energy efficiency of 50 GFLOPS/W. This is more than an order of magnitude away from today's most energy efficient system as listed in Table 1.1.

To illustrate our premise about the need for low-power processors, let us reverse engineer a theoretical EFLOP supercomputer with a realistic power budget of 20 MW. We build our system using cores with 16 GFLOPS (8 ops/cycle @ 2 GHz), assuming that single-thread performance will not improve much beyond the performance we observe today. An Exaflop machine would require 62.5 million of such cores, independently on how they are packed together (multicore density, sockets per node). We also assume that only 30-40% of the total power will be actually consumed by the cores, the rest going to power supply losses, cooling infrastructure, interconnect, stor-

[3]For comparison, the total reported power of all supercomputers as per June 2016 TOP500 list is ∼225 MW

age and memory. That leads to a power budget of 6 MW to 8 MW for 62.5 million cores, which is 0.1 W to 0.13 W per core. Current high performance processors integrating this type of cores require tens of watts at 2 GHz. However, mobile processors like ARM processors, designed for the embedded mobile market, consume less than 0.9 W at that frequency [13], and thus are worth exploring—even though they do not yet provide a sufficient performance, they have a promising roadmap ahead – driven by the new commodity market.

## 1.3 Mobile Processors Evolution

Nowadays we observe a situation very similar to the one we observed between vector and commodity processors: low-power microprocessors targeted at mobile devices, such as smartphones and tablets, integrate enough transistors to include an on-chip floating-point unit capable of running typical HPC applications.

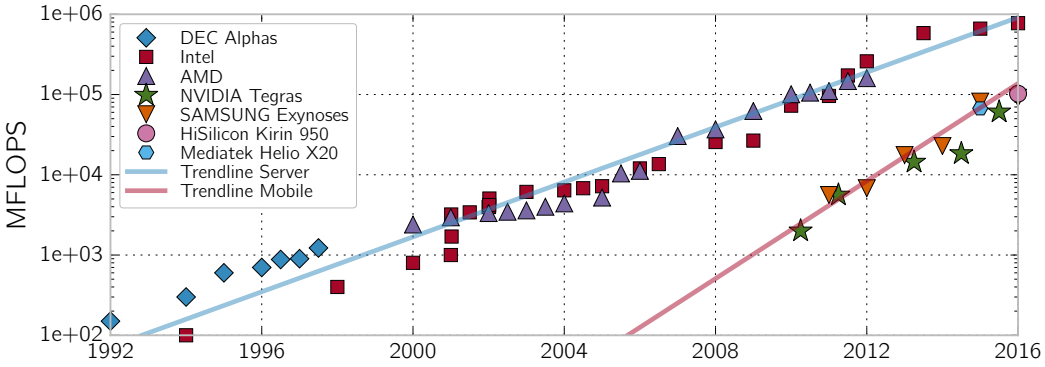



**Figure 1.3:** Server and mobile processors peak floating-point performance development. *Data source: TOP500 list, Microprocessor Report and various WWW sources.*

Figure 1.3 shows the peak floating point performance of current HPC microprocessors from Intel and AMD, compared with new floating-point capable mobile processors from NVIDIA and Samsung. The chart shows that mobile processors are not faster than their HPC counterparts. In fact, they are still ten times slower, but the trend shows that the gap is quickly being closed: the recently introduced ARMv8 Instruction Set Architecture (ISA) not only makes double-precision floating point (FP-64) a compulsory feature, but it also introduces it into the SIMD instruction set. That means that ARMv8 processors, using the same micro-architecture as the ARMv7 Cortex-A15,

would have double the FP-64 performance at the same frequency. Furthermore, mobile processors are approximately 70 times cheaper[4] than their HPC counterparts, matching the trend that was observed in the past.

Given the trend discussed above, it is reasonable to consider whether the same market forces that replaced vectors with RISC microprocessors, and RISC processors with x86 processors, will replace x86 processors with mobile phone processors. That makes it relevant to study the implications of this trend before it actually happens.

### 1.3.1 ARM Processors

The ARM architecture presents a family of RISC computer processor Intelectual Property (IP) cores mainly targeting embedded and mobile markets. ARM itself does not produce processors, but licenses cores design as intellectual property blocks to semiconductor manufacturers for further integration into potential designs. This business model turns the ARM architecture into the dominant cores architecture for embedded market. Official ARM Holdings report for 2012 [9] claims over 95% market share for smartphones and tablets. The predominant processor implementations in this share are Cortex-A9 and Cortex-A15, both based on the ARMv7-a ISA.

ARM-based mobile solutions attracted our attention once cores started implementing features that are desirable/compulsory for HPC. Previous generations of ARM application cores (Cortex-A family, ARMv7 revision of ISA) did not feature a floating-point unit capable of supporting throughputs and latencies required for HPC. First Cortex-A core, Cortex-A8, is an in-order core with an optional non-pipelined floating-point unit and in best case can deliver one floating-point ADD instruction every ∼10 cycles, with even smaller throughputs of MUL/FMAC instructions (FMAC—Fused Multiply Accumulate). In the best case, Cortex-A8 is capable of 0.105 GFLOPS at 1 GHz in double-precision. In addition, it implements NEON [16] SIMD floating-point unit which sadly supports only integer and single-precision floating-point arithmetics and thus is unattractive HPC due to a lack of double-precision support.

The follow-up core, namely the Cortex-A9, introduces out-of-order execution. It has an optional VFPv3 floating-point unit [17] and/or NEON SIMD

[4]We compare the official tray list price of an Intel Xeon E5-2670 [77] with the leaked volume price of NVIDIA Tegra 3 [73]: $1552 vs. $21. A fairer comparison; i.e. of the same price type, would be between the recommended list price for the Xeon with an Intel Atom S1260 [75]: $1552 vs. $64 which gives the ratio of ∼24. The latter is, however, not a mobile processor but a low-power server solution from Intel, and it serves only as a comparison reference.

floating-point unit. The VFPv3 unit is pipelined and is capable of executing one double-precision ADD operation per cycle, or one MUL/FMAC every two cycles. Then, with one double-precision floating-point arithmetic instruction per cycle (VFPv3), a 1 GHz Cortex-A9 provides a peak of 1 GFLOPS in double-precision.

ARM Cortex-A15 [124] is the successor of Cortex-A9 core, implementing the same ARMv7 ISA, with a more advanced microarchitecture implementation compared to ARM Cortex-A9. To begin with, VFPv3 unit is now fully pipelined and the Cortex-A15 can execute one FMAC instruction every cycle which leads to a peak performance of 2 GFLOPS at 1 GHz in double precision. Also, this generation of cores introduces Error-Correcting Code (ECC) protection in both L1 and L2 caches with ability to detect two and correct one error. In addition, Cortex-A15 can scale to sixteen cores on chip configuration, fully cache coherent, using ARM CoreLink CCN-504 [10, 33] on-chip interconnect. Cortex-A15 also improves on maximum addressable memory—even though it is 32-bit architecture and has a natural limit of 4 GB of addressable memory, it implements *large physical address extensions* [18], which removes 4 GB barrier and provides for up to the 1 TB of address space, with a limitation of 4 GB addressable memory per application/process. Last but not least, Cortex-A15 brought a lift in maximum operating frequency – microarchitecture and new technology nodes allowed frequencies up to 2GHz.

At the time we begin our study, ARMv7-a ISA based mobile cores IP were the state-of-the-art. Meanwhile, new 64-bit ARMv8 ISA based cores started to appear. New 64-bit ARMv8 ISA improves some features that are important for HPC. First, using 64-bit addresses removes the 4 GB memory limitation per application, allowing proper sizing of HPC nodes' memory. Also, ARMv8 increases the size of the general purpose register file from 16 to 32 registers. This reduces register spilling and provides more room for compiler optimization. It also improves floating-point performance by extending the NEON instructions with fused multiply-add and multiply-substract, and cross-lane vector operations. More importantly, double-precision floating-point is now part of NEON. All together,this provides a theoretical peak double-precision floating-point performance of 4 FLOPS/cycle for a fully-pipelined SIMD unit.

The first mobile core IP implementation of ARMv8 ISA is the ARM Cortex-A57 [32]. It includes two NEON units, totalling 8 double-precision FLOPS/cycle – this is 4 times better than ARM Cortex-A15 and equivalent to Intel Sandy Bridge microarchitecture double-precision floating-point throughput with Intel Advanced Vector Extensions (AVX). Its microarchitecture allows for implementation of up to 2.5 GHz while staying within a

smartphone thermal envelope.

### 1.3.2 Embedded GPUs

Unlike the discrete GPUs for servers, a mobile GPU is integrated into a SoC, which also includes a multi-core CPU, and multiple workload accelerators and offload engines. Recently, modern mobile GPUs such as the Imagination PowerVR [74], NVIDIA ULP (Ultra Low-Power) GeForce [100] or ARM Mali-T6xx [14] GPUs tend to integrate more computing units in a chip, thus increasing the aggregate peak performance of a mobile SoC.

Like their predecessors, discrete desktop or server GPUs, they were not programmable when we began our study by means of general purpose programming models like CUDA or OpenCL. Instead, mobile GPU were programmed with OpenGL ES [96] being a low-level API not targeting general-purpose computing and thus not very attractive. Recently, mobile GPU vendors are starting to add support for programmability to their solutions. The first OpenCL programmable and computing capable mobile GPU, ARM Mali-T604, has the peak single-precision floating-point performance of 68 GFLOPS in single-precision arithmetic. ARM has also announced the next generation mobile GPU, ARM Mali-T658 [15] which is supposed to deliver four times more computing performance compared to its predecessor. In addition, NVIDIA launched two mobile SoCs with computing capable GPUs, programmable by means of NVIDIA CUDA. These SoCs are Tegra K1 [45] (based on Kepler GPU core architecture), and Tegra X1 [37] (based on Maxwell GPU architecture), offering peak floating-point performance of 384 and 512 GFLOPS in single-precision respectively. What makes on-SoC mobile GPUs attractive is the fact they share main memory with a host CPU, thus avoiding explicit memory copies, and using pinned memory instead – saving the energy on data movements.

## 1.4 Contributions

Our study of mobile processors and SoCs potential for HPC, documented in this thesis, brings the following contributions to the scientific community:

- Evaluation of ARM mobile processors and SoCs in HPC environments, and comparison to their contemporary x86 processor,
- Design and evaluation of the world first mobile SoCs powered HPC cluster built from developer kits,

- Evaluation of mobile SoCs featuring next-generation core IP, and heterogeneous architectures with off and on-chip GPU accelerators,
- Design and thorough evaluation of the Mont-Blanc prototype, the next-generation mobile SoCs powered HPC cluster built with off-the-shelf HPC network and storage solutions, and contemporary system integration,
- Guidelines for the design of the next-generation mobile SoCs based HPC system.

The flow of the thesis follows the timeline of our contributions, and is organized as follows: In Chapter 2 we discuss the related work. Performance of ARM Cortex-A9 and corresponding software stack tuning is discussed in Chapter 4. With Chapter 5 we introduce the Tibidabo cluster – world first mobile SoCs based HPC cluster. Here we present the evaluation, and analysis of energy-efficient computing potential of such a solution. Further, in Chapter 6 we evaluate multiple mobile SoCs and assess the performance of GPGPU (General-Purpose computation on Graphics Processing Unit) computing towards making a selection of a mobile SoC for the Mont-Blanc prototype. The architecture, evaluation, performance analysis of the Mont-Blanc prototype and comparison against a production level supercomputer is depicted in Chapter 7. In Chapter 8 we show a study that aims to give a design guidelines for a next-generation mobile SoCs powered HPC systems. Conclusions and future research directions are shown in Chapter 9.

# 2
# Related Work

The landmark supercomputer[1], ASCI Red [89], was the first top-tier system to utilize the same processors as found in commodity desktop machines – instead of powerful HPC-optimized vector processors. It integrad 7,246 Intel Pentium Pro processors[2], the first x86 commodity processors to support out-of-order execution and SMP (Symmetric Multiprocessing) with up to four processors natively. All components of the system were COTS (Commercial Off-The-Shelf), except the proprietary network. This particular system marked the beginning of the era of system designs based on the same principle: clusters of commodity PC processors; and the beginning of the end for vector processors.

Another example of an HPC machine using technology from different computing segments is the Roadrunner [27] supercomputer. It was based on the Cell/B.E. processor, primarily designed for the Sony Playstation 3 game console. Roadrunner's node was based on dual-core AMD Opteron processors

It topped the Top500 list in June 2008 to be the first to break the petaflop barrier. It uses IBM PowerXCell 8i [41] together with dual-core AMD Opteron processors. The Cell/B.E. architecture emphasizes performance per watt by prioritizing bandwidth over latency and favors peak computation capabilities over simplifying programmability. In the June 2008

[1]From this moment on we will use the terms *supercomputer and* HPC system equally.

[2]Later the number of processors was increased to 9,632 and upgraded to Pentium II processors

Green500 list, it held third place with 437.43 MFLOPS/W, behind two smaller homogeneous Cell/B.E.-based clusters.

The Cell/B.E. represents an example of a consumer device technology[3] used for HPC.

One of the first attempts to use low-power commodity processors in HPC systems was GreenDestiny [129]. They relied on Transmeta TM5600 processor, and although the proposal seemed good for a top platform in energy efficiency, a large-scale HPC system was never produced. Also, its computing-to-space ratio was leading at the time.

MegaProto systems [99] were another approach in this direction. They were based on more advanced versions of Transmeta's processors, namely TM5800 and TM8820. This system was able to achieve good energy efficiency for the time, reaching up to 100 MFLOPS/W using a system with 512 processors. Like its predecessor, MegaProto never made it into a commercial HPC product.

There has been a proposal to use the Intel Atom family of processors in clusters [126]. The platform is built and tested with a range of different types of workloads, but those target data centers rather than HPC. One of the main contributions of this work is determining the type of workloads for which Intel Atom can compete in terms of energy-efficiency with commodity Intel Core i7. A follow-up of this work [84] leads to the conclusion that a cluster made homogeneously of low-power nodes (Intel Atom) is not suited for complex database loads. They propose future research in heterogeneous cluster architectures using low-power nodes combined with high-performance ones.

The use of low-power processors for scale-out systems was assessed in a study by Stanley-Marbell and Caparros-Cabezas [119]. They did a comparative study of three different low-power architecture implementations: x86-64 (Intel Atom D510MO), Power Architecture e500 (Freescale P2020RDB) and ARM Cortex-A8 (TI DM3730, BeagleBoard xM). The authors presented a study with performance, power and thermal analyses. One of their findings is that a single core Cortex-A8 platform is suitable for energy-proportional computing, meaning very low idle power. However, it lacks sufficient computing resources to exploit coarse-grained task-level parallelism and be a more energy efficient solution than the dual-core Intel Atom platform. They also concluded that a large fraction of the platforms' power consumption (up to 67% for the Cortex-A8 platform) cannot be attributed to a specific component, despite the use of sophisticated techniques such as thermal imaging for determining the power breakdown.

[3] Cell/B.E. used to power Sony Playstation 3 game console

The AppleTV cluster [56, 55] is an effort to assess the performance of the ARM Cortex-A8 processor in a cluster environment running HPL. The authors built a small cluster with four nodes based on AppleTV devices with a 100MbE network. They achieved 160.4 MFLOPS with an energy efficiency of 16 MFLOPS/W. Also, they compared the memory bandwidth against a BeagleBoard xM platform and explained the performance differences due to different design decisions in the memory subsystems. In our systems, we employ more recent low-power core architectures and show how improved floating-point units, memory subsystems, and an increased number of cores can significantly improve the overall performance and energy efficiency, while still maintaining a small power footprint.

The BlueGene family of supercomputers has been around since 2004 in several generations [2, 3, 72]. BlueGene systems are composed of embedded cores integrated on an Application Specific Integrated Circuit (ASIC) together with additional architecture-specific fabrics. BlueGene/L, the first such system, is based on the PowerPC 440, with a theoretical peak performance of 5.6 GFLOPS. BlueGene/P increased the peak performance of the computing card to 13.6 GFLOPS by using 4-core PowerPC 450. BlueGene/Q-based clusters are one of the most power efficient HPC machines nowadays delivering around 2.1 GFLOPS/W. A BlueGene/Q computing chip includes 16 4-way SMT in-order cores, each one with a 256-bit-wide quad double-precision SIMD floating-point unit, delivering a total of 204.8 GFLOPS per chip on a power budget of around 55 W (3.7 GFLOPS/W).

In parallel with our work, there have been multiple SoCs and commercial solutions using embedded processors and targeting server market: the Calxeda EnergyCore ECX-1000 [34], and AMD Opteron A1100 [4] are ARM based, while the AMD SeaMicro SM10000-64 [46] and the Quanta Computer S900-X31A [108] are based on the Intel Atom. All extend the embedded multicore with high bandwidth networks, for example 10GbE, and ECC memory protection.

Meanwhile, Other companies have developed custom processors based on the ARM architecture. Applied Micro (APM) X-Gene [6] is a server-class SoC with eight 64-bit ARMv8 cores and four 10GbE links. Cavium, with large experience in networking processors, designed ThunderX [38], another server-class SoC with 48 ARMv8 cores and multiple 10/40GbE interfaces. Qualcomm and Phytium also announced ARMv8 server SoCs with 24 [105] and 64 [40] cores, respectively.

One of the most exciting ARMv8 server chip projects is the Vulcan SoC from Broadcom [65]. It is a custom microarchitecture implementation of the ARMv8 ISA. The Vulcan core is a 4-way SMT out-of-order CPU core, designed to run at frequencies up to 3GHz. Compared to the other ARMv8

implementations, both ARM Cortex-A series and custom cores, it has two floating-point units both supporting FMAC (Fused Multiply Accumulate) in SIMD, for a total of 8 double-precision floating-point operations per cycle. Moreover, the core can support up to 64 outstanding loads and 36 outstanding stores – far more compared to the other ARM implementations. Given its microarchitecture, it is aimed to be competitive with the server-class Intel Xeons. Sadly, this promising approach towards the ARMv8 server core architecture has been shut-down recently after the acquisition of Broadcom by another company.

Some successful deployments of some of these SoCs are already in place. CERN has published a comparison of APM X-Gene compared to Intel Xeon and IBM Power8 chips [1]. PayPal has deployed HP Moonshot servers with APM X-Gene processors claiming half the price, one seventh of the power consumption and 10x more nodes per rack compared to their traditional data center infrastructure [43].

These efforts, however, target the server market and there are still no large-scale demonstrators of such mobile-technology-based processors for HPC. The Mont-Blanc prototype is thus the first demonstrator of an HPC cluster with full HPC software stack running real scientific application, commodity networking, and standard system integration. Our experiments demonstrate the feasibility of the proposed alternative approach, assess system software maturity and project its scalability at a larger scale.

Blem et al. [31] recently examined whether there were any inherent differences in performance or energy efficiency between the ARM and x86 ISAs. They brought once-more the RISC (ARM) vs. CISC (x86) debate, and found that although current ARM and x86 processors are indeed optimised for different metrics, the choice of ISA had an insignificant effect on both power and performance. We do not contradict this finding. We argue that whichever architecture dominates the mobile industry will, provided the level of performance demanded in the mobile space is sufficient, eventually come to dominate the rest of the computing industry.

Li et al. [86] advocate the use of highly-integrated SoC architectures in the server space. They performed a design space exploration, at multiple technology nodes, for potential on-die integration of various I/O controllers, including PCIe, NICs and SATA. They predicted that, for data centers at the 16nm technology node, an SoC architecture would bring a significant reduction in capital investment and operational costs. With our study, we also advocate the use of SoCs, but for future HPC systems based on mobile and embedded technology.

At the International Supercomputing Conference 2016, Fujitsu announced their plains to produce a processor for the Post-K Exascale Supercomputer,

based on their own microarchitecture implementation of ARMv8 architecture specification [39]. This supports our approach of using commodity technology deployed in the mobile and embedded markets, in this case the ARMv8 architecture, for HPC.

# 3 Methodology

In this chapter we present the high-level methodology used in our research. However, each chapter covers the specifics of the methodology in more details if needed.

All results presented in this thesis were either gathered on *real hardware platforms*, or have foundation in a design-space exploration with simulations. One of the strengths of our work is that we build our simulation models around real hardware platforms, and later extrapolate the results.

In this chapter we also list benchmarks used to evaluate the CPU performance and energy efficiency, memory bandwidth and network bandwidth across different platforms. Finally, we present the software tools we used to aid our study as well.

## 3.1 Hardware platforms

Due to the specific nature of mobile SoCs there were/are not available hardware platforms targeting HPC using this very technology. Thus, we opted to use developer boards (as seen in Chapters 4,5,6) for conducting evaluations of single mobile CPU core, single mobile GPU, single node benchmarking, and comparison against x86-based cores and nodes. Additionally, developer boards were used in initial phase of our work for porting and tuning HPC software stack for ARM-based platform.

For large scale HPC studies requiring clusters, we had to deploy prototype

clusters ourselves. Initially, this led to the in-house design and deployment of 256 nodes Tibidabo cluster prototype based on developer kits (see Chapter 5). Although Tibidabo provided valuable insights, we continued our research on a mobile SoC based system utilizing professional system integration and featuring more advanced mobile and embedded technology. Thanks to the Mont-Blanc project [94] we deployed and tuned the 1080 nodes Mont-Blanc prototype (as seen in Chapter 7) which served as a proof-of-concept of mobile technology based HPC, and as the testbed for specifying the next-generation system design (see Chapter 8).

When comparing our prototype platforms and clusters against x86 based machines, we used LiDOng cluster and MareNostrum III supercomputer as reference platforms. Former was installed at TU Dortmund and the latter is hosted by Barcelona Supercomputing Center.

## 3.2 Single core, CPU, and node benchmarks

Throughout this study we benchmark different mobile SoC powered platforms and systems, and compare them against x86 based platforms and systems. Our initial study (see Chapter 4) in compiler maturity of mobile ARM platforms uses Dhrystone and LINPACK1000x1000 benchmarks, where the former represents integer and the latter is a floating-point benchmark. First comparison of an ARM mobile core and an x86 core (see Chapter 4) is done with the SPEC CPU2006 [68] – a well established industry grade benchmark suite for comparing CPU cores having a good mix of both integer and floating-point workloads (SPECINT and SPECFP). However, SPEC CPU2006 imposed a significant porting effort, and since we aimed to evaluate multiple computing node architectures, some of which include computing accelerators (GPU or DSP), we decided to design our own benchmark suite presented in Section 3.2.1. For the purpose of characterization of memory systems we employ STREAM benchmark [90] which measures achievable memory bandwidth from the both single core and entire CPU (node) perspective.

### 3.2.1 Mont-Blanc benchmarks

Mont-Blanc benchmarks is a highly-portable suite of 11 benchmarks, written in C, that stress different architectural features and cover a wide range of algorithms employed in HPC applications. All microbenchmarks are developed in five versions. The serial version is used in single-core scenarios to test the performance of a single CPU core. The multi-threaded version of the benchmarks uses OpenMP programming model. For GPGPU comput-

ing, both NVIDIA's CUDA [102] and OpenCL versions exist. Mont-Blanc benchmarks are developed in OmpSS [48] version as well, allowing full fully exploitation of heterogeneous devices employing CPU and GPU for computations.

Here we present the full list of the Mont-Blanc benchmarks with a brief description.

**Vector Operation (vecop):** This code takes two vectors of a given size and produces an output vector of the same size by performing addition on an element-by-element basis. This workload mimics the vector operations often found in numerical simulations, and other computing intensive regular codes.

**Dense Matrix-Matrix Multiplication (dmmm):** This code takes two dense matrices and produces an output dense matrix that is the result of multiplication of the two input matrices. Matrix multiplication is common computation in many numerical simulations and the benchmark measures the ability of the computing accelerator to exploit data reuse and computing performance.

**3D Stencil (3dstc):** This code takes one 3D volume and produces an output 3D volume of the same size. Each point in the output volume is calculated as a linear combination of the point with the same coordinates in the input volume and the neighboring points in each dimension, i.e., points with the same coordinates as the input point plus/minus an offset in only one dimension. This code evaluates the performance of strided memory accesses on the computing accelerator. Moreover, by allowing the number of stencil points to be variable, different memory load/computation ratios can be evaluated.

**2D Convolution (2dcon):** This code takes two input matrices, first being an input image and the second representing a filter, and produces an output image matrix of the same size as the input image. Each point in the output matrix is calculated as a linear combination of the points in the filter matrix and the points in the input sub-matrix of the same size as filter matrix. Central coordinate of the sub-matrix corresponds to the coordinate of the current output matrix point. Contrary to the 3D stencil computation, neighboring points can include points with the same coordinates as the input point plus/minus an offset in one or two dimensions. This code allows measuring the ability of the computing accelerator to exploit spatial locality when the code performs strided memory accesses.

**Fast Fourier Transform (fft):** This code takes one input vector and produces an output vector of the same size by computing a one-dimensional Fast Fourier Transform. This is computing intensive code that measures the peak floating-point performance, as well as variable stride memory accesses.

**Reduction (red):** This code takes one input vector and applies the addition operator to produce a single (scalar) output value. The amount of data parallelism in this code decreases after each reduction stage. This allows us to measure the capability of the computing accelerator to adapt from massively parallel computation stages to almost sequential execution.

**Histogram (hist):** This code takes an input vector and computes the histogram of values in the vector, using a configurable bucket size. This code uses local privatization that requires a reduction stage which can become a bottleneck on highly parallel architectures.

**Merge Sort (msort):** This code takes an input vector of any arbitrary type, and produces a sorted output vector. It requires synchronization of execution threads after each merge step, and serves as a good hint about the performance of barrier instructions on the computing accelerator.

**N-Body (nbody):** This code takes a list describing a number of bodies including their position, mass, and initial velocity, and updates these parameters with new values after a given simulated time period, based on gravitational interference between the bodies. This code is used to characterize the performance of irregular memory accesses on the computing accelerator.

**Atomic Monte-Carlo Dynamics (amcd):** This code performs a number of independent simulations using the Markov Chain Monte Carlo method. Initial atom coordinates are provided and a number of randomly chosen displacements are applied to randomly selected atoms which are accepted or rejected using the Metropolis method. This code is embarrassingly parallel with no data sharing across execution threads and is a measurement of the peak performance the computing accelerator can achieve in absence of inter-thread communication.

**Sparse Vector-Matrix Multiplication (spvm):** This code takes a vector and a sparse matrix as inputs, and produces an output vector that is the result of multiplication of the input matrix and vector. This code assigns a different workload to each execution thread, and serves as a measurement of the performance of the computing accelerator when load imbalance occurs.

## 3.3 System benchmarks and workloads

For the purpose of systems benchmarking, where a system represents a *distributed memory* cluster consisting of at least two nodes, we utilize MPI benchmarks and applications.

Characterization of a node MPI capability in terms of achievable message bandwidth and latency of a single node is done with Intel MPI benchmarks [76]. More precisely, we used MPI PingPong benchmark which allows

for obtaining of bandwidth and latency figures for MPI data exchange between two processes as a function of the message size in two scheduling scenarios: *intra-node* – where MPI utilizes system interconnect, and *inter-node* when shared memory is used as a communication medium.

Benchmarking scalability and performance of mobile SoC prototype clusters, and comparison against a production level machines is done with full-scale production MPI applications and with additional mini proxy-apps used by US DoE (Department of Energy) and DoD (Deprtment of Defence) for next-generation systems co-design. Porting of full-scale MPI application created a significant effort, thus the choice of the applications was limited to either those requiring the smallest possible effort, or applications ported and provided by the Mont-Blanc project consortium. Further, in order to determine floating-point performance and energy-efficiency of an HPC cluster we employ HPL following the common benchmarking procedures. In the Table 3.1 we list all test applications used in this work.

**Table 3.1:** Methodology: list of parallel MPI applications and benchmarks used for scalability, performance, and energy-efficiency evaluations of mobile SoC clusters.

| **Application** | **Domain** |
|---|---|
| Full-scale applications | |
| Alya [128, 127] | Biomedical Mechanics |
| BigDFT [30, 57] | Electronic Structure |
| BQCD [97] | Quantum Chromodynamics |
| GROMACS [28] | Molecular Dynamics |
| FEAST [123] | PDE solver |
| MP2C [120] | Multi-Particle Collision Dynamics |
| PEPC [132] | Particle Hydrodynamics |
| QuantumESPRESSO [59] | Electronic Structure and Materials Modeling |
| SMMP [49, 50, 91] | Molecular Thermodynamics |
| Benchmarks | |
| HONEI_LBM [125] | Fluid dynamics |
| HYDRO [44, 85] | Hydrodynamics |
| HPL [47] | Solver for dense $n \times n$ system of linear equations |
| Mini-apps | |
| CoMD [52] | Proxy for Molecular Dynamics |
| miniFE [69] | Proxy for Finite Element Method |
| LULESH [82, 81] | Proxy for Hydrodynamics |

## 3.4 Power measurements

Due to the heterogeneity of the hardware platforms we use, different power measurements approaches were taken depending on the available measurement points (probes) and power consumption data. Here we present the power measurements setups we used for the majority of experiments throughout this thesis. If needed, we further describe and explain the methodology specifics in each chapter.

For the purpose of studies relying on the single-node mobile SoC developer boards and x86 platforms, we measure the instantaneous power drawn from the AC socket during the executions of interest. Power acquisition setup is depicted in Figure 3.1.

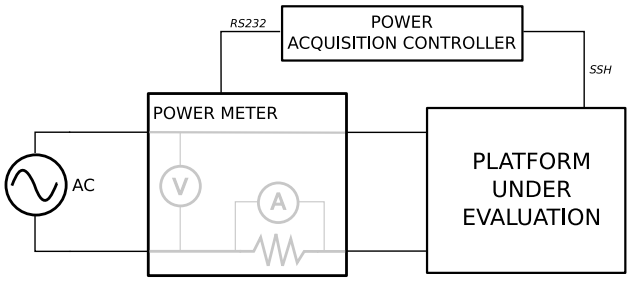



**Figure 3.1:** Methodology: power measurement setup.

Power meter is set to measure voltage and current drawn from the AC line. Current is measured in the return path of the circuit (low side). We employ Yokogawa WT230 power meter [98] with a precision of 0.1%, and it outputs RMS (Root-Means-Square) voltage/current pairs every ∼250ms. Triggering of power measurements starts at Device Under Test (DUT) which sends *start/stop acquisition* commands to the Power Acquisition Controller (PAC) through SSH. The PAC, in turn, triggers acquisition cycle at the power meter through RS232 interface. The PAC also stores power measurement data collected every ∼250ms from the power meter. Power acquisition if fully integrated and automated within benchmarking infrastructure.

## 3.5 Simulation methodology

In Chapter 5 and Chapter 8 we present design-space exploration for alternative mobile SoC powered HPC systems. We base our study on Dimemas [22], in-house coarse grain trace-driven simulator of MPI applications which performs high-level simulation of the execution of MPI applications on target

cluster systems. Dimemas has been used to model the interconnect of the MareNostrum supercomputer with an accuracy within 5% [110], and its MPI communication model for collective communications has been validated showing an error below 10% for the NAS benchmarks [60].

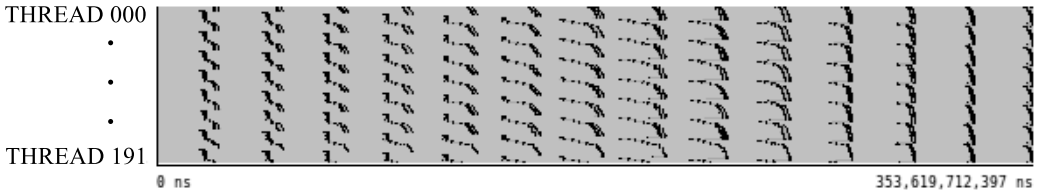
Dimemas uses a high-level model of computing nodes, modelled as SMP (Symmetric Multi-Processing) nodes. At the same time it uses an analytical model of the interconnect [60] to account for the effects of MPI communications. Dimemas allows for parametric studies, varying basic parameters describing a target architecture, such as: number of cores per node, relative core speed[1], memory bandwidth[2], latency of MPI communications through shared memory, inter-node MPI bandwidth (network bandwidth) and inter-node MPI latency. In our simulations we mostly altered number of cores per node, relative core speed, inter-node MPI bandwidth and latency.

As an example, the Paraver [106] visualization of the input and output traces of a Dimemas simulation are shown in Figure 3.2. The chart shows the activity of the application threads (vertical axis) over time (horizontal axis). Figure 3.2a shows the visualization of the original execution on Tibidabo, and Figure 3.2b shows the visualization of the Dimemas simulation using a configuration that mimics the characteristics of our machine (including the interconnect characteristics) except for the CPU speed which is, as an example, 4 times faster. As it can be observed in the real execution, threads do not start communication all at the same time, and thus have computation in some threads overlapping with communication in others. In the Dimemas simulation, where CPU speed is increased 4 times, computation phases (in grey) become shorter and all communication phases get closer in time. However, the application shows the same communication pattern and communications take a similar time as that in the original execution. Due to the computation-bound nature of HPL, the resulting total execution time is largely shortened. However, the speedup is not close to 4x, as it is limited by communications, which are properly simulated to match the behavior of the interconnect in the real machine.

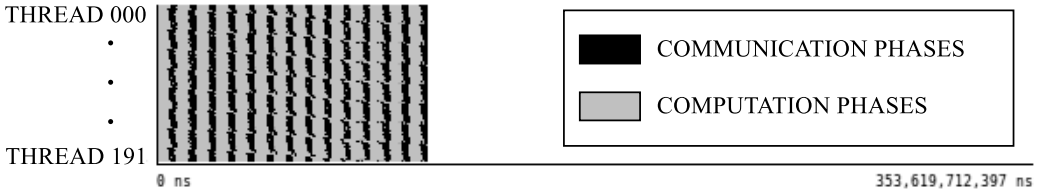
Apart from design-space exploration, we also use Dimemas together with a state-of-the-art methodology [36, 111] to extract basic performance informations in order to estimate scalability of MPI applications beyond the number of cores present in our prototype cluster. In addition, using Dimemas we can simulate bypassing of networking protocols (e.g. TCP/IP) and elimination of OS noise present in application traces, thus we can actually quantify

[1]This is what we call CPU speed ratio. It is a relative number showing the ratio between the CPU speed of the target CPU compared to the one found in a cluster where MPI application trace is collected.

[2]This is actually MPI bandwidth for intra-node communications




(a) Part of the original HPL execution on Tibidabo




(b) Dimemas simulation with hypothetical 4x faster computation cores

**Figure 3.2:** An example of a Dimemas simulation where each row presents the activity of a single processor: it is either in a computation phase (grey) or in MPI communication (black).

the impact of aforementioned phenomena on MPI applications performance for future systems design.

## 3.6 Tools

In this section we present software tools we used to conduct our design-space exploration studies and for analysis of MPI applications running on our prototypes.

### 3.6.1 Extrae

Extrae [25] is a dynamic instrumentation package designed by Barcelona Supercomputing Center, which generates applications execution traces for post-mortem analysis. Extrae offers manual and automatic instrumentation, where we use the former with serial benchmarks and the latter with parallel MPI applications. Manual instrumentation requires altering and annotating the source code with Extrae hooks, while the automatic relies on the linker pre-loading mechanism. Extrae captures time-stamped events, e.g. entry/leave of a MPI function call, and provides support for gathering additional statistics such as performance counters values at each sampling point. Combining time-stamps and performance counters, we get a complete picture about applications performance running on our cluster prototypes.

### 3.6.2 Paraver

For sanity checks and analysis of Extrae traces we use Paraver [106] – a powerful and flexible GUI data browser. It supports trace visualization in terms of time-lines, and 2D and 3D histograms, allowing for detecting OS and hardware issues and different imbalances found in parallel applications, such as: CPU throttling due to overheating, OS noise, timeouts, and load imbalance. We also use Paraver to analyze the output traces from Dimemas simulations.

### 3.6.3 Clustering tools

We use BSC Clustering tool [24] in order to detect and cluster computing phases found in applications' traces by their respective performance counters metric. In our work, we cluster computing phases in the *Total number of cycles* vs IPC (Instructions Per Cycle) space. This allows us to model an MPI application with increased accuracy, instead of assuming all computing phases are equal (see Section 8.2).

### 3.6.4 Basic analysis tool

We use BSC Basic analysis tool [23] to extract fundamental parallel performance parameters [111, 36] from an application execution trace. These parameters are as follows: *Load Balance efficiency* reflecting the potential parallel efficiency loss caused by imbalance in per process execution times; *Serialization efficiency* reflecting imbalance caused by dependencies in the code; *Transfer efficiency* which estimate the performance loss caused by actual data transfers. Combining these factors together we measure and extrapolate parallel efficiency beyond the process count present in an application trace.

### 3.6.5 GA tool

GA tool is our metaheuristic optimizer that searches for a solution of overdetermined systems of equations applying concepts of evolution with Genetic Algorithms (GA) [62]. We use the tool as a part of the IBM's proposed analytical performance prediction methodology of parallel applications using benchmarks [117]. The tool matches a performance counters statistics of a computational phase to that of a serialized execution of multiple benchmarks – in our case Mont-Blanc benchmarks.

## 3.7 Reporting

All benchmarking data gathered from real hardware represents the arithmetic mean of execution times (or rates where appropriate) from at least 10 executions unless stated otherwise. There was no significant variability observed between consecutive measurements thus the error bars were omitted. Finally, due to the deterministic nature of Dimemas MPI simulations (trace replay) each experiment configuration is executed exactly once and the corresponding output is used for further analyses.

Unless stated otherwise, all power data is related to average power consumption, while the energy is calculated as integral of instantaneous power consumption over time[3]

[3]If the average power is the only available metric on the target platform, we multiply it with execution time to get the energy expenditure.

# 4

# ARM Processors Performance Assessment

In this chapter we assess the performance of the ARM Cortex-A9 processor, leader in mobile computing of the period 2010-2011, as a potential building block of a mobile SoC based HPC system. We discuss the importance of compiler maturity for systems where every percentile of performance matters. Further, we compare the ARM Cortex-A9 processor with its contemporary power-optimized Intel Core-i7 M640 processor[1], in order to establish a meaningful reference between our target platform and a widely adopted core architecture for PC's and servers.

## 4.1 Floating-Point Support Issue

Mobile SoCs built around ARM core IP blocks allow for customizing the target design, depending on the workloads one wants to optimize the silicon for. Historically, with ARMv5 ISA specification ARM introduced VFP (Vector Floating Point), and since then one could opt to synthesize a processor with or without a floating-point co-processor[2]. That hardware diversity would in turn introduce a fragmentation in software support – in operating systems

[1] Both NVIDIA Tegra2 and Intel i7 M640 were released on Q1 2010

[2] With the ARMv8 architecture specification the floating-point unit is a mandatory implementation feature.

and corresponding system libraries, and compiler targets. In the case of GCC compiler, this indeed led to fragmentation and there were three different options for selecting whether a processor has a floating-point hardware or not, and if arguments should be passed through floating-point registers or not. This is specified with the GCC option *-mfloat-abi* which, for ARM targets, takes one of the following three values and generates the code for proper Application Binary Interface (ABI):

- ***soft*** - floating-point code is emulated – GCC inserts library-calls for floating point operations. Passing the arguments to floating point functions, both single and double precision, goes through integer registers.
- ***softfp*** - floating-point code is executed in hardware, but the calling conventions stay the same as in the case of ***soft***, using integer registers for passing the arguments.
- ***hardfp*** - floating-point code is executed in hardware, and procedure arguments are passed directly to floating-point registers.

One important fact to note is that, due to the calling conventions, ***soft*** and ***softfp*** code can be intermixed.

## 4.2 Compiler Flags Exploration

For reasons of portability and huge diversity of embedded and mobile SoCs based on ARM core IP, operating system images and compilers opt to provide the highest possible backward compatibility for different targets. For this reason, Ubuntu operating system for ARM platforms that implement floating-point co-processor used to ship GCC compiler package without default support for floating-point hardware. At the time of ARM Cortex-A9 exploration, operating system and libraries were built without ***hardfp*** support, meaning that they were not particularly optimized to take the advantage of the built-in floating-point hardware support by default. More precisely, they supported *softfp* ABI which would enable hardware floating-point support, but GCC v4.3.3 used to produce the code for *soft* ABI by default. Since the *soft* and *softfp* ABIs are compatible, binaries would execute but the performance penalty we discovered was significant due to the emulation of floating-point operations.

For example, considering a simple multiply-accumulate kernel $A = A + (B \times C)$, without any particular optimization, experiences $12\times$ speedup when the target ABI was changed from *soft* to *softfp*. This ABI change, with

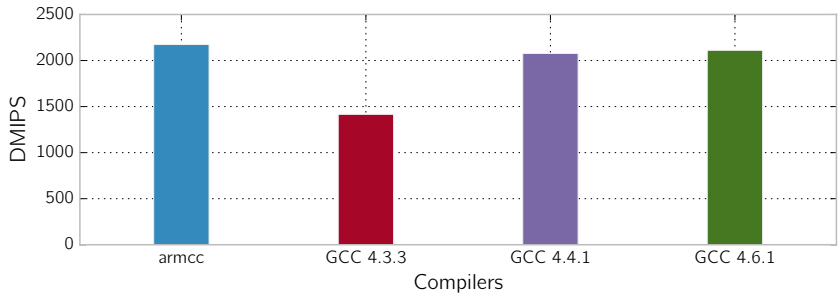



**Figure 4.1:** Comparison of different compilers for ARM Cortex-A9 with Dhrystone benchmark. Every compiler uses highest available optimization level, targeting reduction of execution time.

GCC option *-mfloat-abi=softfp*, and specifying correct revision of floating-point hardware with *-mfpu=vfpv3-d16*, would force the compiler to generate floating-point instructions in the place of library calls for emulation.

Further examination of GCC compiler documentation led to establishment of the default minimum set of platform specific compiler flags: ***-march=armv7-a -mcpu=cortex-a9 -mtune=cortex-a9 -mfloat-abi=softfp -mfpu=vfpv3-d16***. From now on, this is considered default for ARM Cortex-A9 throughout this thesis.

### 4.2.1 Compiler Maturity

Integer performance of ARM Cortex-A9 is claimed to be 2.5 DMIPS/Mhz (Dhrystone MIPS) We wanted to both evaluate this claim, and evaluate the maturity of GCC compiler suite in generating this type of code. Thus, we evaluated the ARM Cortex-A9 core running at 1GHz[3] with Dhrystone using both *armcc*, ARM's proprietary compiler, and GCC compilers.[4]

In Figure 4.1 we depict the achievable DMIPS scores when running Dhrystone binaries using four different compilers – *armcc* and three versions of GCC compiler. GCC v4.3.3 is default package shipped with the Ubuntu distribution, while GCC v4.4.1 and v4.6.1 were compiled from the source. Dhrystone results clearly show that, despite performance advantage of *armcc* over GCC v4.3.3, upgrading the compiler made GCC a competitive alternative for integer codes.

Further, we looked into the quality of the generated floating-point code, through reported GFLOPS, using C-port of the LINPACK 1000x1000 bench-

[3]Expected Dhrystone performance is 2500 DMIPS.

[4]It is well known fact that CPU vendors tend to have specific support in their compilers to emit highly optimized code for well established benchmarks.

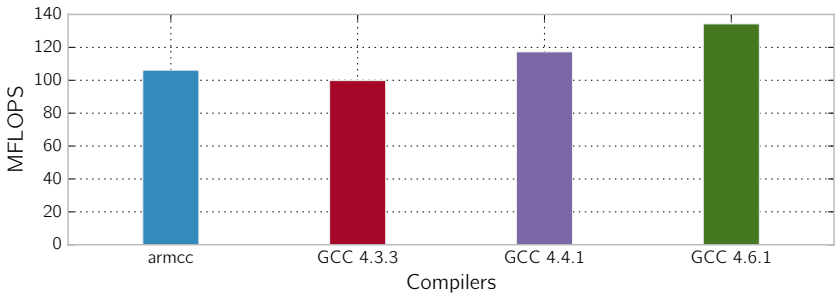



**Figure 4.2:** Comparison of different compilers for ARM with LINPACK1000x1000 benchmark. Every compiler uses highest available optimization level targeting reduction of execution time.

mark [87]. It solves 1000x1000 general dense matrix problem $Ax = b$ in double-precision floating-point arithmetic. Again, we compare *armcc* with multiple versions of GCC compilers. Figure 4.2 depicts achievable MFLOPS using different compilers.

Results from this experiment are twofold: clearly, there is no advantage in using *armcc* compiler for dense matrix codes such as LINPACK1000x1000, and GCC compilers provide sufficient and even higher performance with such codes – clearly showing that community is working towards supporting ARM targets in GCC.

## 4.3 Achieving Peak Floating-Point Performance

In previous section we have shown that ARM Cortex-A9 could achieve ∼130 MFLOPS executing LINPACK1000x1000 benchmark. ARM Cortex-A9 core, given its microarchitecture, has the following throughput of double-precision floating-point instructions:

- *FADD* - addition, one every cycle,
- *FMUL* - multiplication, one every other cycle, and
- *FMAC* - fused multiply-accumulated, one every other cycle.

This means that ARM Cortex-A9, using VFP floating-point co-processor, could achieve one FLOP/cycle. Since our test platform is running at 1GHz, hence we would expect to get 1 GFLOPS peak double-precision performance from underlying hardware.

In Figure 4.3 we evaluate the performance of Cortex-A9 floating-point double-precision pipeline using in-house developed microbenchmarks. These

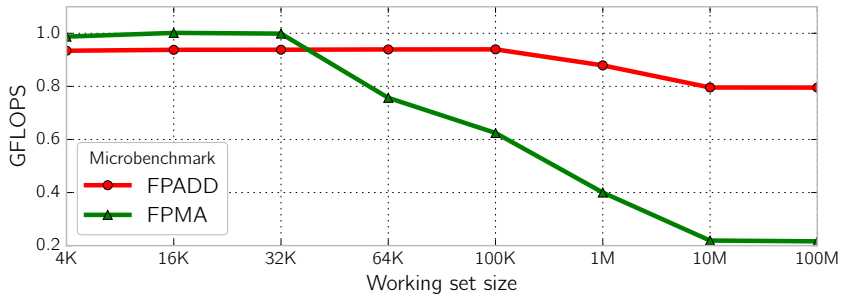



**Figure 4.3:** Exploration of the ARM Cortex-A9 double-precision floating-point pipeline for FADD and FMAC instructions with microbenchmarks. Peak double-precision performance is 1 GFLOPS @ 1 GHz.

benchmarks perform dense double-precision floating-point computation with accumulation on arrays of a given size (input parameter) stressing the FADD and FMAC instructions in a loop. We exploit data reuse by executing the same instruction multiple times on the same elements within one loop iteration. This way we reduce loop condition testing overheads and keep the floating-point pipeline as utilized as possible. The purpose is to evaluate if the ARM Cortex-A9 pipeline is capable of achieving the peak performance of 1 FLOP per cycle. Our results show that the Cortex-A9 core achieves the theoretical peak double-precision floating-point performance when the microbenchmark working set fits in the L1 cache (32 KB). Further, as we increase working set size beyond the size of L1 cache, FMAC instruction pipeline cannot sustain the performance at the same rate as FADD pipeline due to the higher pressure on the memory subsystem.

### 4.3.1 Algebra Backend

HPC applications usually depend on vendor provided algebraic backends, helping in achieving optimal performance by efficiently utilizing on-chip resources, because even the best compiler can not compete with the level of performance possible from a hand-optimized library. In the case of the most common HPC processor architecture today, x86, Intel provides MKL (Math Kernel Library) which accelerates math processing routines. In addition to linear algebra, it provides FFT, vector math and statistics functions.

On the other hand, ARM architecture used to lack such a support. Thus, we had to rely on open-source portable libraries. For algebraic backend we opted for ATLAS [131] – highly portable auto-tuned linear algebra software. Auto-tuning process tries to determine cache hierarchy and its properties, whether a platform has FMAC pipeline or separate FADD and FMUL pipes,

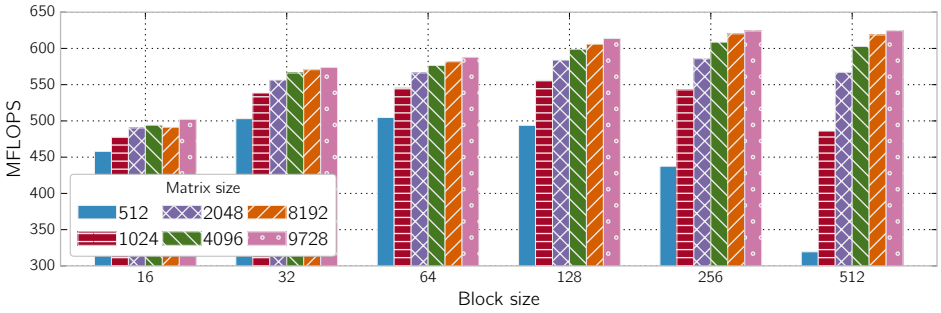



**Figure 4.4:** Performance of HPL on ARM Cortex-A9 for different input matrix and block sizes.

required level of loop unrolling, blocking factors – and in turn adjusts the routines to underlying hardware. This is an iterative process, and it took 26 hours on ARM Cortex-A9 to explore full parameter space.[5]

A widely used benchmark for measuring floating-point performance of HPC systems is HPL, which requires a BLAS (Basic Linear Algebra Subprograms) library backend. With Figure 4.4 we depict the performance of HPL on ARM Cortex-A9, with the ATLAS library providing required BLAS routines. Our results show that the best results on ARM Cortex-A9 processor core are achieved for the biggest possible inputs – which is in line with reported HPL performance [47]. Importantly, Figure 4.4 clearly shows how small values of block sizes affect the performance since the small values are lowering data reuse from the highest levels of the memory hierarchy. Finally, ARM Cortex-A9 core achieves 625 MFLOPS, leading to 62.5% HPL efficiency.

## 4.4 Comparison Against a Contemporary x86 Processor

In this section we present a comparison between an ARM Cortex-A9 platform against a contemporary x86 processor, power optimized Intel Core-i7 M640[6] (see Table 4.1).

For single core comparison of both performance and energy, we use Dhrystone [130], STREAM [90], and 17 benchmarks from SPEC CPU2006 [68]

[5]ATLAS is able to auto-tune itself in a shorter time. For example, auto-tuning for an Intel Nehalem x86 platform took 22 minutes. This difference is due to a narrower search space for a well established architecture. Support for ARM architecture within ATLAS was limited to Cortex-A8, with NEON co-processor.

[6]Intel Nehalem microarchitecture.

**Table 4.1:** Experimental platforms: comparison of the ARM Cortex-A9 against its contemporary x86 processor.

| | ARM Platform | Intel Platform |
|---|---|---|
| **SoC** | Tegra 2 | Intel Core i7-640M |
| **Architecture** | ARM Cortex-A9 (ARMv7-a) | Nehalem |
| **Core Count** | Dual core | Dual core |
| **Operating Frequency** | 1 GHz | 2.8 GHz |
| **Cache size(s)** | L1:32 KB I, 32KB D per core<br>L2: 1 MB I/D shared | L1: 32KB I, 32KB D per core<br>L2: 256 KB I/D per core<br>L3: 4 MB I/D shared |
| **RAM** | 1 GB DDR2-667<br>32-bit single channel<br>2666.67 MB/s per channel | 8 GB DDR3-1066<br>64-bit dual channel<br>8533.33 MB/s per channel |
| **Compiler** | GCC 4.6.2 | GCC 4.6.2 |
| **OS** | Linux 2.6.36.2 (Ubuntu 10.10) | Linux 2.6.38.11 (Ubuntu 10.10) |

benchmark suite[7].

Both platforms, ARM Cortex-A9 developer kit and a power optimized Intel Core i7 laptop, execute benchmarks with the same input set sizes in order to establish fair comparison conditions. Both platforms run GNU/Linux OS and use GCC v4.6 compiler. We measure power consumption at AC socket connection point for both platforms, and calculate energy-to-solution by integrating power samples. Due to the different natures of the laptop and the development board, and in order to provide a fair comparison in terms of energy efficiency, we measure only the power of components that are necessary for execution of the benchmarks, so all unused devices are disabled. On the ARM Cortex-A9 platform, we disable Ethernet during the benchmarks execution. On the Intel Core-i7 platform, graphics output, sound card, touch-pad, bluetooth, WiFi, and all USB (Universal Serial Bus) peripherals are disabled, and the corresponding modules are unloaded from the kernel. Also, the hard drive is spun down, and the Ethernet is disabled during the execution of the benchmarks. Multithreading could not be disabled, but all experiments are single-threaded and we set their logical core affinity in all cases. Benchmarks are compiled with -O3 level of optimization using GCC v4.6.2 compiler suite on both platforms.

### 4.4.1 Results

In terms of performance, in all the tested single-core benchmarks, the Intel Core i7 outperforms ARM Cortex-A9 core, as expected given the obvious design differences.

Table 4.2 shows the comparison between two platforms. In the case of

[7]Software stack was immature and it was not possible to port entire benchmark suite.

**Table 4.2:** Performance and energy-to-solution comparison between Intel Core i7-640M and ARM Cortex-A9 with Dhrystone and STREAM benchmarks

| Platform | Dhrystone | | | STREAM | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | *perf* | *energy* | | *perf (MB/S)* | | | | *energy (avg.)* | |
| | *(DMIPS)* | *abs (J)* | *norm* | *copy* | *scale* | *add* | *triad* | *abs (J)* | *norm* |
| Intel Core i7 | 19246 | 116.8 | 1.056 | 6912 | 6898 | 7005 | 6937 | 481.5 | 1.059 |
| ARM Cortex-A9 | 2213 | 110.8 | 1.0 | 1377 | 1393 | 1032 | 787 | 454.8 | 1.0 |

Dhrystone, Core i7 performs better by a factor of nine, but ARM platform uses 5% less energy to execute the benchmark. If we factor these results for the frequency difference, we get that ARM Cortex-A9 has 3.1× lower performance/MHz. Similarly, in the case of STREAM, Core i7 provides five times better performance but ARM platform uses 5% less energy to do the same amount of memory intensive work. In this case, the memory bandwidth comparison is not just a core microarchitecture comparison because achievable memory bandwidth is also dependant on the memory subsystem. However, bandwidth efficiency metrics which shows the achieved bandwidth out of the theoretical peak, shows to what extent the core, cache hierarchy, and on-chip memory controller are able to exploit off-chip memory bandwidth. We use the largest working set that could be fit into the both platforms, which is 800MB. Our results indicate that ARM Cortex-A9 platform is almost as balanced as Intel Core-i7 (40.5 vs 50.6 % bandwidth efficiency) for the simple *copy* kernel. However, in the case off *add* kernel, ARM Cortex-A9 drops its bandwidth efficiency compared to Intel Core-i7 platform – 27 vs 41 %[8].

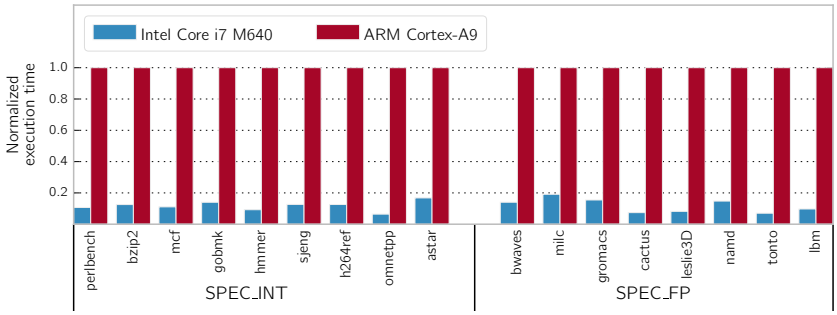
In the case of SPEC CPU2006 suite (Figure 4.5), Intel Core i7 core is significantly faster than ARM Cortex-A9 (up to 10 times), but at the same time, ARM platform uses less power resulting in 1.2 times smaller energy-to-solution (on average).

When frequency difference of the two platforms is factored out from the performance, assuming linear scalability of performance with the frequency, the difference in performance of the two platforms becomes smaller: Intel Core i7 M640 core would be only up to 3.2 times faster.

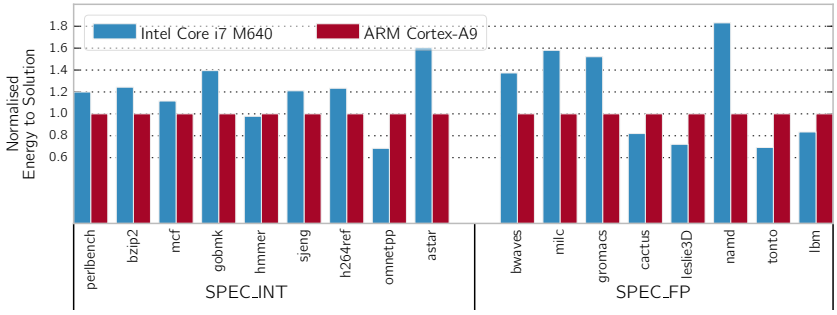
### 4.4.2 Discussion

Experiments presented in this chapter indicate that an ARM Cortex-A9 core is considered 3.2 times slower than an Intel Core i7 M640 core on per cycle basis. This means that we need to compensate for the performance difference utilizing higher mobile cores count, keeping the operating frequency intact.

[8]Similar trend is also observed for the *triad* kernel.




**(a)** Performance




**(b)** Energy to solution

**Figure 4.5:** Comparison between Intel Core i7-640M and ARM Cortex-A9 with SPEC CPU2006 benchmarks: a) execution time and b) energy-to-solution results. All results are normalized to ARM Cortex-A9

This is why we decided to design and deploy a cluster of ARM Cortex-A9 processors, named Tibidabo, and to evaluate its capability to exploit more cores in order to compensate for performance difference, and to investigate whether we could still maintain advantage of energy-efficiency. We deal with this topic in the following chapter.

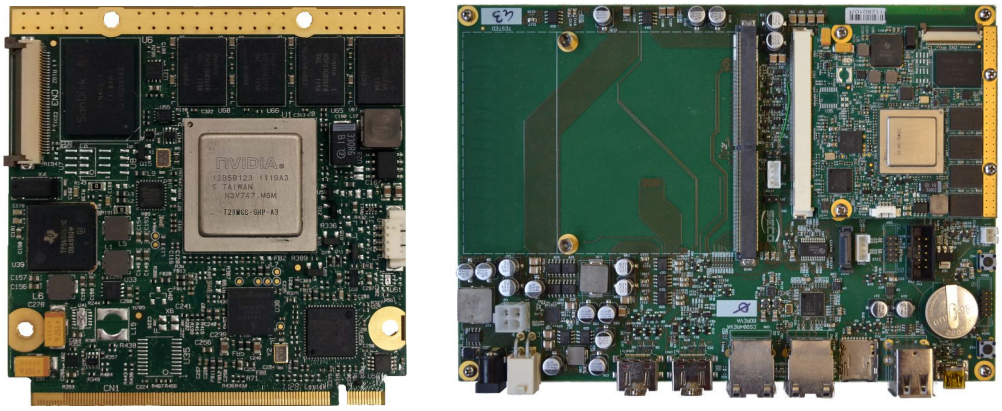
# 5 Tibidabo, The First Mobile HPC Cluster

In this chapter we present Tibidabo – the world first HPC cluster that we designed, built, and deployed usgin mobile SoCs. We present its architecture, software stack, and assess its performance and energy efficiency. Further, we compare Tibidabo against a contemporary x86 based cluster when running state-of-the-art PDE (Partial Differential Equations) solvers on the both. Finally, we finish the study with the projections of the performance and energy efficiency of future mobile SoCs powered systems.

## 5.1 Architecture

The computing chip in the Tibidabo cluster is the NVIDIA Tegra2 SoC, which integrates a dual-core ARM Cortex-A9[1] running at 1 GHz and implemented using TSMC's 40nm LPG (Low-power triple Gate oxyde) performance optimized process. NVIDIA Tegra2 features a number of application-specific accelerators targeted at the mobile market, such as video and audio encoder/decoder, and image signal processor, but none of these can be used for general-purpose computation and only contribute as a SoC area overhead in an HPC scenario. The GPU in Tegra2 does not support general programming models such as CUDA or OpenCL, but only OpenGL ES and cannot

[1] Evaluated in the previous chapter.



**(a)** Q7 module **(b)** Q7 carrier board

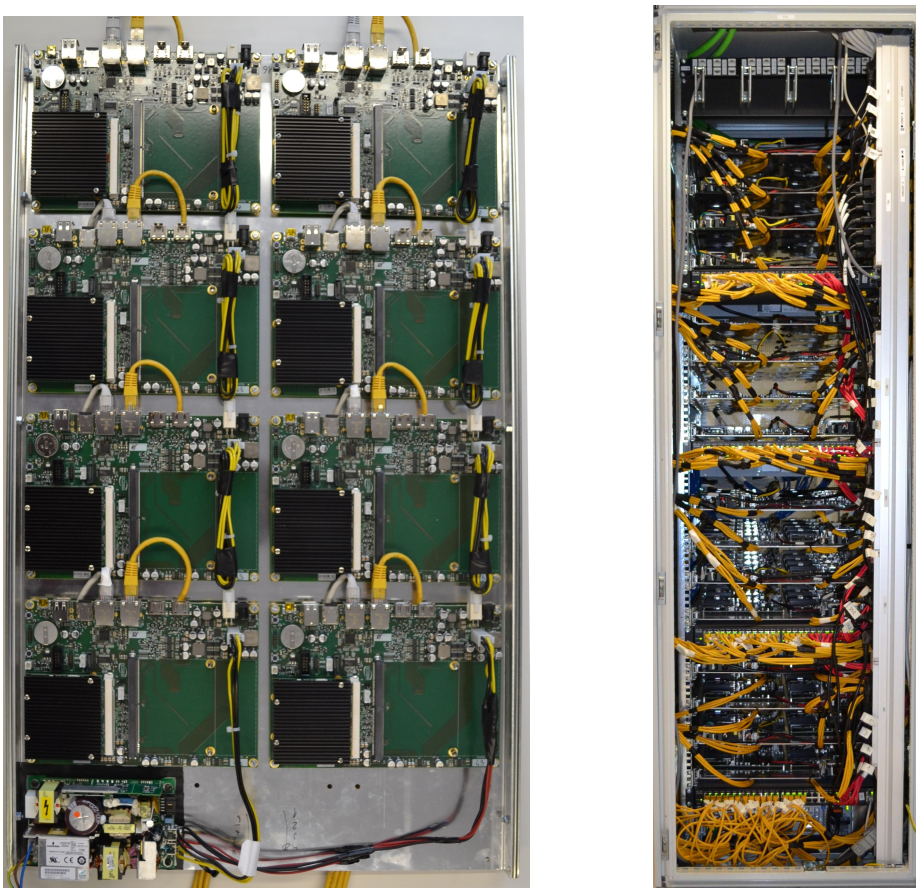
**Figure 5.1:** Tibidabo prototype: physical view of a) the node card and b) the node motherboard with a populated node card.

be used for HPC efficiently. However, more advanced GPUs actually support these programming models and a variety of HPC systems use them to accelerate certain workloads.

NVIDIA Tegra2 is the central part of the Q7 module [115] (See Figure 5.1a). The module also integrates 1 GB of DDR2-667 memory, 16 GB of eMMC storage, and exposes Tegra's USB and PCIe interfaces to the carrier board. Use of Q7 modules allows for a potential easy upgrade when next-generation SoCs become available, and reduces the cost of replacement in case of failure.

Each Tibidabo node is built using Q7-compliant carrier boards [116] (See Figure 5.1b). Each board hosts one Q7 module, integrates one 1GbE NIC (connected to Tegra2 through PCIe), one 100MbE NIC (connected to Tegra2 through USB), $\mu$SD card adapter and exposes other connectors and related circuitry that are not required for our HPC cluster, but are required for embedded software/hardware development (RS232, HDMI, USB, SATA, embedded keyboard controller, compass controller, etc.).

These boards are organized into blades (See Figure 5.2a), and each blade hosts 8 nodes and a shared Power Supply Unit (PSU). In total, Tibidabo has 128 nodes and it occupies 42 U standard rack space: 32 U for compute blades, 4 U for interconnect switches and 2 U for the file server.



(a) Blade with 8 boards (b) Tibidabo rack

**Figure 5.2:** Tibidabo prototype: blade and rack physical view.

## 5.2 Software Stack

Tibidabo runs Ubuntu 10.10 on top of Linux Kernel 2.6.32.2. We use SLURM for job management and scheduling. Regarding MPI runtime, Tibidabo relies on MPICH v1.4.1. At the time of Tibidabo deployment only MPICH reliably worked when integrated with SLURM. Applications that need algebraic backend rely on ATLAS library [131]. Those requiring an optimized FFT backend, use FFTW [54]. We chose ATLAS and FFTW since there were no existing vendor tuned libraries for the given purposes, as explained in previous chapter. Despite the auto-tuning architecture of the both, we had to port them to our chip architecture – to properly describe underlying hardware, and to describe timer routines – in order to achieve the most optimal performance.

## 5.3 Evaluation

In this section we present a parallel performance and energy-efficiency evaluation of Tibidabo cluster. We also provide a break down of a single node power consumption in order to understand which are the major power sinks in our cluster design.

### 5.3.1 Methodology

For the measurement of energy efficiency (MFLOPS/W), we used Yokogawa WT230 power meter [98] with an *effective sampling rate*[2] of 10 Hz, a basic precision of 0.1%, and RMS (Root-Means-Square) output values given as voltage/current pairs. We repeat our runs to get at least an acquisition interval of 10 minutes. The meter is connected to act as an AC supply bridge and to directly measure power drawn from the AC line. We have developed a measurement daemon that integrates with the OS and triggers the power meter to start collecting samples when the benchmark starts, and to stop when it finishes. Collected samples are then used to calculate the energy-to-solution and energy efficiency. To measure the energy efficiency of the whole cluster, the measurement daemon is integrated within the SLURM [133] job manager, and after the execution of a job, power measurement samples are included alongside the outputs from the job. In this case, the measurement point is on the power distribution unit of the entire rack.

### 5.3.2 Cluster Performance

Our single-core performance evaluation presented in the previous chapter shows that the ARM Cortex-A9 is ∼9 times slower than the Intel Core i7 640M at their maximum operating frequencies, which means that we need our applications to exploit a minimum of 9 parallel processors in order to achieve a competitive time-to-solution. More processing cores in the system means a higher demand for scalability. In this section we evaluate the performance, energy efficiency and scalability of the whole Tibidabo cluster.

Figure 5.3 shows the parallel speedup achieved by the (HPL) [47] benchmark and several other HPC applications. Following common practice, we perform a weak scalability test for HPL and a strong scalability test for

[2]Internal sampling frequencies are not known. This is the frequency at which the meter outputs new pairs of samples.

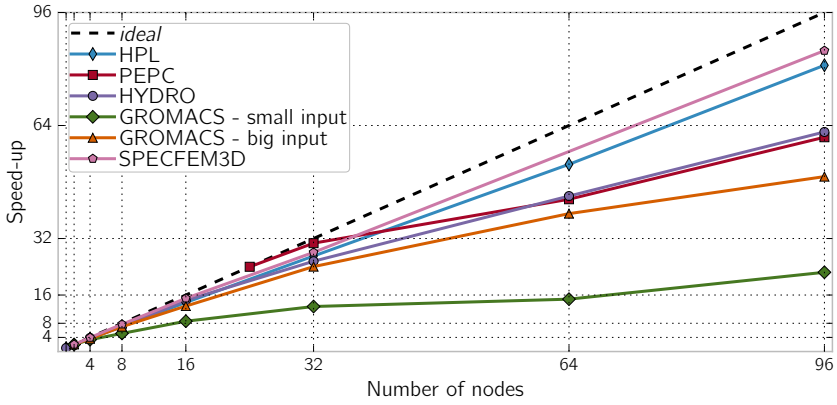



**Figure 5.3:** Tibidabo prototype: scalability of HPC applications.

the rest.[3] We have considered several widely used MPI applications: GROMACS [28], a versatile package to perform molecular dynamics simulations; SPECFEM3D_GLOBE [83] that simulates continental and regional scale seismic wave propagation; HYDRO, a 2D Eulerian code for hydrodynamics; and PEPC [132], an application that computes long-range Coulomb forces for a set of charged particles. All applications are compiled and executed out-of-the-box, without any manual tuning of the respective source codes.

If the application could not execute on a single node due to large memory requirements, we calculated the speedup with respect to the smallest number of nodes that can handle the problem. For example, PEPC with the reference input set requires at least 24 nodes, so we plot the results assuming that on 24 nodes the speedup is 24.

We have executed SPECFEM3D and HYDRO with an input set that is able to fit into the memory of a single node, and they show good strong scaling up to the maximum available number of nodes in the cluster. In order to achieve good strong scaling with GROMACS, we have used two input sets, both of which can fit into the memory of two nodes. We have observed that scaling of GROMACS improves when the input set size is increased. PEPC does not show optimal scalability because the input set that we can fit in our cluster is too small to show the strong scalability properties of the application [132].

HPL shows good weak scaling. In addition to HPL performance, we also measure power consumption, so that we can derive the MFLOPS/W met-

[3]Weak scalability refers to the capability of solving a larger problem size in the same amount of time using a larger number of nodes (the problem size is limited by the available memory in the system). On the other side, strong scalability refers to the capability of solving a fixed problem size in less time while increasing the number of nodes.

ric used to rank HPC systems in the Green500 list. Our cluster achieves 120 MFLOPS/W (97 GFLOPS on 96 nodes - 51% HPL efficiency), competitive with AMD Opteron 6128 and Intel Xeon X5660-based clusters, but 19x lower than the most efficient GPU-accelerated systems, and 21x lower than Intel Xeon Phi (November 2012 Green500 #1). The reasons for the low HPL efficient performance include lack of architecture-specific tuning of the algebra library, and lack of optimization in the MPI communication stack for ARM cores using Ethernet.

### Single Node Power Consumption Breakdown

In this section we analyze the power of multiple components on a compute node. The purpose is to identify the potential causes of inefficiency—on the hardware side—that led to the results in the previous section.

We were unable to take direct power measurements of the individual components in the Q7 card and carrier board, so we checked them in the provider specifications for each of the components. The CPU core power consumption is taken from the ARM website [8]. For the L2 cache power estimate, we use power models of the Cortex-A9's L2 implemented in 40nm and account for long inactivity periods due to the 99% L1 cache hit rate of HPL (as observed with performance counters reads). The power consumption of the NICs is taken from the respective datasheets [78, 118]. For the DDR2 memory, we use Micron's spreadsheet tool [92] to estimate power consumption based on parameters such as bandwidth, memory interface width, and voltage.

Figure 5.4 shows the average power breakdown of the major components in a compute node over the total compute node power during an HPL run on the entire cluster. As can be seen, the total measured power on the compute node is significantly higher than the sum of the major parts. Other on-chip and on-board peripherals in the compute node are not used for computation so they are assumed to be shut off when idle. However, the large non-accounted power part (labeled as OTHER) accounts for more than 67% of the total power. That part of the power includes on-board Low-Dropout (LDO) voltage regulators, on-board multimedia devices with related circuitry, corresponding share of a blade PSU losses and on-chip power sinks. Figure 5.5 shows the Tegra2 chip die. The white outlined area shows the chip components that are used by in an HPC cluster environment. This area is less than 35% of the total chip area. If the rest of the chip area is not properly power and clock gated, it would leak power even though it is not being used, thus also contributing to the OTHER part of the compute node power.

Although the estimations in this section are not exact, we actually overestimate the power consumption of some of the major components when taking

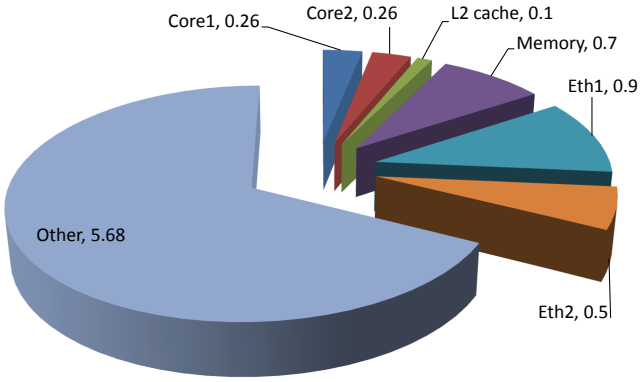



**Figure 5.4:** Tibidabo prototype: power consumption breakdown of main components on a compute node. 'Other' fraction represents non-accounted part including integration power-losses and unaccounted components power sinks. The compute node power consumption while executing HPL is 8.4 W. This power is computed by measuring the total cluster power and dividing the power by the total number of nodes.

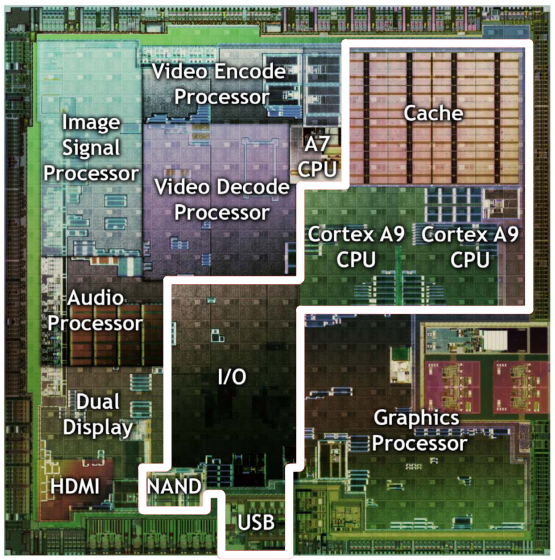



**Figure 5.5:** NVIDIA Tegra2 die photo: the area marked with white border line comprises the on-chip mobile SoC components actually used in an HPC scenario. It represents less than a 35% of the total chip area. *source [101]*

the power from the multiple data sources. We use either *typical* or *maximum* power consumption, whichever is available in components datasheets. Therefore, our analysis shows that up to 16% of the power is spent on the computation components: cores (including on-chip cache-coherent interconnect and L2 cache controller), L2 cache, and memory. The remaining 84% or more is then the overhead (or system *glue*) to interconnect those computation components with other computation components in the system. The reasons for this significant power overhead is the small size of the compute chip (two cores), and use of development boards targeted to embedded and mobile software development.

The conclusions from this analysis are twofold. HPC-ready carrier boards should be stripped-out of unnecessary peripherals to reduce area, cost and potential power sinks/wastes. And, at the same time, the computation chips should include a larger number of cores: less boards (nodes) are necessary to integrate the same number of cores, and the power overhead of a single compute chip is distributed among a larger number of cores. This way, the power overhead should not be a dominant part of the total power but just a small fraction.

### 5.3.3 Interconnect

In this section we present the evaluation of a Tibidabo node's NIC, aiming to discover whether a large overhead is introduced by the TCP/IP (Transmission Control Protocol/Internet Protocol) software stack. We therefore compare TCP/IP with a direct communication Ethernet protocol called Open-MX [61]. Open-MX is a high-performance implementation of the Myrinet Express message-passing stack that works over ordinary Ethernet networks. It integrates seamlessly with the OpenMPI and MPICH message passing libraries.

Open-MX bypasses the heavyweight TCP/IP stack and reduces the number of memory copies as much as possible thus reducing latency, CPU load, and cache pollution. For large messages, over 32KB, it uses rendezvous and memory pinning to achieve zero copy on the sender side and single copy on the receiver side. All actual communication management is implemented in the user-space library, with a kernel driver that takes care of initialization, memory registration, and passing and receiving raw Ethernet messages.

The latency and bandwidth results were measured using the PingPong benchmark from the Intel MPI Benchmark suite [76]. The PingPong benchmark measures the time to exchange one message of a given size between two MPI processes and reports bandwidth and latency. We map the processes on two different nodes, measuring inter-node latency and bandwidth. Figure 5.6

shows the results, for (a,b) two SECO Q7 boards with Tegra 2 at 1GHz, (c,d) two Arndale boards with Exynos 5 at 1.0GHz, and (e,f) two Arndale boards at 1.4GHz. In all cases we used 1GbE links; on SECO boards the network controller is connected via PCI Express and on Arndale it is connected via a USB 3.0 port.

From Figure 5.6a, it can be seen that the latency of Tegra 2 with TCP/IP is around 100$\mu$s, which is large compared to today's top HPC systems. When Open-MX is used, the latency drops to 65$\mu$s. Arndale running at 1GHz shows a higher latency (Figure 5.6c), of the order of 125$\mu$s with TCP/IP and 93$\mu$s when Open-MX is used. When the frequency is increased to 1.4GHz (Figure 5.6e), latencies are reduced by 10%.

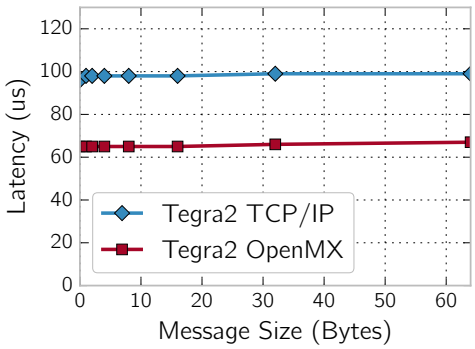
Although the Exynos 5 provides better performance than Tegra 2, all network communication has to pass through the USB software stack and this yields higher latency, both with MPI and Open-MX. When the frequency of the Exynos 5 SoC is increased, the latency decreases, which indicates that a large part of the overhead is caused by software, rather than the network controller and the network itself. Hence, it is crucial to reduce this overhead either by using more agile software solutions, such as Open-MX, or by introducing hardware support to accelerate the network protocol. Some SoCs, such as Texas Instrument's KeyStone II [121] already implement protocol accelerators.

Figure 5.6 shows the effective network bandwidth achieved as a function of the message size. The maximum bandwidth that can be achieved on the 1GbE link is 125 MB/s, and with MPI over TCP/IP none of the platforms is achieving it. In this case Tegra 2 can achieve 65 MB/s, and Exynos 5 can achieve 63 MB/s – utilizing less than 60% of the available bandwidth. When Open-MX is employed, the situation improves significantly for Tegra 2, now capable of reaching 117 MB/s – 93% of the theoretical maximum bandwidth of the link. Due to the overheads in the USB software stack, Exynos 5 exhibits smaller bandwidth than Tegra 2, but with an improvement over TCP/IP: 69 MB/s running at 1GHz and 75 MB/s running at 1.4GHz.

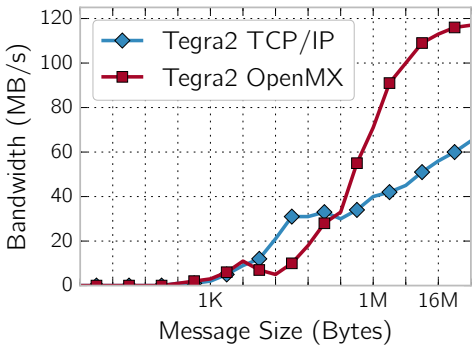
## 5.4 Comparison Against an X86-Based Cluster[4]

In this section we present a study which aimed on comparing Tibidabo against an x86-based production system in terms of performance and en-

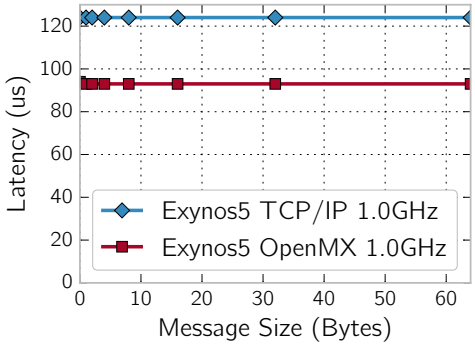
[4]Parts of this section have previously been published as a result of collaboration between BSC, TU Dortmund and CNRS [66]



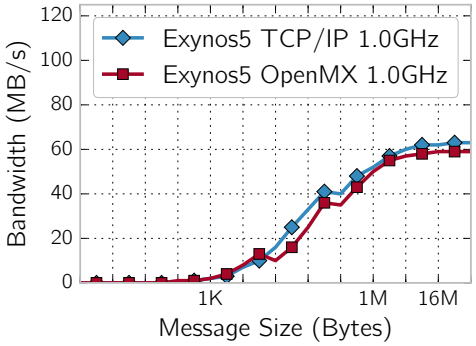

(a) Tegra2 latency



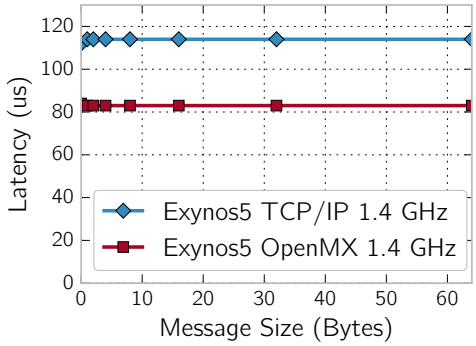

(b) Tegra2 bandwidth



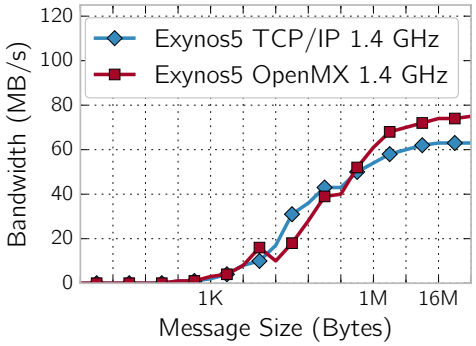

(c) Exynos5 @1GHz, latency




(d) Exynos5 @1GHz, bandwidth




(e) Exynos5 @1.4GHz, latency




(f) Exynos5 @1.4GHz, bandwidth

**Figure 5.6:** Interconnect measurements: influence of CPU performance on achievable MPI bandwidth and latency. All data acquired running MPI Ping-Pong benchmark.

ergy consumption. For performance metrics we look into *Time-to-solution*, whereas for energy we measure *energy-to-solution* – power integrated over runtime.

### 5.4.1 Reference x86 System

The system we compare Tibidabo against represents a partition of LiDOng system installed at TU Dortmund[5]. Each LiDOng node comprises a dual-socket quad-core Intel Xeon X5550 processor running at 2.66GHz. Nodes are populated with 16GB of DDR3 memory (eight 2GB memory DIMMs), and feature two 1Gbit Ethernet NICs corresponding to two separate networks. The first network is utilized for MPI, while the second network provides access to storage through Lustre file system. The MPI network switches are implemented as full crossbars.

### 5.4.2 Applications

For this comparative study we employ three PDE (Partial Differential Equations) applications, namely FEAST [123], HONEI_LBM, and SPECFEM3D_GLOBE[35]. FEAST is a PDE toolkit which provides finite-element discretization and multilevel solvers. For the purpose of this study, we do not execute full applications but we employ FEAST as an application proxy solving a standard Poisson problem. HONEI_LBM is a parallel Computational Fluid Dynamics (CFD) solver, built on top of the portable HONEI libraries [125]. SPECFEM3D_GLOBE is a well known petascale-ready code which simulates three-dimensional seismic wave propagation. Regarding the floating-point precision, FEAST requires double precision, while HONEI_LBM and SPECFEM3D_GLOBE run in single-precision.

### 5.4.3 Power Acquisition

Both systems' power consumptions are monitored on a single node, excluding power spent on the network. In both systems we assume equal power consumption distribution over the nodes taking a part of the same run. This is obvious in the case of Tibidabo since we schedule 2 processes per node, and the total number of MPI ranks is even.

Process scheduling on LiDOng is done in the same manner – if the MPI process scheduling produces idle cores, they are equally distributed among the nodes.

[5]This evaluation was conducted during 2012, current system specifications may have changed due to upgrades.

### 5.4.4 Input Configurations

The Tibidabo cluster features only 1GB of memory per node, out of which only ∼800MB could be utilized by user processes. Thus, the memory footprint of the input for both systems is constrained by the maximum achievable input on Tibidabo. On the other side, LiDOng has 16GB of memory per node, two sockets and eight cores, allowing for evaluation and comparison of the different mappings of MPI processes on compute nodes. Executions on Tibidabo are scheduled to use 2 cores per node, and we employ weak scaling as we increase the number of processes (nodes). In the case of LiDOng, we evaluate three different mappings regarding the scheduling of MPI processes with constraints on memory and computing capacity, keeping the total input size per configuration the same on both systems:

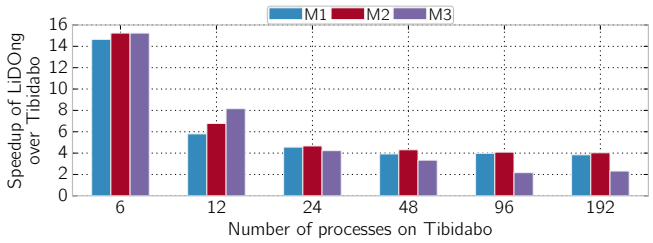
- **M1**: In this mapping both systems distribute the work utilizing the same amount of *per core* memory, in turn leading to the same number of MPI processes on the both systems. Note that this mapping does not use all available cores on LiDOng nodes – it uses 6 out of 8.

- **M2**: LiDOng system utilizes maximum number of cores per node, and the same number of nodes as in *M1* mapping for each tested configuration. This leads to the more processes compared to the *M1* mapping.

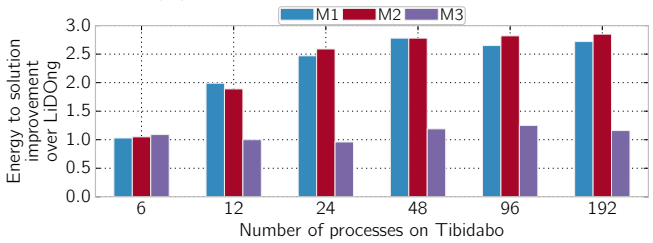
- **M3**: This mapping fits the input to as few nodes of LiDOng as possible. Thus, this configuration yields smaller number of MPI processes compared to both *M1* nad *M2* mapping.

### 5.4.5 Results

In this section we present performance and energy-to-solution comparative figures of Tibidabo and LiDOng systems when running applications listed in Section 5.4.2. Note that when reporting speedup it shows how many times is a given execution faster on LiDOng compared to Tibidabo, and contrary when reporting energy-to-solution – we show improvement of Tibidabo compared to LiDOng in terms of energy savings.




**(a)** Speedup over Tibidabo.

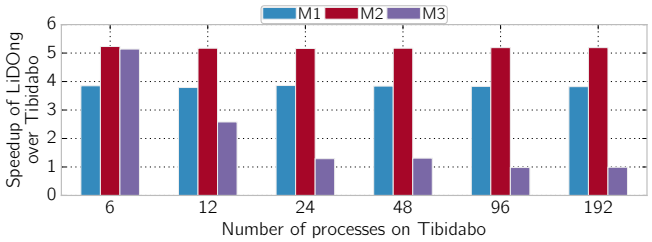



**(b)** Energy to solution improvement over LiDOng. Higher is better.

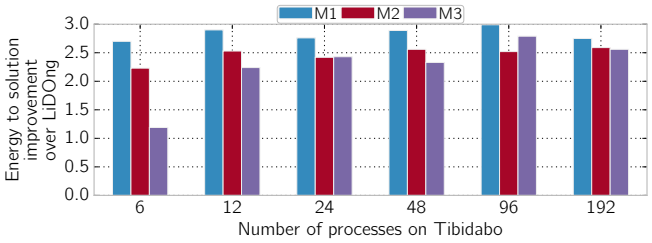
**Figure 5.7:** Performance and energy comparison between Tibidabo prototype and its contemporary x86 cluster with FEAST application: a) speedup b) energy.

The results of comparison of Tibidabo and LiDOng systems when running FEAST application are depicted in Figure 5.7. Performance wise (see Figure 5.7a), LiDOng experiences speedup over Tibidabo in the range of 2 – 15×, depending on the input size and actual mapping on LiDOng. Highest performance advantage is achieved with scenarios M2 and M3 – when the input is scheduled on only one LiDOng node thus utilizing only shared memory mechanism of OpenMPI (the smallest input, 8 processes). However, for the largest input LiDOng is four times faster than Tibidabo with M2 mapping, and only two times faster with M3 mapping. Energy wise (see Figure 5.7b, Tibidabo is more energy efficient across all input sizes and mappings. Under worst case running time Tibidabo manages to achieve 3% energy savings over LiDOng. In the most favorable scenario for Tibidabo regarding runtime, M3 mapping on LiDOng and the biggest input, Tibidabo saves 16% of energy. Finally, Tibidabo consumes less energy than LiDOng system for all inputs and corresponding mappings listed in Section 5.4.4.

Looking into the results for HONEI_LBM, shown in Figure 5.8, results exhibit similar trends. Performance wise (see Figure 5.8a), for the same amount of memory per core, LiDOng is ∼4 times faster with all inputs. However, in the case of M2 mapping LiDOng experiences 5× speedup across all problem sizes. As we increase problem size, with M3 mapping LiDOng shows almost no speedup over Tibidabo for big problem sizes. However,




**(a)** Speedup over Tibidabo.



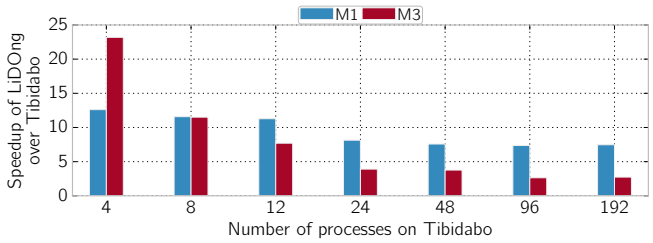

**(b)** Energy to solution improvement over LiDOng. Higher is better.

**Figure 5.8:** Performance and energy to solution comparison between Tibidabo prototype and LiDOng with HONEI_LBM application: a) speedup b) energy.

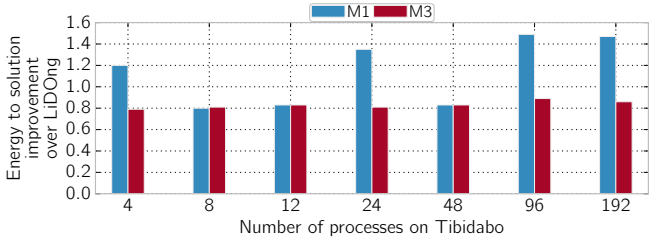
Tibidabo is more energy efficient across all configurations and inputs. The biggest energy savings are achieved for the M1 mapping, and the smallest for the M3 mapping: ranging from 3× to only 20% savings. The smallest energy savings are achieved for the smallest input, while the biggest savings are in place for the second biggest input.

SPECFEM3D_GLOBE reveals that there are even smaller differences for different mappings. Note, however, that for the M2 mapping it was not possible in this case due to the specific nature of the application. In the case of the execution time, speedup of LiDOng over Tibidabo is almost twice higher compared to the other two applications under evaluation. This has implications on the energy to solution, since Tibidabo is not always more energy efficient than LiDOng. The maximum energy savings of 50% are achievable for the biggest inputs, when M1 mapping is used.

With our experiments we have shown that depending on the actual input size and the number of computing processes, Tibidabo can offer even performance, or can be as 15× slower compared to the x86-based computing cluster on a set of PDE solvers. This performance difference is also reflected to potential energy savings Tibidabo could offer, going from 0.8 – 3× compared to aforementioned system. We have shown that Tibidabo becomes




**(a)** Speedup over Tibidabo.




**(b)** Energy to solution improvement over LiDOng. Higher is better.

**Figure 5.9:** Performance and energy to solution comparison between Tibidabo prototype and LiDOng with SPECFEM3D_GLOBE application: a) speedup b) energy.

competitive in the case when one tries to minimize the number of nodes for an execution – fitting the input size (memory requirement) of an application to the smallest possible number of nodes.

## 5.5 Projections

In this section we project what would be the performance and power consumption of our cluster if we could have set up an HPC-targeted system using the same low-power components. One of the limitations seen in the Section 5.3.2 is that having only two cores per chip leads to significant power consumption overheads due glueing them together in order to create a large-scale system with a large number of cores. At the time of the production of Tibidabo prototype Cortex-A9 was the leader in mobile computing. However, during the evaluation of the Tibidabo prototype, Cortex-A15 was announced as the highest performing processor in the ARM family, having features more suitable for HPC. Therefore, in this section we evaluate cluster configurations with higher multicore density (more cores per chip) and we also project what would be the performance and energy efficiency if we could have used Cortex-A15 cores instead. To complete the study, we evaluate multiple frequency operating points to show how frequency affects performance and energy effi-

**Table 5.1:** Estimation of performance and energy efficiency of potential Tibidabo prototype upgrades: core architecture, performance and power model parameters, and results for performance and energy efficiency of clusters with 16 cores per node.

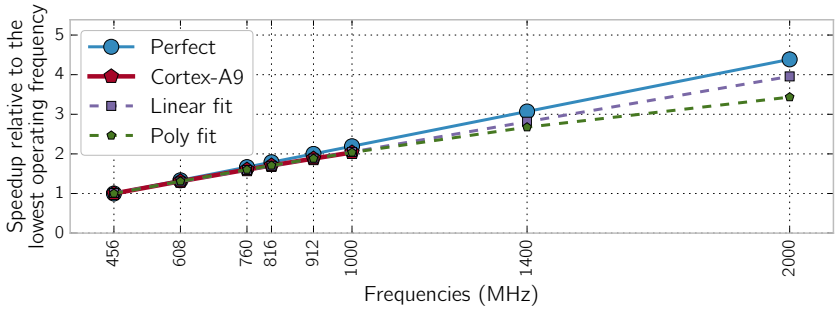
| **Configuration #** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|
| *CPU input parameters* | | | | | |
| **Core architecture** | Cortex-A9 | Cortex-A9 | Cortex-A9 | Cortex-A15 | Cortex-A15 |
| **Frequency (GHz)** | 1.0 | 1.4 | 2.0 | 1.0 | 2.0 |
| **Performance over** $A9_{1GHz}$ | 1.0 | 1.2 | 1.5 | 2.6 | 4.5 |
| **Power over** $A9_{1GHz}$ | 1.0 | 1.1 | 1.8 | 1.54 | 3.8 |
| *Per node power figures for 16 cores per chip configuration [W]* | | | | | |
| **CPU cores** | 4.16 | 4.58 | 7.49 | 7.64 | 18.85 |
| **L2 cache** | 0.8 | 0.88 | 1.44 | Integrated with cores | |
| **Memory** | 5.6 | 5.6 | 5.6 | 5.6 | 5.6 |
| **Ethernet NICs** | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 |
| *Aggregate power figures [W]* | | | | | |
| **Per node** | 17.66 | 18.16 | 21.63 | 20.34 | 31.55 |
| **Total cluster** | 211.92 | 217.87 | 259.54 | 244.06 | 378.58 |

ciency.

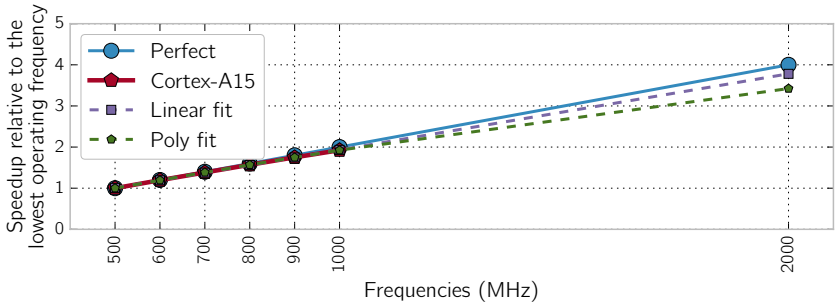
For our projections, we use an analytical power model and the Dimemas cluster simulator [22]. For more details on Dimemas simulator, please refer to the Chapter 3. The input to our simulations is an execution trace obtained from a 96 nodes HPL execution on the Tibidabo cluster.

Table 5.1 shows the parameters used to estimate the performance and energy efficiency of multiple cluster configurations. For this analysis, we use single-core HPL runs to measure the performance ratios among different core configurations. The reasons to use HPL are twofold: it makes heavy use of floating-point operations, as it happens in many HPC applications, and is the reference benchmark used to rank HPC machines both on Top500 and Green500 lists.

To project the performance scaling of HPL for Cortex-A9 designs clocked at frequencies over 1 GHz, we execute HPL in one of our Tibidabo nodes using two cores at multiple frequencies from 456 MHz to 1 GHz. Then, we fit a $2^{nd}$ order polynomial trend line on the performance points to project the performance degradation beyond 1 GHz. Figure 5.10a shows a performance degradation below 10% at 1 GHz compared to perfect scaling from 456 MHz. The polynomial trend line projections to 1.4 and 2.0 GHz show a 14% and 25% performance loss over perfect scaling from 456 MHz respectively. A polynomial trend line seems somewhat pessimistic if there are no fundamental architectural limitations, so we can use these projections as a lower bound




**(a)** Dual-core Cortex-A9




**(b)** Dual-core Cortex-A15

**Figure 5.10:** Performance of HPL on ARM Cortex-A9 and Cortex-A15 at multiple operating frequencies and extrapolation to frequencies beyond 1 GHz. Data points are normalized to the performance at the lowest operating frequency.

for the performance of those configurations.

For the performance of Cortex-A15 configurations we perform the same experiment on a dual-core Cortex-A15 test chip clocked at 1 GHz [11]. HPL performance on Cortex-A15 is 2.6 times faster compared to our Cortex-A9 Tegra2 boards. To project the performance over 1 GHz we run HPL at frequencies ranging between 500 MHz and 1 GHz and fit a $2^{nd}$ order $2^{nd}$ order polynomial trend line on the results. The performance degradation compared to perfect scaling from 500MHz at 2 GHz is projected to 14% (see Figure 5.10b so the performance ratio over Cortex-A9 at 1 GHz is 4.5x. We must say that these performance ratios of Cortex-A15 over Cortex-A9 are for HPL, which makes heavy use of floating-point code. The performance ratios of Cortex-A15 over Cortex-A9 for integer code are typically 1.5x at 1 GHz and 2.9x at 2 GHz (both compared to Cortex-A9 at 1 GHz). This shows how, for a single compute node, Cortex-A15 is better suited for HPC double-precision floating-point computation.

For the power projections at different clock frequencies, we are using a

power model for Cortex-A9 based on 40nm technology as this is what many Cortex-A9 products were using at the time of our study, and for the Cortex-A15 on 28nm technology as this is the process that is used for most product produced in 2013. The power consumption in both cases is normalized to the power of Cortex-A9 running at 1 GHz. Then, we introduce these power ratios in our analytical model to project the power consumption and energy efficiency of the different cluster configurations. In all our simulations, we assume the same number of total cores as in Tibidabo (192) and we vary the number of cores in each compute node. When we increase the number of cores per compute node, the number of nodes is reduced, thus, reducing integration overhead and pressure on the interconnect (i.e. less boards, cables and switches). To model this effect, our analytical model is as follows:

From the power breakdown of a single node presented in Figure 5.4, we subtract the power corresponding to the CPUs and the memory subsystem (L2 + memory). The remaining power in the compute node is considered to be *board overhead*, and does not change with the number of cores. The board overhead is part of the power of a single node, to which we add the power of the cores, L2 cache and memory. For each configuration, the CPU core power is multiplied by the number of cores per node. Same as in Tibidabo, our projected cluster configurations are assumed to have 0.5 MB of L2 cache per core and 500 MB of RAM per core—this assumption allows for simple scaling to large numbers of cores. Therefore, the L2 cache power (0.1 W/MB) and the memory power (0.7 W/GB) are multiplied both by half the number of cores. The L2 core power for the Cortex-A9 configurations is also factored for frequency, for which we use the core power ratio. The L2 in Cortex-A15 is part of the core macro, so the core power already includes the L2 power.

For both Cortex-A9 and Cortex-A15, the CPU macro power includes the L1 caches, cache coherence unit and L2 controller. Therefore, the increase in power due to a more complex L2 controller and cache coherence unit for a larger multicore are accounted when that power is factored by the number of cores. The memory power is overestimated, so the increased power due to the increased complexity of the memory controller to scale to a higher number of cores is also accounted for the same reason. Furthermore, a Cortex-A9 system cannot address more than 4 GB of memory so, strictly speaking, Cortex-A9 systems with more than 4 GB are not realistic. However, we include configurations for higher core counts per chip to show what would be the performance and energy efficiency if Cortex-A9 included *large physical address extensions* as the Cortex-A15 does to address up to 1 TB of memory [19].

The power model is summarized in these equations:

$$P_{pred} = \frac{n_{tc}}{n_{cpc}} \times \left( \frac{P_{over}}{n_{nin}} + P_{eth} + n_{cpc} \times \left( \frac{P_{mem}}{2} + p_r \times \left( P_{A9_{1G}} + \frac{P_{L2\$}}{2} \right) \right) \right) \quad (5.1)$$

$$P_{over} = P_{tot} - n_{nin} \times (P_{mem} + 2 \times P_{A9_{1G}} + P_{L2\$} + P_{eth}) \quad (5.2)$$

where $P_{pred}$ represents the projected power of simulated clusters, while $n_{tc} = 192$ and $n_{nin} = 96$ are constants and represent the total number of cores and total number of nodes in Tibidabo respectively. $n_{cpc}$ is the number of cores per chip. $P_{over}$ represents the total Tibidabo cluster power overhead (evaluated in Equation 5.2). Parameter $p_r$ defines the power ratio derived from core power models and normalized to Cortex-A9 at 1 GHz. $P_{A9_{1G}}$, $P_{mem}$, $P_{L2\$}$ and $P_{eth}$ are constants defining a core, per core memory, per core L2 cache and per node Ethernet power consumptions in Tibidabo. $P_{tot} = 808$ W is the average power consumption of Tibidabo while running HPL.

In our experiments, the total number of cores remains constant and is the same as in the Tibidabo cluster ($n_{tc} = 192$). We explore the total number of cores per chip ($n_{cpc}$) that, having one chip per node, determines the total number of nodes of the evaluated system. Table 5.1 shows the resulting total cluster power of the multiple configurations using 16 cores per chip, and a breakdown of the power for the major components.

For the performance projections of the multiple cluster configurations, we provide Dimemas with a CPU core performance ratio for each configuration, and a varying number of processors per node. Dimeams produces a simulation of how the same 192-core[6] application will behave based on the new core performance and multicore density, accounting for synchronizations and communication delays. Figure 5.11 shows the results. In all the simulations we keep a network bandwidth of 1 Gb/s (1GbE) and a memory bandwidth of 1400 MB/s (from the maximum memory bandwidth measured with STREAM).

The results show that, as we increase the number of cores per node (at the same time reducing the total number of nodes), performance does not show further degradation with 1GbE interconnect until we reach the level of performance of Cortex-A15. None of the Cortex-A15 configurations reach its maximum speedup due to interconnect limitations. The configuration

[6]Out of 128 nodes with a total of 256 processors, 4 nodes are used as login nodes and 28 are unstable. There are two major identified sources for instabilities: cooling issues and problems with the PCIe driver, which drops the network connection on the problematic nodes.

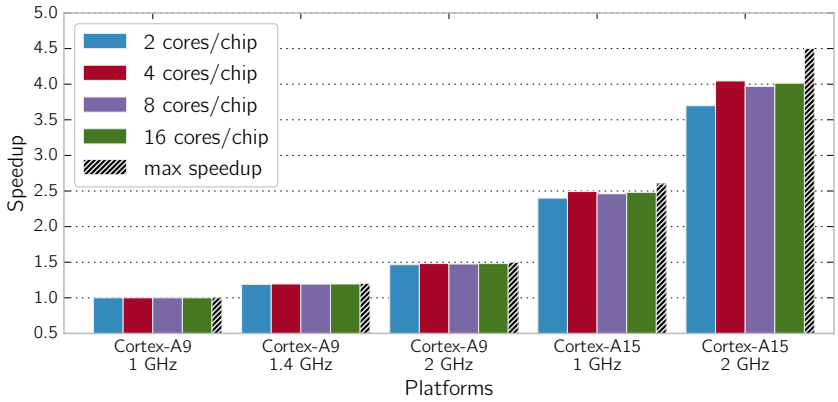



**Figure 5.11:** Tibidabo prototype: projected speedup for the evaluated cluster configurations. The total number of MPI processes is constant across all experiments.

with two Cortex-A15 cores at 1 GHz scales worse because the interconnect is kept the same as in Tibidabo. With a higher number of cores, we are reaching 96% of the speedup of a Cortex-A15 at 1 GHz. Further performance increase with Cortex-A15 at 2 GHz shows further performance limitation due to interconnect communication—reaching 82% of the maximum possible speedup (see Figure 5.11) with two cores and reaching 91% with sixteen.

Increasing computation density potentially improves MPI communication because more processes communicate on chip rather than using the network, and the memory bandwidth is larger than the interconnect bandwidth. Setting a larger machine than Tibidabo, with faster mobile cores and a higher core count, will require a faster interconnect. In Section 5.6 we explore the interconnect requirements when using faster mobile cores.

The benefit of increased computation density (more cores per node) is actually the reduction of the integration overhead and the resulting improved energy efficiency of the system (Figure 5.12). The results show that by increasing the computation density, with Cortex-A9 cores running at 2 GHz we can achieve an energy efficiency of 563 MFLOPS/W using 16 cores per node, which is already 4.7× improvement over Tibidabo. The configuration with 16 Cortex-A15 cores per node has an energy efficiency of 1004 MFLOPS/W at 1 GHz and 1046 MFLOPS/W at 2 GHz (8.7x improvement).

Using these models, we project the energy efficiency of our cluster if it used higher performance cores and included more cores per node. However, all other features remain the same, so inefficiencies due to the use of non-optimized development boards, lack of software optimization, and lack of vector double-precision floating-point execution units is accounted in the

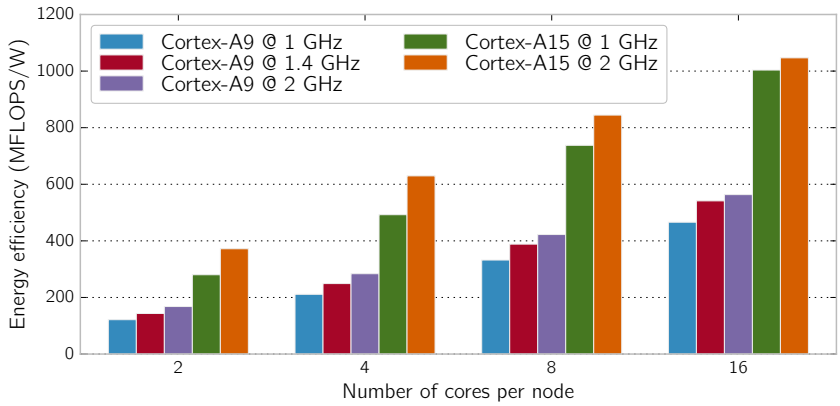



**Figure 5.12:** Tibidabo prototype: projected energy efficiency for the evaluated cluster configurations.

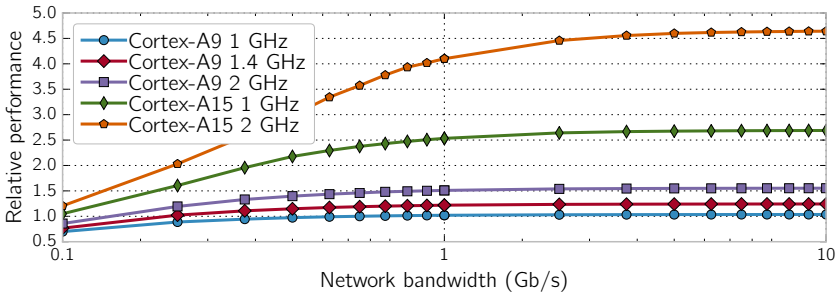
model. Still, with all these inefficiencies, our projections show that such a cluster would be competitive in terms of energy efficiency with Sandy Bridge and GPU-accelerated systems in the June 2013 Green500 list[7], showing a promise for future ARM-based platforms actually optimized for HPC.

## 5.6 Interconnect requirements

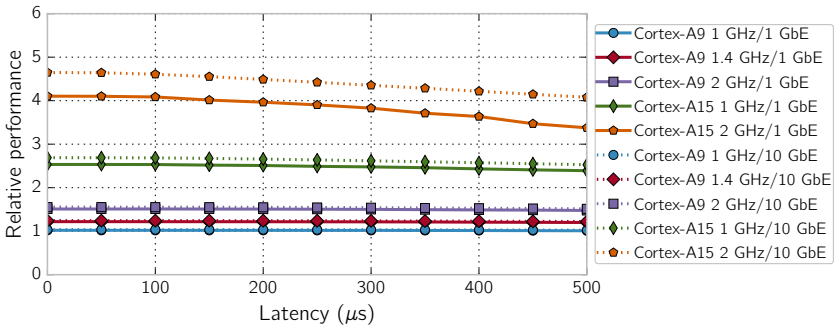
Cluster configurations with higher-performance cores and more cores per node, potentially impose a higher pressure on the interconnection network. The result of increasing the node computation power while maintaining the same network bandwidth is that the interconnect bandwidth-to-flops ratio decreases. This may lead to the network becoming a bottleneck. To evaluate this effect, we carried out simulations of the evaluated cluster configurations using a range of network bandwidths (Figure 5.13a) and latency values (Figure 5.13b). The baseline for these results is the cluster configuration with Cortex-A9 at 1 GHz, 1 Gb/s of bandwidth and 50 $\mu$s of latency.

The results in Figure 5.13a show that a network bandwidth of 1 Gb/s is sufficient for the evaluated cluster configurations with Cortex-A9 cores and the same size as Tibidabo. The Cortex-A9 configurations show a negligible improvement with 10 Gb/s interconnects. On the other hand, configurations with Cortex-A15 do benefit from an increased interconnect bandwidth: the 1 GHz configuration reaches its maximum at 3 Gb/s, and the 2 GHz configuration at 8 Gb/s.

[7]Here we refer to the homogeneous and heterogeneous systems placed from #40–50 and are based on Intel Xeon E5-2670 and NVIDIA Tesla 2090 GPUs




(a) Bandwidth sensitivity




(b) Latency sensitivity

**Figure 5.13:** Tibidabo prototype: interconnection network impact on extrapolated cluster upgrades. Only configurations with 16 cores per node are shown.

The latency evaluation in Figure 5.13b shows the relative performance with network bandwidths of 1 Gb/s and 10 Gb/s for a range of latencies with a 50 $\mu$s step. An ideal zero latency does not show a significant improvement over 50 $\mu$s latency and increasing the latency with a factor of ten, has a significant impact on the Cortex-A15 at 2 GHz configuration only. Therefore, the latency of Tibidabo's Ethernet network, although being larger than that of specialized and custom networks used in supercomputing, is low enough for all the evaluated cluster configurations which have the same size as Tibidabo. However, further evaluation is needed to study the effects of bandwidth and latency sensitivity with an increased number of nodes. Our simulation methodology did not allow us to explore this effect, but since recently, there is a new simulation methodology aiming to support this kind of design-space exploration [64].

### 5.6.1 Lessons Learned and Next Steps

In this chapter We have described the architecture of our Tegra2-based cluster, the first attempt to build an HPC system using ARM processors. Our

performance and power evaluation shows that an ARM Cortex-A9 platform is competitive with a mobile Intel Nehalem Core i7-640M platform in terms of energy efficiency for a reference benchmark suite like SPEC CPU2006. We have also demonstrated that, even without manual tuning, HPC applications scale well on our cluster.

However, building a supercomputer out of commodity-of-the-shelf low-power components is a challenging task because achieving a balanced design in terms of power is difficult. As an example, the total energy dissipated at the PCB (Printed Circuit Board) voltage regulators is comparable or even higher than the energy spent on the CPU cores. Although the core itself provides a theoretical peak energy efficiency of 2-4 GFLOPS/W, this design imbalance results in the measured HPL energy efficiency of 120 MFLOPS/W.

In order to achieve system balance, we identified two fundamental improvements to put in practice. The first one is to make use of higher-end ARM multicore chips like Cortex-A15, which provides an architecture more suitable for HPC while maintaining comparable single-core energy efficiency. The second one is to increase the computing density by adding more cores to the chip. The recently announced ARM CoreLink CCN-504 cache coherence network [10, 33] scales up to 16 cores and is targeted to high-performance architectures such as Cortex-A15 and next-generation 64-bit ARM processors. In a resulting system which implements these design improvements, the CPU cores power is better balanced with that of other components such as the memory. Our projections based on ARM Cortex-A15 processors with higher multicore integration density show that such systems are a promising alternative to current designs built from high performance parts. For example, a cluster of the same size as Tibidabo, based on 16-core ARM Cortex-A15 chips at 2 GHz would provide 1046 MFLOPS/W.

A well known technique to improve energy efficiency is the use of SIMD units. As an example, BlueGene/Q uses 256-bit-wide vectors for quad double-precision floating-point computations, and the Intel MIC (Many Integrated Cores) architecture uses 512-bit-wide SIMD units. Both Cortex-A9 and Cortex-A15 processors implement the ARMv7-a architecture which only supports single-precision SIMD computation. Most HPC applications require calculations in double-precision so they cannot exploit the current ARMv7 SIMD units. The ARMv8 architecture specification includes double-precision floating-point SIMD, so further energy efficiency improvements for HPC computation are expected from ARMv8 chips featuring those SIMD units.

In all of our experiments, we run the benchmarks out of the box, and did not manually tune any of those codes. Libraries and compilers include architecture-dependent optimizations that, for the case of ARM processors, target mobile computing. This leads to two different scenarios: the optimiza-

tions of libraries used in HPC, such as ATLAS or MPI, for ARM processors are one step behind; and optimizations in compilers, operating systems and drivers target mobile computing, thus trading-off performance for quality of service or battery life. We have put together an HPC-ready software stack for Tibidabo but we have not put effort in optimizing its several components for HPC computation yet. Further energy efficiency improvements are expected when critical components such as MPI communication functions are optimized for ARM-based platforms, or the Linux kernel is stripped-out of the components/modules not used by HPC applications.

As shown in Figure 5.5, the Tegra2 chip includes a number of application-specific accelerators that are not programmable using standard industrial programming models such as CUDA or OpenCL. If those accelerators were programmable and used for HPC computation, that would reduce the integration overhead of Tibidabo. The use of SIMD or SIMT (Single Instructions Multiple Threads) programmable accelerators is widely adopted in supercomputers, such as those including general-purpose programmable GPUs (GPGPUs). Although the effective performance of GPGPUs is between 40% and 60%, their efficient compute-targeted design provides them with high energy efficiency. GPUs in mobile SoCs are starting to support general-purpose programming. One example is the Samsung Exynos5 [114] chip, which includes two Cortex-A15 cores and an OpenCL-compatible ARM Mali T-604 GPU [20]. This design, apart from providing the improved energy efficiency of GPGPUs, has the advantage of having the compute accelerator close to the general purpose cores, thus reducing data transfer latencies. Such an on-chip programmable accelerator is an attractive feature to improve energy efficiency in an HPC system built from low-power components.

Another important issue to keep in mind when designing such kind of systems is that the memory bandwidth-to-flops ratio must be maintained. Currently available ARM-based platforms make use of either memory technology that is behind compared to top-class standards (e.g., many platforms still use DDR2, DDR3 memory instead of e.g. DDR4), or memory technology targeting low power (e.g., LPDDR2, LPDDR3). For a higher-performance node with a higher number of cores and including double-precision floating-point SIMD units, current memory choices in ARM platforms may not provide enough bandwidth, so higher-performance memories must be adopted. Low-power ARM-based products including DDR3 are already announced [34] and the recently announced DMC-520 [10] memory controller enables DDR3 and DDR4 memory for ARM processors. These upcoming technologies are indeed good news for low-power HPC computing. Moreover, *package-on-package* memories which reduce the distance between the computation cores and the memory, and increase pin density can be used to include several

memory controllers and provide higher memory bandwidth.

Finally, Tibidabo employs 1 Gbit Ethernet for the cluster interconnect. Our experiments show that 1GbE is not a performance limiting factor for a cluster of Tibidabo size employing Cortex-A9 processors up to 2 GHz and for compute-bound codes such as HPL. However, when using faster mobile cores such as Cortex-A15, a 1GbE interconnect starts becoming a bottleneck. Current ARM-based mobile chips include peripherals targeted to the mobile market and thus, do not provide enough bandwidth or are not compatible with faster network technologies used in supercomputing, such as 10GbE or Infiniband. However, the use of 1GbE is extensive in supercomputing—32% of the systems in the November 2012 TOP500 list use 1GbE interconnects—, and potential communication bottlenecks are in many cases addressable in software [88]. Therefore, although support for a high-performance network technology would be desirable for ARM-based HPC systems, using 1GbE may not be a limitation as long as the communication libraries are optimized appropriately for Ethernet communication and the communication patterns in HPC applications are tuned appropriately keeping the network capabilities in mind.

## 5.7 Conclusions

In this chapter we presented Tibidabo, the world's first ARM-based HPC cluster, for which we set up an HPC-ready software stack to execute HPC applications widely used in scientific research such as SPECFEM3D and GROMACS. Tibidabo was built using commodity off-the-shelf components that are not designed for HPC. Nevertheless, our prototype cluster achieves 120 MFLOPS/W on HPL, competitive with AMD Operton 6128 and Intel Xeon X5660-based systems. We identified a set of inefficiencies of our design given the components target mobile computing. The main inefficiency is that the power taken by the components required to integrate small low-power dual-core processors offsets the high energy efficiency of the cores themselves. We perform a set of simulations to project the energy efficiency of our cluster if we could have used chips featuring higher-performance ARM cores and integrating a larger number of them together.

Based on these projections, a cluster configuration with 16-core Cortex-A15 chips would be competitive with Sandy Bridge-based homogeneous systems and GPU-accelerated heterogeneous systems in the Green500 list.

We also explained the major issues and how they should evolve or be improved for next clusters made from low-power ARM processors. These issues include, apart from the aforementioned integration overhead, the lack

of optimized software, the use of mobile-targeted memories, the lack of double-precision floating-point SIMD units, and the lack of support for high-performance interconnects. Based on our recommendations, an HPC-ready ARM processor design should include a larger number of cores per chip (e.g., 16) and use a core microarchitecture suited for high-performance, like the one in Cortex-A15. It should also include double-precision floating-point SIMD units, support for multiple memory controllers servicing DDR3 or DDR4 memory modules, and probably support for a higher-performance network, such as Infiniband, although Gigabit Ethernet may be sufficient for many HPC applications. On the software side, libraries, compilers, drivers and operating systems need tuning for high performance, and architecture-dependent optimizations for ARM processor chips.

Recent announcements show an increasing interest in server-class low-power systems that may benefit HPC. The new 64-bit ARMv8 ISA improves some features that are important for HPC. First, using 64-bit addresses removes the 4GB memory limitation per application. This allows more memory per node, so one process can compute more data locally, requiring less network communication. Also, ARMv8 increases the size of the general-purpose register file from 16 to 32 registers. This reduces register spilling and provides more room for compiler optimization. It also improves floating-point performance by extending the NEON instructions with fused multiply-add and multiply-substract, and cross-lane vector operations. More importantly, double-precision floating-point is now part of NEON. All together, this provides a theoretical peak double-precision floating-point performance of 4 FLOPS/cycle for a fully-pipelined SIMD unit.

These encouraging industrial roadmaps, together with research initiatives such as the EU-funded Mont-Blanc project [94], may lead ARM-based platforms to accomplish the recommendations given in this paper in a near future.

# 6

# Mobile Developer Kits

In the previous chapter we have presented the performance of the Tibidabo prototype cluster – the world first large scale designed, produced and deployed from developer kits powered by mobile processor cores IP – ARM Cortex-A9 processor. We have shown that due to its architecture it suffers from small computational density per node, which in turn leads to low energy efficiency. Also, we have shown that there is a significant power waste due to best-effort system integration using developer kits instead of a professional integration solutions. In order to increase the computational density of a computing system built with commodity mobile and embedded low-power devices, we can take three approaches: increase the number of cores-per-chip, or/and leverage computational potential of accelerators e.g. GPUs, and use the high-performance mobile processors IP - such as ARM Cortex-A15 which was announced at the time of Tibidabo prototype deployment. In this chapter we show how both off-chip and on-chip GPUs can increase achievable performance of an HPC system node based on a mobile SoC. We present evaluation of two different platforms powered by mobile SoCs – the first integrates a compute capable discrete GPU, and the second one features an on-chip compute capable GPU. Further, we show the improvement of both performance and energy-efficiency of a high-performance mobile core IP, ARM Cortex-A15, as a potential building block for an HPC node based on mobile and embedded technology.

## 6.1 Evaluation Methodology

During our study, we evaluated multiple computing node architectures, some of which include compute accelerators, and in these cases the effort of porting real world production-level applications with thousands of lines of code was unaffordable. Hence, in order to evaluate these mobile platforms we developed and used a number of benchmarks that stress different architectural features and cover a wide range of algorithms employed in HPC applications (for more information please consult Chapter 3 Section 3.2.1). To get a comprehensive overview of mobile platforms, we measure both the performance (execution time) and power consumption while executing the Mont-Blanc benchmark suite 3.2.1.

## 6.2 CARMA Kit: a Mobile SoC and a Discrete GPU

NVIDIA CARMA developer kit [104], is a platform that provides two computing configurations - homogeneous and heterogeneous. The first one is a quad-core ARM Cortex-A9 processor cluster, and the second one couples this cluster with a mobile discrete GPU for computing acceleration. The CARMA board (see Figure 6.1 has the same layout as the Tegra 2 platform used for Tibidabo cluster node (see Figure 5.1b). The main improvement over the Tibidabo node developer board is that it is based on the more powerful NVIDIA Tegra 3 SoC [103] featuring quad-core ARM Cortex-A9 running at 1.3 GHz. Moreover, CARMA kit doubles the available memory with 2 GB of DDR3 memory and features a single 1GbE NIC (interfaced through USB to Tegra 3). It uses four PCIe 1.0 lanes, providing 1GB/s of bandwidth[1], to connect to a discrete mobile GPU.

The featured GPU in this configuration is the NVIDIA Quadro 1000M, which is an entry-level laptop GPU, with 45 W of TDP (Thermal Design Power). This GPU is not particularly well suited for applications that use double-precision floating point since the ratio between single-precision and double-precision floating-point instructions throughput is 1:8, meaning that peak double-precision performance is 8 times lower than the peak single-precision performance[2]. Tegra 3 itself brings some improvements in on-chip

[1]NVIDIA Quadro 1000M GPU supports PCIe 2.0 x16 totaling 8GB/s of bandwidth to a host.

[2]NVIDIA GPUs that are commonly used in HPC systems, such as those based on NVIDIA Fermi architecture, have a ratio of 1:2 between single and double-precision floating point

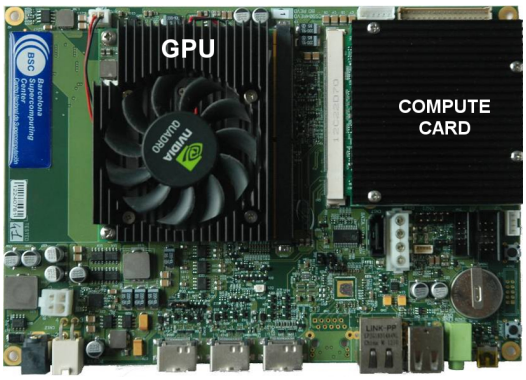



**Figure 6.1:** Physical layout of the NVIDIA CARMA kit with the highlighted computing elements.

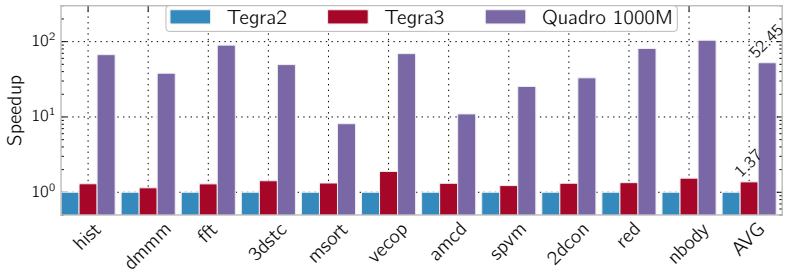
accelerators performance, but like in the Tegra 2, they are not programmable with CUDA, OpenCL or similar programming models. There is also a $5^{th}$ companion CPU core which in a typical mobile scenario runs latency insensitive workloads (like background tasks), but this core cannot be used as a computational resource in an HPC scenario since it is not exposed to the OS.

### 6.2.1 Evaluation Results

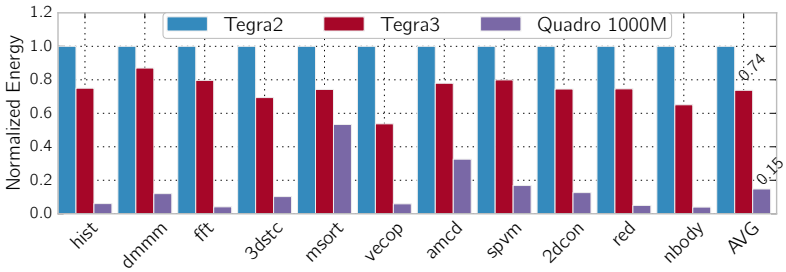
In this section we present performance (execution time) and energy figures for the CARMA platform running Mont-Blanc benchmarks. We present the results for two possible computing scenarios – with and without discrete GPU accelerator, and compare the platform to NVIDIA Tegra 2 platform used as the building block of the Tibidabo cluster. In the case of computing without GPU accelerator, we evaluate both single core and multicore scenarios.

Although the core microarchitecture is the same in both Tegra 2 and Tegra 3, the higher operating frequency of the latter is reflected in the resulting performance: when evaluation the single core computing scenario Tegra 3 provides 1.37× speedup compared to the Tegra 2 (see Figure 6.2a). Since Tegra 3 offers slightly higher memory bandwidth (DDR3 vs DDR2 in Tegra 2), even better performance is observed for those benchmarks that are memory intensive and having simple access pattern, such as *vecop*, where the performance on Tegra 3 improves beyond the clock speed difference. Regarding the energy savings (see Figure 6.2b) Tegra 3 platform consumes 26% less energy on average compared to the Tegra 2 platform.

Figures 6.3a and 6.3b show the performance and energy-to-solution results for all benchmarks using all the available CPU cores: two on Tegra 2 and four on Tegra 3. On average, Tegra 3 completes execution two times

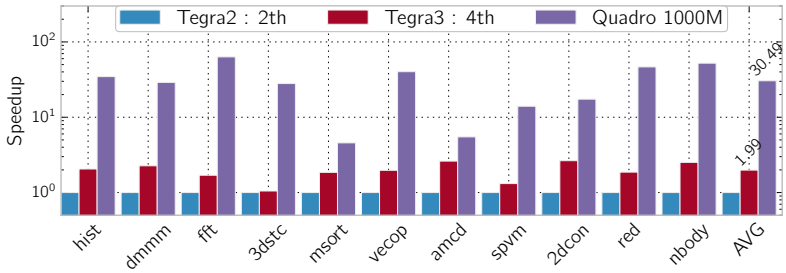



(a) performance

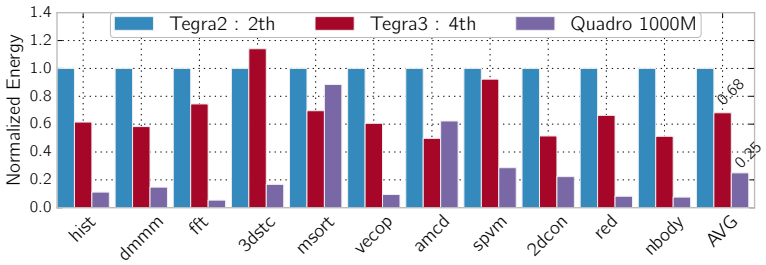



(b) energy

**Figure 6.2:** Evaluation of NVIDIA CARMA Kit: single core a) performance and b) energy results. All data is normalized to NVIDIA Tegra 2 platform.




(a) performance




(b) energy

**Figure 6.3:** Evaluation of NVIDIA CARMA Kit: multi-threaded a) performance and b) energy results. All data is normalized to NVIDIA Tegra 2 platform.

faster. Although it uses twice the number of cores, Tegra 3 requires 67% of the energy-to-solution on average. The larger number of cores in the Tegra 3 MPSoC roughly translates into a doubling of the computational power, but a very small increment in the system power consumption. The power consumed by the CPU cores accounts only for a fraction of the total platform power budget, hence the power consumption of the whole board does not increase linearly with the number of cores. This result shows that energy efficiency benefits from increasing the multicore density as long as the workloads scale reasonably well. Only *3D stencil* hass a worse energy-to-solution when running on Tegra 3 compared to Tegra 2. This benchmark is very memory intensive because only one stencil point is used thus the ratio of floating point operations to memory operations is the lowest among all the benchmarks. As a result, this code requires a very large number of accesses that consume more power on Tegra 3 because of the higher frequency of the memory clock.

Both Figure 6.2 and Figure 6.3 also show the performance of the CUDA version of the benchmarks running only on the discrete GPU. These performance and energy-to-solution results use all the processing cores in the GPU, as we do not have a way to restrict the execution to a subset of the cores. The GPU performance is, on average, 30x better than dual-core Tegra 2 and 15x better than quad-core Tegra 3. Energy-to-solution provided with the discrete GPU also show benefits, saving on average 65% of energy with respect to Tegra 2 and 49% with respect to Tegra 3. However, there are two benchmarks which do not show energy savings – *merge-sort* and *atomic monte carlo dynamics* (see Figure 6.3b). In these cases running the benchmark on the GPU takes more energy than using the Tegra 3 cores only, but still less than Tegra 2. This shows that off-loading tasks to the discrete GPU pays-off only if the achieved speedup is large enough to compensate for the increased platform power consumption when running on the GPU compared to using just the CPU.

Evaluation results are in line with our initial assumptions: increased multicore density of a mobile SoC shows improvement in performance (2% in our case), and reduces energy-to-solution to 68% of the base case. Also, we have successfully demonstrated that coupling a discrete GPU to a mobile SoC is feasible, and also brings a significant performance improvement and energy savings on parallel workloads – up to 30× for the former and up to 65% for the latter.

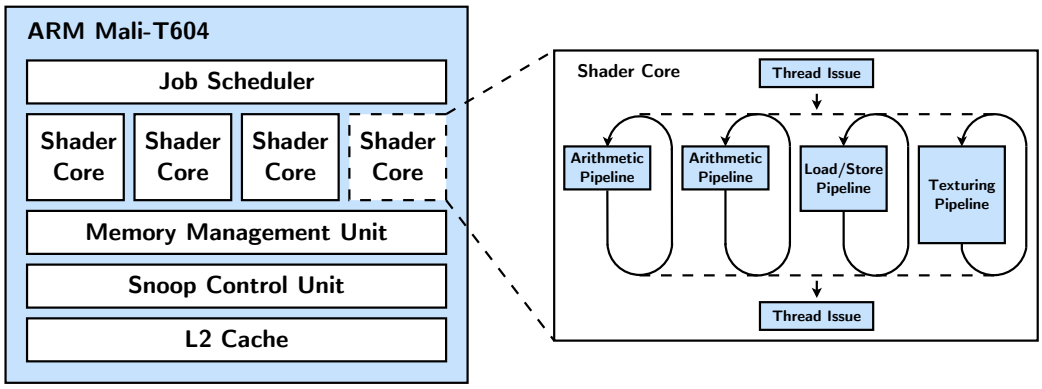



**Figure 6.4:** Architecture of the ARM Mali-T604 GPU.

## 6.3 Arndale Kit: Improved CPU Core IP and On-Chip GPU

Arndale developer kit [21], released in 2012, comprises the Samsung Exynos 5250 embedded system-on-chip (SoC) and is equipped with 2 GB of DDR3L-1600 memory. The Samsung Exynos 5250 integrates a dual-core ARM Cortex-A15, running at 1.7 GHz with 32 KB of private L1 instruction and data cache, and 1 MB of shared L2 cache. The Cortex-A15 is improved over the Cortex-A9 in terms of microarchitecture including, but not limited to, higher number of outstanding cache misses, longer out-of-order pipeline, and improved branch predictor [124][3]. Alongside the CPU cores, the SoC features a four-core ARM Mali-T604 GPU – OpenCL programmable on-chip GPU. Until recently, the main focus of embedded GPUs was 2D and 3D high-quality graphics, and the ARM Mali-T604 GPU is the first of the kind to support general-purpose computing with OpenCL. We depict the architecture of ARM Mali-T604 in the following section.

### 6.3.1 ARM Mali-T604 GPU IP

Figure 6.4 depicts the architectural details of the ARM Mali-T604 GPU. The GPU supports up to four shader cores, each having two arithmetic, one load/store, and texturing pipeline. Each arithmetic pipe is capable of 17 ALU (Arithmetic Logic Unit) operations, which in turn leads to 72.48 GFLOPS

[3] All these microarchitectural improvements allow for increased operating frequency of the CPU.

in single precision at 533 MHz[4][5]. The *Job Manager*, implemented in hardware, abstracts the GPU core configuration from the driver and distributes the computational tasks to maximize the GPU resource utilization. Unlike desktop and server GPUs, ARM Mali-T604 shares the main memory with the CPU thus avoiding explicit memory transfers which in turn saves the energy on the data movements – this is facilitated with the help of *Memory Management Unit* which maps memory from the CPU's address space into the GPU's address space. The L2 cache is shared between the shader cores and maintained coherent by the *Snoop Control Unit*.

### 6.3.2 Evaluation Results

In this section we introduce and discuss a performance and energy evaluation of the on-SoC ARM Mali-T604 GPU. Figure 6.5 depicts the performance and energy figures when utilizing different computing elements available on the Samsung Exynos 5250 SoC – CPU cores and the GPU. Evaluation consists of single-core (serial), dual-core (OpenMP) and GPU (OpenCL) executions. We exercise double-precision data sets, and normalize all results to the single-core executions on the ARM Cortex-A15 CPU..

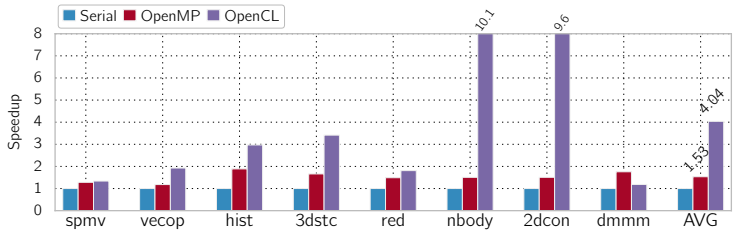
Regarding the executions on the GPU, with OpenCL, four out of eight benchmarks (*spmv*, *vecop*, *red*, and *dmmm*) show a performance improvement below 2× over the single-core executions. Benchmarks *hist* and *3dstc* experience a speedup of 3× and 3.4×. Furthermore, *2dcon* and *nbody* benchmarks show speedup of 9.6× and 10× respectively. Finally, compared to single-core execution, multi-core and GPU executions achieve the following speedups on average across the entire benchmarks set: multi-core 1.53× and GPU 4.04×.

Energy-to-solution results are depicted in Figure 6.5b. Multi-core executions are either even in terms of energy expenditure compared to single-core execution (*spmv* and *vecop*), or offer up to 30% savings – the case of *hist* benchmark. Further, all benchmarks executed on the GPU experience significant energy savings compared to the single-core executions – ranging from 27% (*dmmm*) to 89% (*nbody*)OpenCL set show energy savings compared to Serial versions - ranging from 5% (*spvm*) to 89% (*nbody*). On average, across the entire benchmarks set, with the use of the GPU, we achieve 56% energy savings.

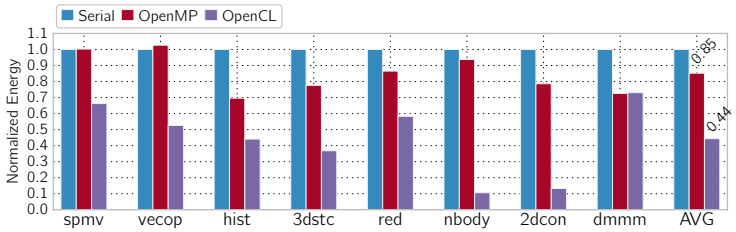
The results we have shown clearly indicate that a mobile capable GPU, like ARM Mali-T604, could offer performance and energy savings benefits

[4]FLOPS are calculated as follows: 7 from dot products, 1 from scalar addition, 4 from vec4 addition, 4 from vec4 multiply, and 1 from scalar multiply.

[5]However, given the missing details regarding ALU internals, it is not clear how to derive peak double-precision floating-point performance.




(a) performance




(b) energy

**Figure 6.5:** Evaluation of the ARM Mali-T604 GPU: a) performance and b) energy results.

compared to mobile CPU cores. The effort of application porting and tuning is in line with the required effort for their server counterparts.

## 6.4 Putting It All Together

Results from previous sections, and Tibidabo prototype cluster (see Chapter 5 look encouraging. That is why we decided to opt for building a next-generation HPC cluster powered by mobile SoCs. In this section, we present the results of comparing Tibidabo node, CARMA and Arndale kit, as potential building blocks for our next-generation cluster, against a contemporary x86 microarchitecture based processor.

### 6.4.1 Comparison Against a Contemporary x86 Architecture

In this section we examine the performance, energy efficiency, and memory bandwidth of single platforms powered by mobile SoCs. We chose developer boards with three different SoCs: NVIDIA Tegra 2 and Tegra 3, and Samsung Exynos 5250[6]. These SoCs cover two successive ARM processor microarchitectures: NVIDIA Tegra 2 and Tegra 3 rely on ARM Cortex-A9

[6]Commercial name of this SoC is Samsung Exynos 5 Dual

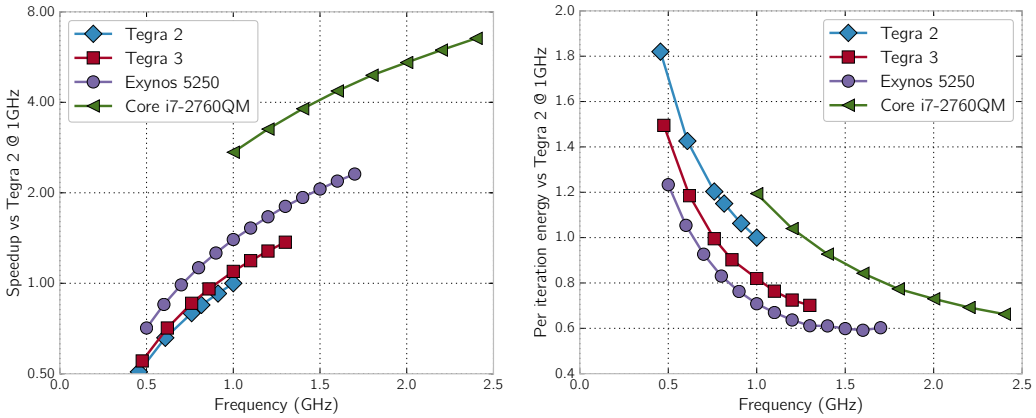
**Table 6.1:** Platforms under evaluation

| **SoC name** | NVIDIA Tegra 2 | NVIDIA Tegra 3 | Samsung Exynos 5250 | Intel Core i7-2760QM |
|---|---|---|---|---|
| **CPU** | | | | |
| *Architecture* | Cortex-A9 | Cortex-A9 | Cortex-A15 | SandyBridge |
| *Max. frequency (GHz)* | 1.0 | 1.3 | 1.7 | 2.4 |
| *Number of cores* | 2 | 4 | 2 | 4 |
| *Number of threads* | 2 | 4 | 2 | 8 |
| *FP-64 GFLOPS* | 2.0 | 5.2 | 6.8 | 76.8 |
| **GPU** | Integrated (graphics only) | Integrated (graphics only) | Integrated Mali-T604 (OpenCL) | Intel HD Graphics 3000 |
| **Cache** | | | | |
| *L1 (I/D)* | 32K/32K private | 32K/32K private | 32K/32K private | 32K/32K private |
| *L2* | 1M shared | 1M shared | 1M shared | 256K private |
| *L3* | - | - | - | 6M shared |
| **Memory controller** | | | | |
| *Number of channels* | 1 | 1 | 2 | 2 |
| *Width (bits)* | 32 | 32 | 32 | 64 |
| *Max. frequency (MHz)* | 333 | 750 | 800 | 800 |
| *Peak bandwidth (GB/s)* | 2.6 | 5.86 | 12.8 | 25.6 |
| **Developer kit** | | | | |
| *Name* | SECO Q7 module + carrier | SECO CARMA | Arndale 5 | Dell Latitude E6420 |
| *DRAM size and type* | 1 GB DDR2-667 | 2 GB DDR3L-1600 | 2 GB DDR3L-1600 | 8 GB DDR3-1133 |
| *Ethernet interfaces* | 1 Gb, 100 Mb | 1 Gb | 100 Mb | 1 Gb |

cores, capable of one Fused Multiply-Add (FMAC) operation every two cycles, and the Samsung Exynos 5250 integrates ARM Cortex-A15, with a single-cycle fully-pipelined FMAC unit. It is not only the CPU that makes a difference between the aforementioned platforms, the memory subsystems are also improved with each new generation. More insights about important characteristics of the different SoCs and the corresponding hardware platforms we use in our evaluation are shown in Table 6.1.

In addition, we include one laptop platform, which contains the same Intel Sandy Bridge microarchitecture used in current state-of-the-art Intel Xeon server processors.[7] We chose the laptop as a platform for comparison since the laptop integrates a set of features similar to those of mobile developer kits. In order to achieve a fair comparison in energy efficiency between the developer boards and the Intel Core i7 we boot the laptop directly into the Linux Command Line Interface (CLI), and we further disable the screen in order to reduce the non-essential power consumption. We give a quantitative measure of the difference in performance between mobile SoCs and high-performance x86 cores, which is driven by the different design points.

[7]We used a Sandy Bridge Intel Core i7. A server-class Intel Xeon also integrates Intel QPI (Quick Path Interconnect) links and PCIe Gen3, but these are not relevant for single node performance comparisons.




**(a)** performance **(b)** energy

**Figure 6.6:** Mobile platforms comparative evaluation: single core evaluation of a) performance and b) energy. All data is normalized to Tegra 2 @ 1GHz.

### Methodology

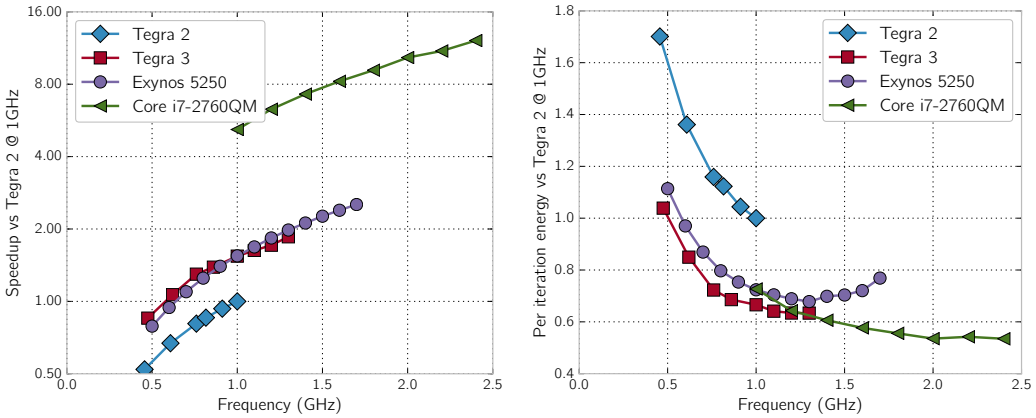
In our experiments, the problem size for the benchmarks is the same for all the platforms, thus each platform has the same amount of work to perform in one iteration. We set the number of iterations such that the total execution time is approximately for all platforms, and the benchmark runs for long enough to get accurate energy consumption figures. We evaluate our platforms with Mont-Blanc benchmarks for assessing the compute capability of each platform, and we use STREAM benchmark for measuring effective per-platform memory bandwidth.

Both power and performance are measured only for the parallel region of the workload, excluding the initialization and finalization phases.[8]

### Evaluation Results

Each frequency sweep data point, shown in Figure 6.6 and 6.7, represents the average across all the benchmarks normalized to the baseline Tegra 2 platform running at its maximum frequency of 1GHz. Frequency points are chosen from each platform's available operating frequency points provided by the *cpufreq* Linux utility. For performance results we show the speedup with respect to the baseline, and for energy efficiency we normalize the results

[8]It was not possible to give a fair comparison of the benchmarks including initialization and finalization,since the developer kits use NFS or $\mu$SD card storage whereas the laptop uses its hard drive.




**(a)** performance **(b)** energy

**Figure 6.7:** Mobile platforms comparative evaluation: multicore evaluation of a) performance and b) energy. All data is normalized to Tegra 2 @ 1GHz.

against the baseline.

Figure 6.6 shows the single-core CPU performance and energy efficiency for each SoC as we vary the CPU frequency. We demonstrate that the performance improves linearly, on average, as the frequency is increased.

Tegra 3 brings a 9% improvement in execution time over Tegra 2 when they both run at the same frequency of 1GHz. Although the ARM Cortex-A9 core is the same in both cases, Tegra 3 has an improved memory controller which brings a performance increase in memory-intensive benchmarks[9]. The Arndale platform at 1 GHz shows a 30% improvement in performance over Tegra 2, and 22% over Tegra 3, due to the improved ARM Cortex-A15 microarchitecture. Compared with the Intel Core i7 CPU, the Arndale platform is only two times slower.

Averaged across all the benchmarks, the Tegra 2 platform at 1GHz consumes 23.93 J to complete the work in one iteration. At the same frequency, Tegra 3 consumes 19.62J, giving an improvement of 19%, and Arndale consumes 16.95J, a 30% improvement. The Intel platform, meanwhile, consumes 28.57J, which is higher than all the ARM-based platforms.

When we run the CPUs at their highest operating frequencies, instead of all at the frequency of 1GHz, the Tegra 3 platform becomes 1.36 times faster than the Tegra 2 platform, and it requires 1.4 times less energy. The Exynos 5 SoC from the Arndale platform brings additional improvements in performance: it is 2.3 times faster than Tegra 2 and 1.7 times faster than

[9]Here we refer to *vecop* and *3dstc* benchmarks from the Mont-Blanc benchmarks.

Tegra 3. The Intel core at its maximum frequency is only 3 times faster than the Arndale platform.

We can see that the generation change of the ARM cores has closed the gap in performance with respect to their Intel counterparts. Our experiments show that from the situation when Tegra 2 was 6.5 times slower we have arrived to the position where Exynos 5 is now only 3 times slower than the Sandy Bridge core.

Figure 6.7 depicts the results for the multi-core executions with OpenMP version of the benchmarks. The benchmarks always use all the cores that are available in the platform (two in Tegra 2, four in Tegra 3, two in Arndale and four in Sandy Bridge). In all the cases, multicore execution has brought improvements, both in performance and in energy efficiency, with respect to the serial version of the benchmarks. In case of Tegra 2 and Tegra 3 platforms, the OpenMP version uses 1.7 times less energy per iteration. Arndale exhibits better improvement (2.25 times), while the Intel platform reduces energy to solution 2.5 times.

In the single-core evaluation Intel platform, as a whole, was the least energy efficient platform, but in multicore scenario it became the most energy efficient one due to an increased amount of power spent on computing elements (cores). Further, Tegra 3 with its four cores is more energy efficient than Arndale platform, but Arndale is able to scale its operating frequency beyond that of Tegra 3 and thus to offer higher-performance in turn.

Our energy efficiency results show that for all the platforms the SoC is not the main power sink in the system. When we increase the frequency of the CPU, its power consumption increases (at least) linearly with the frequency, but we see that the overall energy efficiency improves. This leads to the conclusion that the majority of the power is used by other components, rather than CPU cores.

**Memory Bandwidth** With Figure 6.8 we depict the achievable memory bandwidth of each platform, measured using the STREAM benchmark.

Our results show a significant improvement in memory bandwidth, approximately 4.5 times, between the Tegra platforms (ARM Cortex-A9) and the Samsung Exynos 5250 (ARM Cortex-A15). This appears to be mostly due to the better Cortex-A15 microarchitecture which also improves the number of outstanding memory requests [124], and due to an additional channel in memory controller. Compared with the peak memory bandwidth, the multicore results imply an efficiency of 62% (Tegra 2), 27% (Tegra 3), 52% (Exynos 5250), and 57% (Intel Core i7-2760QM). These results show that increase in computing capability of mobile platforms is also followed with a

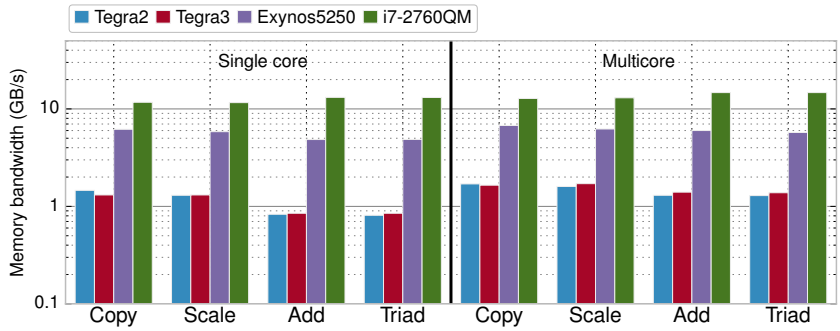



**Figure 6.8:** Mobile platforms comparative evaluation: memory bandwidth

proper increase in sustainable memory bandwidth. Finally, the ability of the aforementioned mobile platforms to utilize the memory bandwidth is on pair with the contemporary x86 architecture-based CPU.

## 6.5 Conclusions

In this chapter we have shown that there is a continuous improvement in the performance that a mobile SoC could offer to High-Performance Computing.. Apart from improved core microarchitecture with the transition from ARM Cortex-A9 to Cortex-A15, mobile SoCs' GPUs became compute capable and programmable by means of standard programming models for accelerators like OpenCL and CUDA. These advances, as we have demonstrated, are followed with an improvement in memory technology leading to increase of memory bandwidth needed to sustain the compute performance of a mobile SoC.

Evaluation results presented in this chapter, served as a guide for selecting a chip for the next-generation mobile SoCs powered HPC prototype. We selected Samsung Exynos 5250 since it offers an improved core microarchitecture improving both performance and energy efficiency, and integrates an on-SoC GPU which could bring a significant performance boost for particular HPC kernels. In the next chapter, we present the architecture and evaluate the Mont-Blanc prototype – an HPC system built with the aforementioned Samsung Exynos 5250 SoC, commodity networking and storage, using standard HPC system integration.

# 7 The Mont-Blanc Prototype

This thesis is tightly coupled with the Mont-Blanc project, which aims at providing an alternative HPC system solution based on the current commodity technology: mobile chips. As a demonstrator of such an approach, the project designed, built and set-up a 1080-node HPC cluster made of Samsung Exynos 5250 SoCs. The Mont-Blanc project established the following goals: to design and deploy a sufficiently large HPC prototype system based on the current mobile commodity technology; to port and optimize software stack and enable its use for HPC; to port and optimize a set of HPC applications to be run at this HPC system.

The contributions of this chapter are:

- A detailed description of the Mont-Blanc prototype architecture
- A thorough performance and power evaluation of the prototype, comparing it to a Tier-0 production system in Europe, the MareNostrum III supercomputer.
- A set of recommendations for the next-generation HPC system built around the Mont-Blanc approach.

## 7.1 Architecture

In this section we present the architecture of the Mont-Blanc prototype. We highlight peculiarities of each building block as we introduce them.

### 7.1.1 Compute Node

The Mont-Blanc compute node is a *Server-on-Module* architecture. Figure 7.1 depicts the Mont-Blanc node card (Samsung Daughter Board or SDB) and its components. Each SDB is built around a Samsung Exynos 5250 SoC integrating two ARM Cortex-A15 CPUs @ 1.7 GHz sharing 1 MB of on-die L2 cache, and a mobile 4-core ARM Mali-T604 GPU @ 533MHz. The SoC connects to the on-board 4 GB of LPDDR3-1600 RAM through two 32-bit memory channels shared among the CPUs and GPU, providing a peak memory bandwidth of 12.8 GB/s.

The node interconnect is provided by the ASIX AX88179 USB 3.0 to 1Gb Ethernet bridge, and an Ethernet PHY (Physical layer). An external 16 GB $\mu$SD card provides the boot-loader, OS system image, and local scratch storage.

The node connects to the blade through a proprietary bus using a PCI-e 4x form factor edge connector (EMB connector).

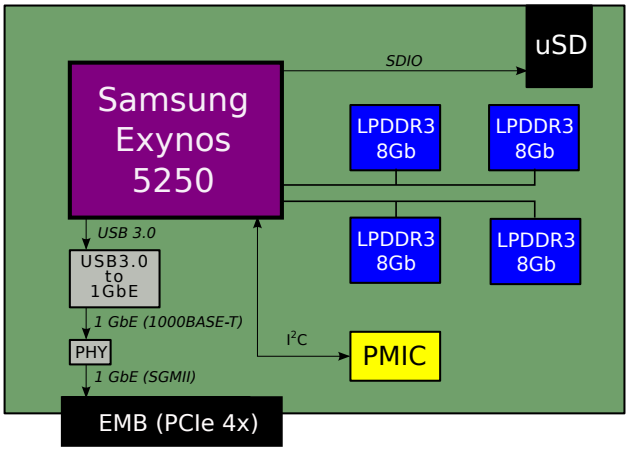



**Figure 7.1:** The Mont-Blanc prototype: compute node block scheme (not to scale).

### 7.1.2 The Mont-Blanc Blade

Figure 7.2 describes the architecture of the Mont-Blanc blade, named Ethernet Mother Board (EMB), depicted in Figure 7.3). The blade hosts 15 Mont-Blanc nodes which are interconnected through an on-board 1GbE switch fabric. The switch provides two 10GbE up-links. In addition, the EMB provides management services, power consumption monitoring of SDBs, and blade

level temperature monitoring. The EMB enclosure is air-cooled through the fans installed on the front side.

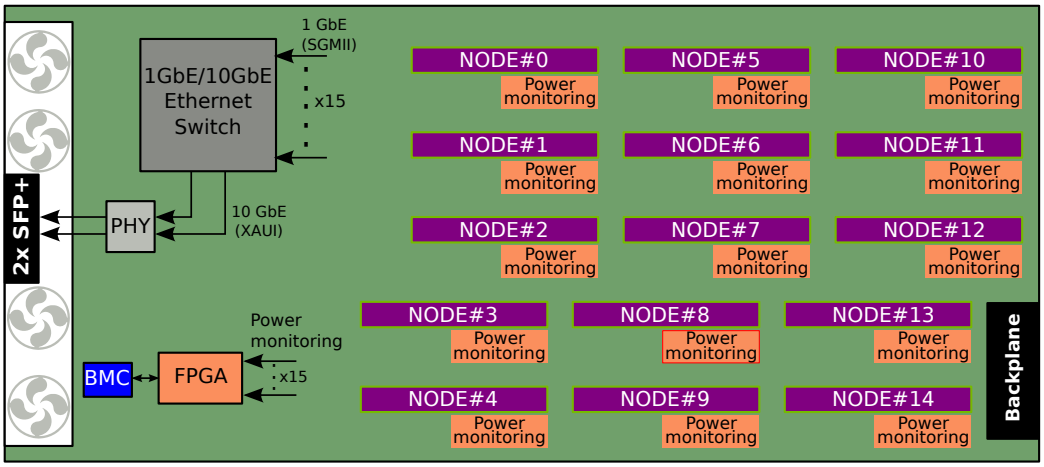



**Figure 7.2:** The Mont-Blanc prototype: compute blade block scheme.

### 7.1.3 The Mont-Blanc System

The entire Mont-Blanc prototype system (shown in Figure 7.4) fits into two standard 42U-19″ racks. Each Mont-Blanc rack hosts up to four 7U Bullx chassis which in turn integrate nine Mont-Blanc blades each. In addition, racks are populated with two 2U 10GbE Cisco Nexus 5596UP Top-Of-the-Rack (TOR) switches, one 1U prototype management 1GbE switch[1], and two 2U storage nodes.

#### System interconnect

The Mont-Blanc prototype implements two separate networks: the 1GbE management network, and the 10GbE MPI network. The management network is out of the scope of this paper, thus we depict the implementation of the MPI interconnect only in Figure 7.5.

The first level of switching is provided inside the blades using a 1GbE switch fabric providing two 10GbE up-links. Switching between the blades occurs at the TOR switches with a switching capacity of 1.92 Tbps per switch. The racks are directly connected with four 40GbE links.

[1]Not visible, mounted on the back.


**Figure 7.3:** The Mont-Blanc prototype: compute blade physical view.

#### Storage

The Lustre parallel filesystem is built on a Supermicro Storage Bridge Bay based on x86-64 architecture, with a total capacity of 9.6 TB providing 2-3.5 GB/s read/write bandwidth (depending on the disk zone). The storage system is connected to the top-of-the-rack switches with four 10GbE links.

#### Cooling

Compute nodes are passively cooled using a top-mounted heatsink, while blades provide active air-cooling through variable speed front-mounted fans in a temperature control loop.

### 7.1.4 The Mont-Blanc Software Stack

The work done during the research phase of this thesis helped maturing the HPC software stack on the ARM architecture. Today, working with the Mont-Blanc prototype feels like working with any other HPC cluster.

The Mont-Blanc prototype nodes run Ubuntu 14.04.1 Linux on top of the customized Linaro Kernel version 3.11.0 which enables user space driver for OpenCL programming of the ARM Mali-T604 GPU. The rest of the software stack components are shown in Figure 7.6.

A very relevant part of the Mont-Blanc software stack is the OmpSs

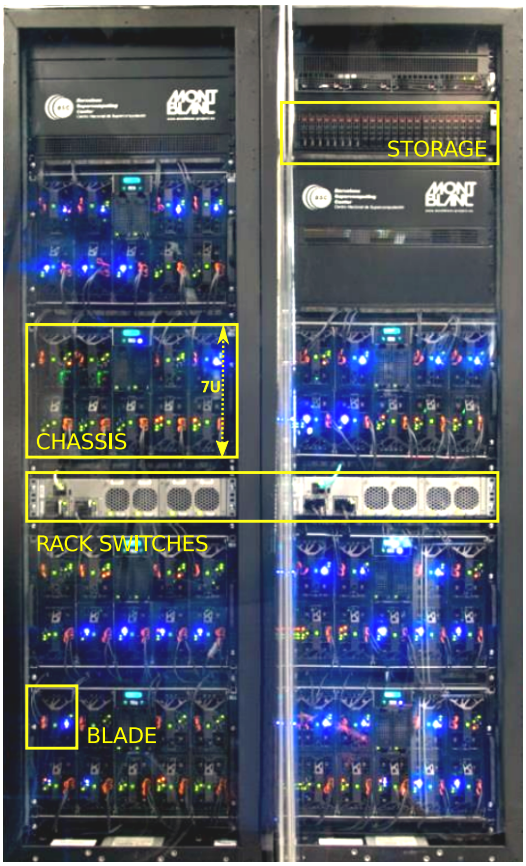



**Figure 7.4:** The Mont-Blanc prototype: physical view of the entire system.

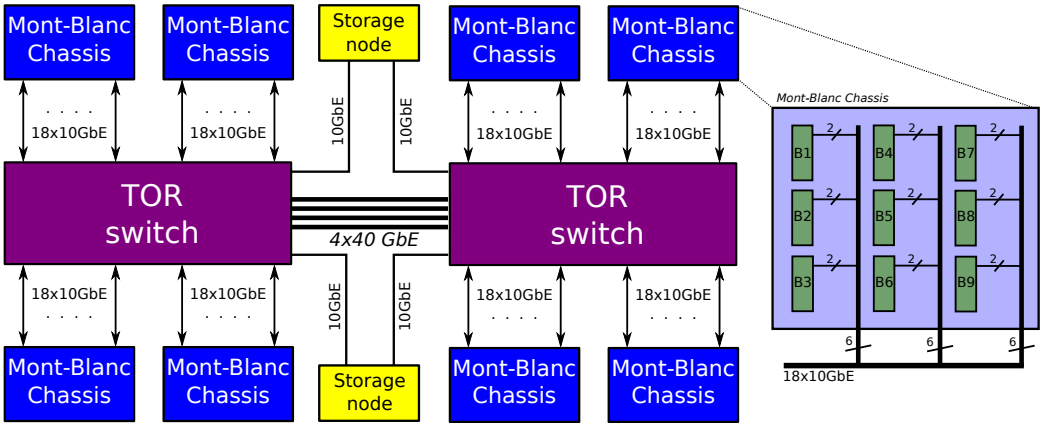



**Figure 7.5:** The Mont-Blanc prototype: system interconnect topology.

| **Compilers** | | | | |
|---|---|---|---|---|
| GNU | JDK | Mercurium | | |
| **Scientific libraries** | | | | |
| ATLAS | LAPACK | SCALAPACK | FFTW | |
| BOOST | clBLAS | clFFT | PETSc | HDF5 |
| **Performance analysis** | | | **Debugger** | |
| Extrae | Paraver | Scalasca | Allinea DDT | |
| **Runtime libraries** | | | | |
| Nanos++ | OpenCL | OpenMPI | MPICH3 | |
| **Cluster management** | | | | |
| SLURM | Nagios | Ganglia | | |
| **Hardware support** | | | **Storage** | |
| Power monitor | | | LustreFS | |
| **Operating System** | | | | |
| Ubuntu | | | | |

**Figure 7.6:** The Mont-Blanc prototype: system software stack

programming model [48], a forerunner of OpenMP support for tasks, provided by the Mercurium compiler and the Nanos++ runtime complex. OmpSs is a task-based programming model with explicit inter-task dataflow that allows the runtime system to orchestrate out-of-order execution of the tasks, selectively off-loading of tasks to the GPU when possible, or running them on the CPU if the GPU is busy. Applications ported to OmpSs can make simultaneous use of the CPU and the GPU, dynamically adapting to load imbalance situations during execution [107].

### 7.1.5 Power Monitoring Infrastructure

The Mont-Blanc prototype provides a unique infrastructure to enable high-frequency measurements of power consumption at the granularity of a single compute node, scaling to the whole size of the prototype.

The Mont-Blanc system features a digital current and voltage meter in the power supply rail to each SDB. An FPGA (Field-Programmable Gate Array) on each EMB accesses the power sensors in each SDB via I2C (Inter Integrated Circuit) interface and stores the averaged values every 1,120$ms$ in a FIFO buffer. The Board Management Controller (BMC) on the EMB communicates with the FPGA to collect the power data samples from the FIFO before storing them in its DDR2 memory along with a timestamp of the reading. User access to the data is then provided by the BMC over the management Ethernet through a set of custom Intelligent Platform Management Interface (IPMI) commands.

To provide application developers with power traces of their applications, the power measurement and acquisition process is conveniently encapsulated and automated in a custom-made system monitoring tool. The tool is de-

**Table 7.1:** The Mont-Blanc prototype: compute performance summary.

| | | |
|---|---|---|
| ***Compute Node*** | | |
| | **CPU** | **GPU** |
| Compute element | 2×ARM Cortex-A15 | 1×ARM Mali-T604 |
| Frequency | 1.7 GHz | 533 MHz |
| Peak performance (SP) | 27.2 GFLOPS | 72.5 GFLOPS |
| Peak performance (DP) | 6.8 GFLOPS | 21.3 GFLOPS |
| *Memory (shared)* | 4 GB LPDDR3-800 | |
| ***Blade = 15×Node*** | | |
| Peak performance (SP) | 408 GFLOPS | 1.08 TFLOPS |
| Peak performance (DP) | 102 GFLOPS | 319.5 GFLOPS |
| *Memory* | 60 GB | |
| ***Chassis = 9×Blade*** | | |
| Peak performance (SP) | 3.67 TFLOPS | 9.79 TFLOPS |
| Peak performance (DP) | 0.92 TFLOPS | 2.88 TFLOPS |
| *Memory* | 540 GB | |
| ***System = 8×Chassis*** | | |
| Peak performance (SP) | 29.38 TFLOPS | 78.3 TFLOPS |
| *Total (SP)* | *107.7 TFLOPS* | |
| Peak performance (DP) | 7.34 TFLOPS | 23 TFLOPS |
| *Total (DP)* | *30.3 TFLOPS* | |
| *Memory* | 4.32 TB | |

veloped with a focus on simplicity and scalability by respectively employing MQTT [95], for lightweight transport messaging, and Apache Cassandra, a scalable, distributed database for storing the acquired power data along with other time-series based monitoring data.

### 7.1.6 Performance Summary

Table 7.1 shows the performance figures of the Mont-Blanc prototype. The two Cortex-A15 cores provide a theoretical peak performance of 27.2 GFLOPS in single-precision (SP) and 6.8 GFLOPS in double-precision (DP). The performance discrepancy comes from the fact that the SIMD unit, denoted as NEON, supports only SP floating-point (FP) operations, therefore DP FP instructions execute in a scalar unit.

The on-chip quad-core Mali-T604 GPU provides 72.5 GFLOPS SP and 21.3 GFLOPS DP [7]. The overall node performance is 99.7 GFLOPS SP and 28.1 GFLOPS DP.

Table 7.1 shows the peak performance at the blade, chassis and entire system levels for CPU and GPU separately. The whole system has a peak performance of 107.7 TFLOPS SP and 30.3 TFLOPS DP.

Due to the 32-bit nature of the SoC architecture, each node integrates only 4 GB of memory. The high node integration density of 1080 nodes (2160 cores) in 56U (over 19 nodes per U) adds up to 4.32 TB of memory, and an

**Table 7.2:** Peak performance comparison of Mont-Blanc and MareNostrum III nodes.

| | *Mont-Blanc* | | *MareNostrum III* | |
|---|---|---|---|---|
| **Frequency** [GHz] | 1.7 | | 2.6 | |
| # **sockets** | 1 | | 2 | |
| | CPU | GPU | CPU | GPU |
| **Peak FP-64** [GFLOPS] | 6.8 | 21.3 | 332.8 | *-n/a-* |
| **Memory BW** [GB/s] | 12.8 | | 51.2 | |
| **Network BW** [Gb/s] | 1 | | 40 | |
| **Intersocket BW** [GB/s] | *-n/a-* | | 32 | |

| **Tag** | **Full name** |
|---|---|
| 2dc | 2D convolution |
| amcd | Markov Chain Monte Carlo method |
| dmm | Dense matrix-matrix multiplication |
| hist | Histogram calculation |
| ms | Generic merge sort |
| nbody | N-body calculation |
| 3ds | 3D volume stencil computation |
| fft | One-dimensional Fast Fourier Transform |
| red | Reduction operation |
| vecop | Vector operation |

**Table 7.3:** List of Mont-Blanc benchmarks

aggregate 13.8 TB/s memory bandwidth.

## 7.2 Compute Node Evaluation

In this section, we present a comparison between the Samsung Exynos 5250 SoC[2] used in the Mont-Blanc prototype, and its contemporary 8-core Intel Xeon E5-2670 [3] server processor running at 2.6 GHz and used in the MareNostrum III supercomputer [26]. The MareNostrum node is a dual-socket implementation, using DDR3-1600 memory DIMMs. For a side-by-side peak performance comparison of Mont-Blanc and MareNostrum nodes please consult Table 7.2.

*Methodology:* We present and discuss both core to core, and node to node performance and energy figures when executing the Mont-Blanc benchmark suite [109] (see Table 7.3). We report performance (execution time) and energy differences by normalizing to that of MareNostrum. We obtain node power using the power monitoring infrastructure of the Mont-Blanc prototype (see Section 7.1.5) , and the node energy consumption in MareNostrum provided through LSF job manager.

[2]Introduced in Q3 2012
[3]Introduced in Q1 2012

### 7.2.1 Core Evaluation

In Figure 7.7, we present the performance comparison on a core-to-core basis between the Mont-Blanc prototype and MareNostrum supercomputer. This comparison, using single-threaded benchmarks, gives a sense of the performance difference between both cores without the interference of scheduling and synchronization effects of parallel applications.

Across the benchmark suite, Mont-Blanc is from 2.2 to 12.7 times slower. The Cortex-A15 core underperforms the Intel SandyBridge mainly due to: the lack of SIMD DP FP extensions (vectorization observed in *dmm*, *3ds*, *fft*, *red*, *vecop*); lower per socket memory bandwidth (12.8 vs 51.2 GB/s); and limited memory subsystem resources geared towards low power[4] (more off-chip accesses observed in *2dc*, *amcd*, *hist*, *nbody*). On average, across the entire suite, a Mont-Blanc core is 4.3x slower than MareNostrum.

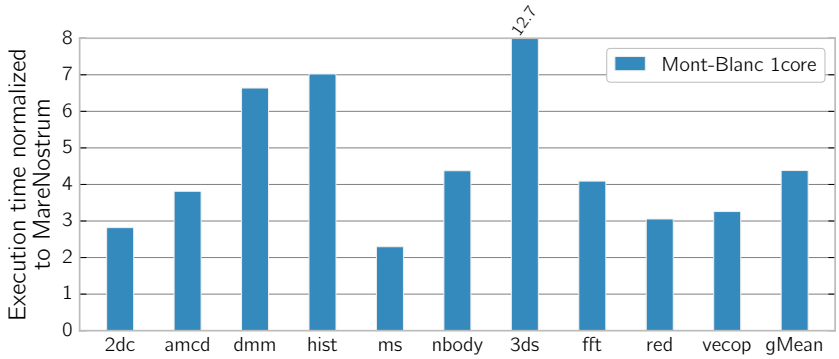



**Figure 7.7:** Mont-Blanc vs MareNostrum III: core to core performance comparison with Mont-Blanc benchmarks.

### 7.2.2 Node Evaluation

In Figure 7.8, we compare performance and energy consumption on a node-to-node basis between the Mont-Blanc prototype and the MareNostrum III supercomputer.

Given the characteristics of the Mont-Blanc SoC and its software stack, we evaluate three different computing scenarios: homogeneous CPU computing with OpenMP (blue bars), heterogeneous CPU + GPU with OpenCL (red bars), and heterogeneous with OmpSs (violet bars).

[4]Intel Xeon E5-2670 features 20 MB of third level cache.

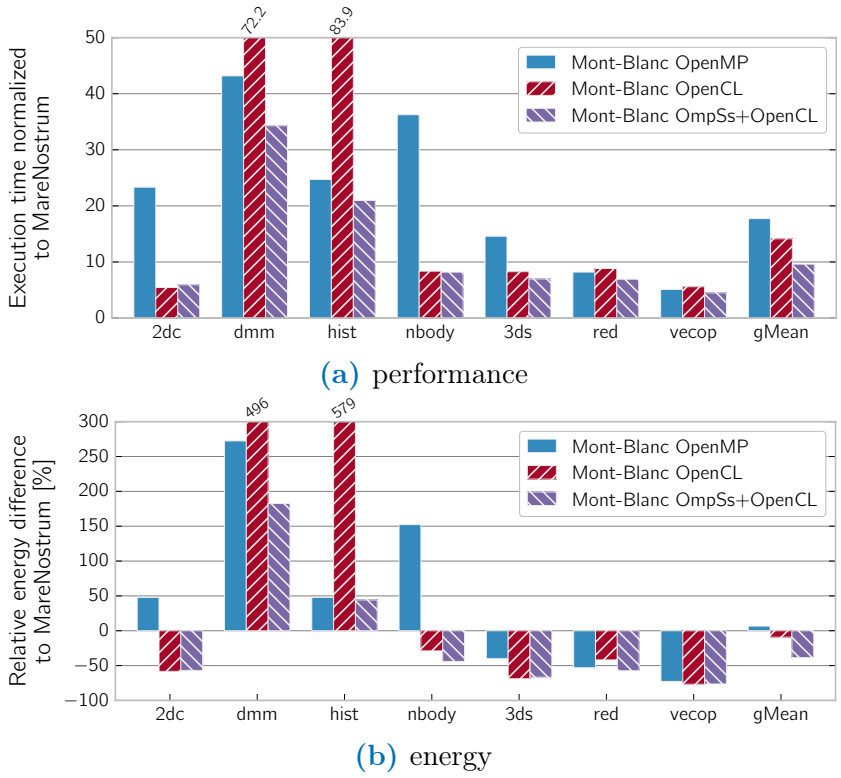



**Figure 7.8:** Mont-Blanc vs MareNostrum III: node to node a) performance and b) energy comparison with Mont-Blanc benchmarks. Computational resources of the Mont-Blanc are used as follows: OpenMP - two CPU cores, OpenCL - GPU, OmpSS + OpenCL - one CPU core + GPU.

Comparing CPU-only computing, a dual-core Mont-Blanc node is 18x slower than a 16-core MareNostrum node. When using OpenCL to off-load all compute tasks to the GPU, Mont-Blanc is 14x slower than MareNostrum. Finally, using OmpSs to exercise both the GPU and the CPU, we significantly reduce the gap to only 9x across the benchmark suite.

Energy wise, when using only CPUs with OpenMP, a Mont-Blanc node consumes 5% more energy compared to a MareNostrum node. As we close the performance gap, Mont-Blanc nodes become more energy efficient on average: from consuming 20% less energy when using only GPU, to consuming 45% less energy when using both GPU and CPU cores.

Our results show that, when using the embedded GPU, Mont-Blanc can be significantly more energy-efficient than an homogeneous cluster like MareNostrum III. However, Mont-Blanc needs applications to scale to 10-15x more nodes in order to match performance, and interconnection network performance is critical in that case.

### 7.2.3 Node Power Profiling

Energy has two dimensions: power and time. Execution time depends on how the application performs on the underlying architecture. Power depends on how much the application stresses compute resources, processor physical implementation and SoC power management. The power monitoring infrastructure in the Mont-Blanc prototype (Section 7.1.5) helps the user reason about both factors. Comparing the power of different mappings[5] (CPU, GPU, or CPU+GPU), the user can estimate the speedup required to compensate the power differences and run the system at the best energy efficiency point.

Figure 7.9 shows a high sampling rate power profile of one Mont-Blanc node for different mappings of the execution of the 3D-stencil benchmark. The different mappings include one CPU core (sequential), dual core (OpenMP), GPU (OpenCL), and GPU + 1 CPU (OmpSs).

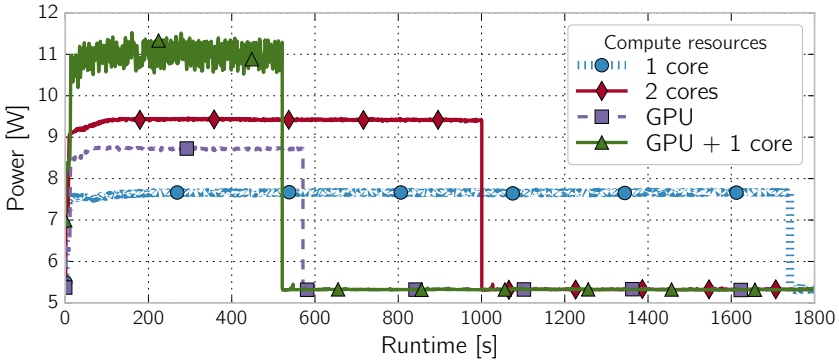



**Figure 7.9:** The Mont-Blanc prototype: power profile demonstration of different compute to hardware mappings for 3D-stencil computation. Note: markers are only to distinguish lines, not sampling points.

The node idle power is 5.3W. This includes the static power of all the components given that frequency scaling is disabled for benchmarking purposes. The average power consumption when running on one and two CPU cores is 7.8W and 9.5W respectively. This includes the power consumption of the SoC, memory subsystem and network interface.

Node power when using the GPU and the GPU + 1 CPU is 8.8W and 11W, respectively. When running on the GPU alone, one of the cores is still active as a helper thread that synchronously launches kernels to the GPU and therefore blocks until they complete. When running OmpSs on the GPU

[5]Counting only elements contributing to the computing.

+ 1 CPU, one of the cores is the GPU helper and the other one runs a worker thread and contributes to computation, thus adding that extra power.

Our results show that the extra power required by OmpSs because of adding one core to GPU computation outweighs the performance improvement, leading to 15% higher energy to solution in the 3D stencil benchmark (as we show in Figure 7.8).

From our results with other benchmarks, node power varies across different workloads although it remains in the same range seen in Figure 7.9. The maximum power seen for executions with two CPU cores is 14W, and 13.7W for executions with the GPU plus one CPU core.

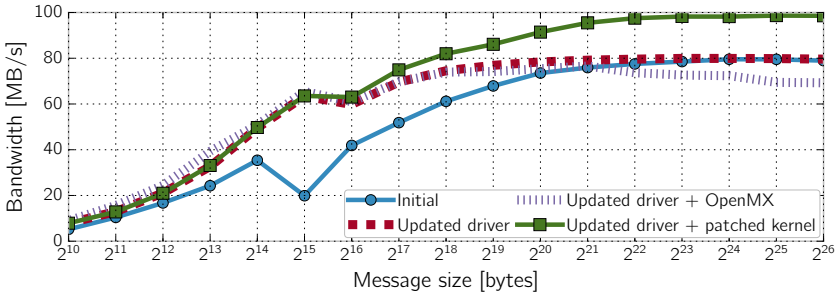
The above shows the relevance of the power measurement infrastructure in the Mont-Blanc prototype. It allows us to explain where and how the power is being spent, even at high frequencies. The ability to visualize power over time is even more valuable for applications showing different phases that may benefit of different CPU-GPU mappings. This way, the user can identify the best mapping for each application phase.

In systems without a power profile (which just provide the total job energy consumption), such analysis requires a less accurate and time-consuming trial-and-error approach looking at power deltas over multiple runs of different configurations.

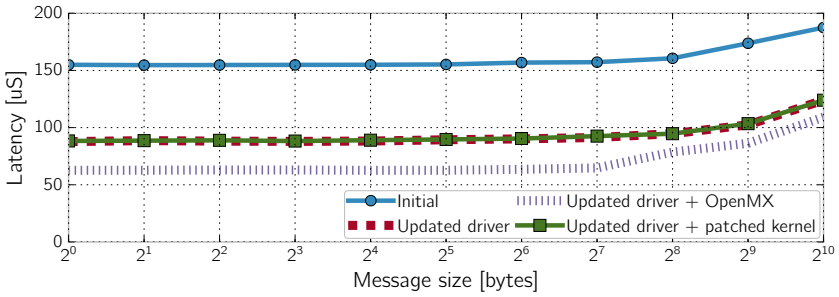
## 7.3 Interconnection Network Tuning and Evaluation

In this section, we quantify the latency and bandwidth of the Mont-Blanc interconnection network. Since the Mont-Blanc interconnect is implemented using a lossy Ethernet technology, it is of paramount importance that every layer is properly tuned. Thus, we discuss the improvements in different parts of the interconnect stack which in turn affect the overall interconnect performance.

In Figure 7.10, we present both bandwidth and latency measurements obtained from the Mont-Blanc prototype using the Intel MPI PingPong benchmark. We present four curves per graph, each corresponding to incremental improvements on the node network interface.




(a) Bandwitdh




(b) Latency

**Figure 7.10:** The Mont-Blanc prototype: inter-node bandwidth and latency tuning. The figure illustrates different optimizations of the network subsystem and how they reflect on the resulting achievable bandwidth and latency. All data was gathered using MPI Ping-Pong benchmark.

After the initial deployment of the Mont-Blanc prototype we measured achievable MPI throughput and latency of 80 MB/s and 156 $\mu$s respectively (blue line). The results were obtained using the NIC driver built-into the Linux Kernel.

We updated the driver using a proprietary version provided by the USB-to-GbE bridge maker[6], achieving significant improvements in both throughput and latency for small messages: up to 3.4x better throughput for messages under 64KB, and only 88$\mu$s latency for zero-sized messages (red line). However, bandwidth for larger messages stayed the same as in the initial configuration. The new driver provides a configurable wait interval between the consecutive bulk transfers on the USB bus, which we reduced to the bare minimum.

Additionally, we did a back-port of a Linux Kernel patch [93] which improved throughput in USBNET driver for USB 3.0 compliant devices. This patch improved throughput for messages larger than 64 KB, achieving a

[6]ASIX chip AX88179

maximum throughput of 100 MB/s (green line). For reference purposes, the same benchmark run on a server class x86_86 system (with integrated 1GbE NIC) achieves 112.5 MB/s and a 46.5$\mu$s latency [80], so we achieve 89% of the potential bandwidth, but our latency is still 1.9x higher.

Most of the ping-pong latency is due to the TCP/IP protocol stack, which runs on the ARM Cortex-A15 CPU. We also deployed the Open-MX [61] protocol stack (a free implementation of the Myricom protocol) to replace TCP/IP. The lighter-weight protocol reduced latency for small messages to 65 $\mu$s, that also increased bandwidth for messages under 32 KB. However, it degraded the throughput for the larger message sizes (violet line).

Since most MPI application will exchange large messages, we prefer to optimize bandwidth over latency and select the proprietary driver + patched USBNET kernel for the stable network configuration of the prototype.

## 7.4 Overall System Evaluation

In this section, we evaluate the Mont-Blanc cluster using full-scale, production MPI applications, as listed in Table 7.4, plus three reference mini-apps used by US DOE National Labs [42]. All test applications use OpenMPI, and run on the CPU only.

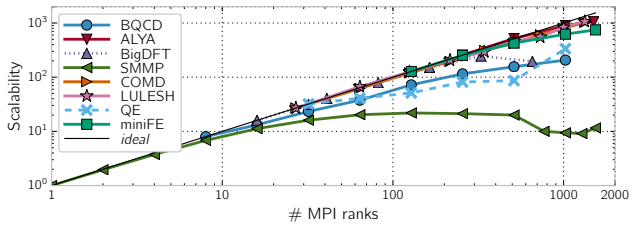
**Table 7.4:** The Mont-Blanc prototype: MPI applications used for scalability evaluation.

| Application | Domain |
|---|---|
| BigDFT [30], [57] | Electronic Structure |
| BQCD [97] | Quantum Chromodynamics |
| MP2C [120] | Multi-Particle Collision Dynamics |
| QuantumESPRESSO [59] | Electronic Structure and Materials Modeling |
| SMMP [49, 50, 91] | Molecular Thermodynamics |
| Alya [128, 127] | Biomedical Mechanics |
| CoMD [52] | Proxy for Molecular Dynamics |
| LULESH [82, 81] | Proxy for Hydrodynamics |
| miniFE [69] | Proxy for Finite Element Method |

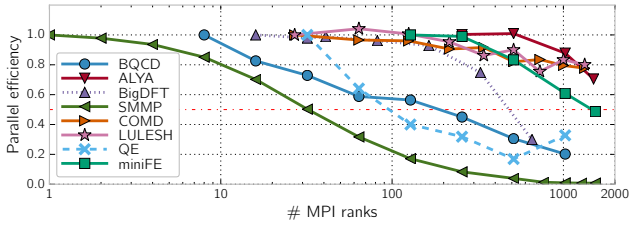
### 7.4.1 Applications Scalability

In Section 7.2.2, we show that a Mont-Blanc node is 18x slower than a MareNostrum III node when using only the CPU cores. This means we should linearly scale a workload to 18x more compute nodes to achieve equivalent performance.

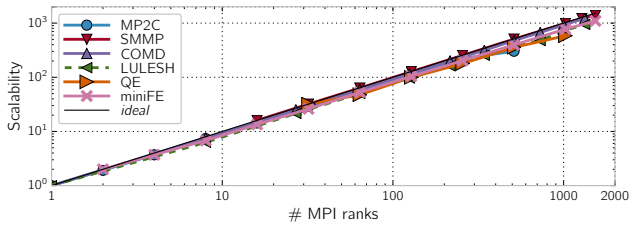
In Figure 7.11, we show both *strong* and *weak* scaling figures for MPI applications on the Mont-Blanc prototype. Each graph is accompanied with the corresponding parallel efficiency graph in order to provide more details about the application's scalability. Note that 16 Mont-Blanc nodes already span 2 EMB blades, and 32 nodes span 3 blades. Also, most applications had their baseline run with more than one node due to the 4GB/node DRAM limitation.



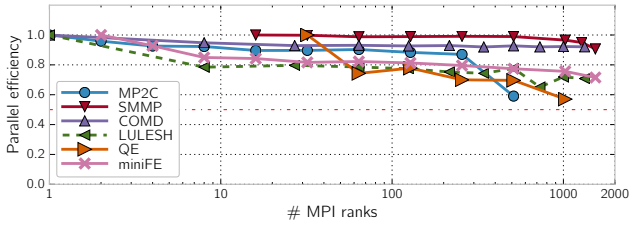

**(a)** Strong scaling.




**(b)** Strong efficiency.




**(c)** Weak scaling.




**(d)** Weak efficiency.

**Figure 7.11:** The Mont-Blanc prototype: scalability and parallel efficiency of MPI applications.

Strong scaling for miniFE quickly degrades starting at 32 nodes. Parallel

efficiency drops to 50%, and performance flattens, and even degrades at 512 nodes. BQCD and QE also exhibit quick strong scaling degradation, but still run at more than 50% efficiency on 64 nodes. The rest of the applications scale linearly to hundreds of nodes, with 4 of them still running at more than 50% efficiency at the full scale of the system.

Our results show that it is reasonable to scale applications to 16 nodes in order to compensate for the difference to a MareNostrum III node. However, not all applications will scale further to compensate for multiple MareNostrum III nodes.

Weak scaling results are much better. Most of the applications still run at more than 70% efficiency at the maximum problem size. Notably, CoMD and SMMP run at more than 90% efficiency, but QE and MP2C degrade to 60% efficiency.

Detailed performance analysis reveals the causes for lack of scalability: besides the low bandwidth / high latency 1GbE network, the system suffers from lost packets in the interconnect, each incurring at least one Retransmission Time-Out (RTO), and load imbalance introduced by scheduler preemptions.

### Lost Packets

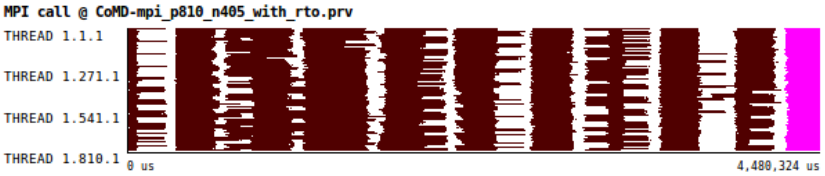
In Figure 7.12, we show an execution profile for the real CoMD run, and a Dimemas [22] simulated run eliminating network retransmissions. Both traces have the same time scale.

The simulated profile shows that the native execution suffered from many lost packets (most communications suffer from at least 1 retransmission), thus reducing performance by 1.47x. This is of course application dependent, and depends on the communication patterns, message sizes, volume of communication, etc.

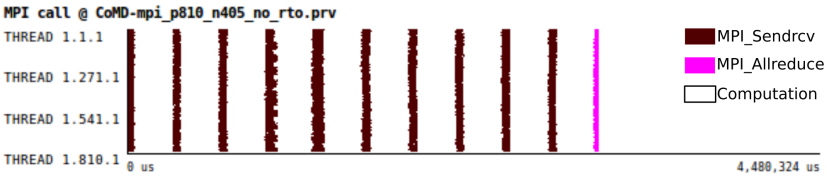
Further analysis of the duration of MPI send operations shows that CoMD experiences multiple retransmissions per packet, with and average MPI send duration of 158ms (compared to 50ms optimum), and often reaching 400ms.

Figure 7.13a shows the performance degradation as a function of how many nodes experience a retransmission penalty on every message they send. The results show that the penalty is linear with respect to the retransmission delay. But more important, the results show that as soon as one node has to retransmit, the whole applications pays almost the full penalty.

Figure 7.13b shows the performance degradation as a function of how many messages need to be retransmitted (by any node). The results show that the penalty is linear with the retransmission delay and the retransmission probability. Both results combined indicate that it is important to avoid




**(a)** Packet loss in place.




**(b)** No packet loss.

**Figure 7.12:** The Mont-Blanc prototype: illustration of the TCP/IP packet loss effect on MPI parallel applications: a) trace with, and b) without packet loss. Trace without packet loss is replayed with the Dimemas simulator in order to remove TCP/IP retransmissions and corresponding timeouts. The X axis represents time, the Y axis represents the process number.

retransmissions in the whole system, or to cluster retransmissions in time, because as soon as one node has to retransmit, it does not matter if others also have to retransmit. For example, a glitch in a switch that causes all nodes connected to it to retransmit would have a similar penalty to a glitch in the NIC of one of the nodes, forcing it alone to retransmit.

To minimize the penalty of retransmissions, we reduce the $RTO_{min}$ parameter in the TCP/IP stack from the default 200ms to 5ms (the lowest possible in our system). While the lowering of $RTO_{min}$ parameter reduces retransmission penalties, it would be desirable implementing Retransmission Early Detection (RED) to reduce the effects of retransmissions. However, packet loss does not exclusively happen at switch buffers, but we also observed nodes can drop packets. In addition, our blade switches which forward most of the network traffic do not support Explicit Congestion Notification (ECN) markings, thus not being able to control transmission rates.

### Pre-emptions

Figure 7.14 shows a histogram of the duration of computational phases in the real CoMD execution. The gradient color shows the total time spent in computation phase of a given duration (green/light is low, blue/dark is high).

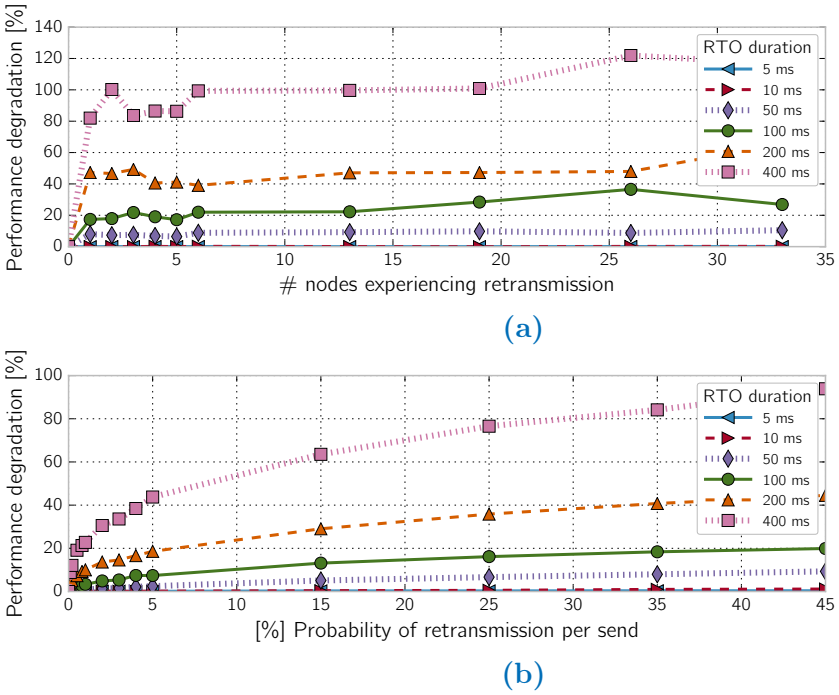



**Figure 7.13:** Performance degradation due to retransmissions: a) every message is affected for selected nodes; b) random messages are affected.

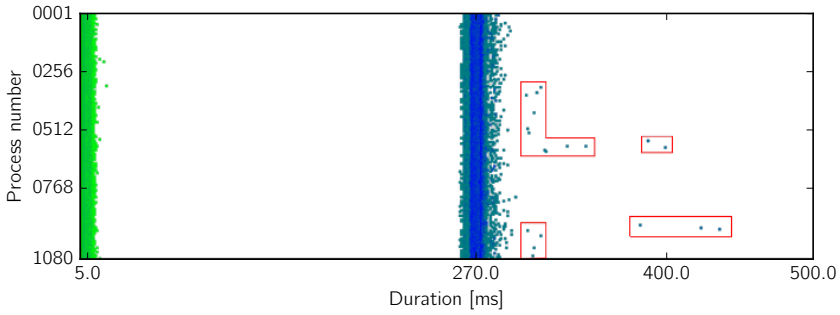



**Figure 7.14:** The Mont-Blanc prototype: illustration of computational noise effect. Figure shows the 2D Histogram of computational phases duration of representative CoMD application execution. X axis represents bins of durations, Y axis represents process number. Gradient coloring: green-blue. Coloring function: logarithmic.

The figure shows two main regions of $5ms$ and $270ms$ durations. We match the $5ms$ regions to the TCP/IP retransmissions (matching the $5ms$ RTO setting, and confirming that many processes suffer retransmissions). Then, the remaining time is spent in $270ms$ regions, matching the duration

of one inner iteration of the application. Beyond the $270ms$ boundary, we identify a set of outliers taking significantly more time (marked with red polygons).

Checking the IPC (Instructions per Cycle) of these computation phases, we confirm that the divergence in execution time is not related to load imbalance in the application. There are external factors introducing this variation. We attribute them to scheduler preemptions, and from now on treat them as OS noise in the discussions to come.

Further simulations of different noise injection frequencies and noise duration indicate that the performance impact of OS noise is linear with the probability of noise being injected, and the ratio for the noise duration to the computational burst. That is, applications with short computational bursts are more prone to suffer OS noise performance degradation than applications with long computational bursts.

### 7.4.2 Comparison With Traditional HPC

Figure 7.15 shows a comparison between Mont-Blanc and MareNostrun III when using the same number of MPI ranks (same number of cores). Since applications are not completely malleable in the number of MPI ranks they can use, the reported number of cores is different for each application, ranging from 257 to 1536.

Our results show that Mont-Blanc is 3.5× slower on average (matching the Mont-Blanc benchmarks evaluation in Section 7.2.1), and requires 9% less energy to run the applications. However, none of the applications is optimized to use the GPU or OmpSs. On the basis of the results in Section 7.2, we would expect Mont-Blanc to result in better energy efficiency once the GPU is used alongside the CPU.

Table 7.5 shows a comparison of the Mont-Blanc prototype and MareNostrum III when aiming to equalize their execution times. For this experiment we exercise the strong-scaling capability of applications on the Mont-Blanc prototype, so we keep the input set constant and increase the number of MPI ranks to get the same execution time as on MareNostrum III with 64 MPI ranks (4 nodes).

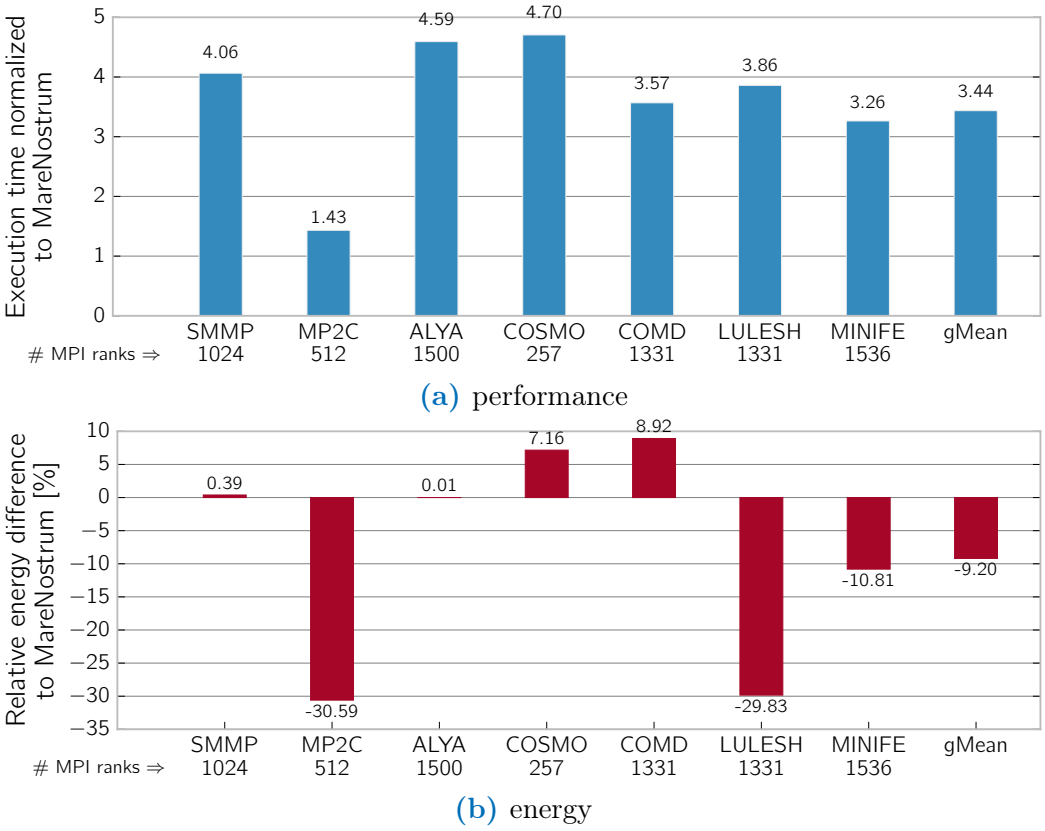



**Figure 7.15:** Mont-Blanc vs MareNostrum III comparison with MPI applications for the same number of MPI ranks: a) performance (execution time) and b) energy comparison. Each application feature its own number of MPI ranks. In both cases lower is better for Mont-Blanc.

**Table 7.5:** Mont-Blanc vs MareNostrum III comparison with MPI applications targeting same execution time, using the same input set.

| | *CoMD* | | *miniFE* | |
|---|---|---|---|---|
| | MN[a] | MB[b] | MN | MB |
| **# MPI ranks** | 64 | 240 | 64 | 224 |
| **Execution time [s]** | 70.72 | 68.05 | 71.66 | 72.19 |
| **Avg. power [W]** | 992 | 1083 | 1065 | 1034 |
| **Energy [Wh]** | 195 | 205 | 212 | 207 |
| **# rack units** | 8 | 7 | 8 | 7 |

[a] MareNostrum III
[b] Mont-Blanc

Our results show that Mont-Blanc needs 3.5–3.75 more MPI ranks to match the MareNostrum III execution time. This is consistent with the 3.5× slowdown observed for constant number of MPI ranks, and shows similar scalability on both systems. In terms of energy consumption, both systems consume approximately the same amount of energy. Regarding rack space, Mont-Blanc requires 7 rack units (1 BullX chassis, 9 blades x 15 nodes, 270 cores), while MareNostrum III requires 8 rack units (4 2U nodes).

We conclude that, when using only the CPUs, Mont-Blanc and MareNostrum III are equally energy efficient when targeting equal execution time (performance) on both systems.

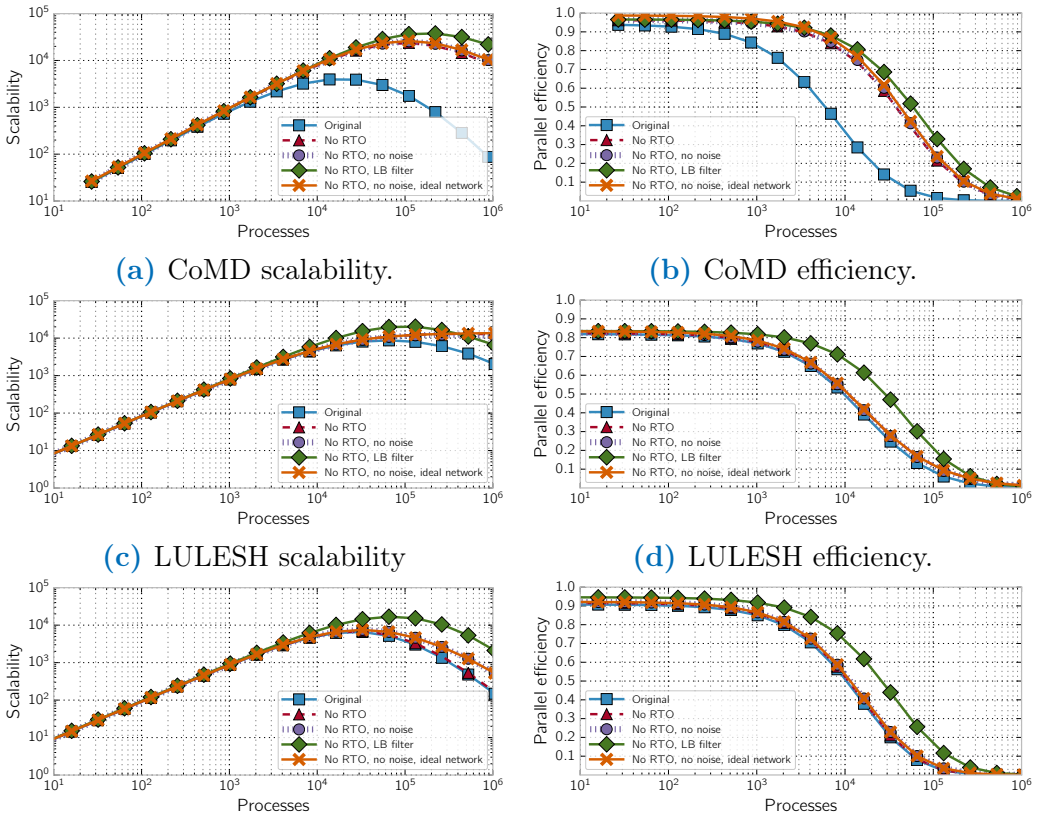
## 7.5 Scalability Projection

The results in Section 7.4 show that the scalability of the Mont-Blanc prototype is affected by the choice of interconnect technology (Ethernet via USB), and potential load imbalance (introduced by the system, or intrinsic to the application). Further, in Section 7.4.2, we revealed a need for good parallel scalability to compensate for lower per node performance compared against the MareNostrum III supercomputer. These issues conceal the potential of the Mont-Blanc approach at scale.

To unveil the scalability of the prototype architecture to larger systems, we employ a state-of-the-art modeling methodology [36, 111] that allows us to project scalability of the current deployment once certain issues have been fixed.

To validate our hypothesis about factors preventing applications to scale, we simulate weak scaling scenarios where we have removed network retransmissions, OS preemptions, and improved load balance in the application.

We remove retransmissions by having the Dimemas simulator assume that messages are always delivered. We remove OS preemptions by recomputing the CPU burst durations using the cycle counter (multiplying by the cycle time). Since the cycle counter is virtualized, it does not count while the application is preempted. To simulate a better load balance, we evenly redistribute the instruction count in a computation phase across all the MPI ranks, and compute the burst duration using the average IPC for the compute phase. Finally, we also simulate an ideal network (lossless, zero latency, infinite bandwidth) to determine if a better (hardware-supported) network would improve the system.




**(a)** CoMD scalability. **(b)** CoMD efficiency.

**(c)** LULESH scalability **(d)** LULESH efficiency.

**(e)** miniFE scalability **(f)** miniFE efficiency.

**Figure 7.16:** The Mont-Blanc prototype: measured and simulated scalability and parallel efficiency. Simulated configurations are as follows: ▲ w/o TCP/IP retransmissions; ● w/o TCP/IP retransmissions and preemptions; ♦ w/o TCP/IP retransmissions + balanced load; ✖ w/o TCP/IP retransmissions, preemptions, and with ideal network parameters.

Figure 7.16 shows the results of estimation of parallel scalability and efficiency for up to 1 million MPI ranks. The baseline setup (blue curve) shows that none of the 3 simulated applications scale beyond 100K MPI ranks with an efficiency over 50%. If we consider that the whole MareNostrum III system has 3,056 nodes (48.9K processors), and that Mont-Blanc is 3.5x slower at the same number of MPI ranks, a Mont-Blanc system with 50K nodes would be 1.75x slower than MareNostrum III, consume the same energy (but less power), and use 12.5% less space.

However, removing only the network packet loss (red line, triangle) already shows a significant improvement for CoMD, now scaling well to 30K processes and still improving performance up to 100K processes. LULESH and miniFE do not seem to be heavily affected by the lossy network.

When we eliminate both the retransmissions, and the OS preemptions (purple line, circle), CoMD does not exhibit a significant improvement. It is clear that its scalability was dominated by the retransmissions. However, LULESH and miniFE show some improvement, that indicates that serialization introduced by OS noise was causing significant damage.

If we go one step further, and look at the ideal network simulations (orange line), we observe that none of the three applications is limited by network performance (after we remove the retransmissions and OS noise). At this point, we have obtained a scalability improvement of 7x for CoMD, 1.2x for LULESH, and 1.1x for miniFE, most of it due to using a lossless network.

Further analysis reveals that load imbalance is the biggest issue affecting scalability of the prototype (green line). Improvement of load balance has a visible impact on all three applications, with LULESH and miniFE being the most affected (notably, the two that were less sensitive to the network performance).

Improving the load balance in the application is beyond the capabilities of the hardware, and while it will affect Mont-Blanc systems and traditional systems like MareNostrum III in a similar way, Mont-Blanc needs to scale to a higher number of processes to compensate for the slower compute nodes, making it the most critical aspect of application development.

## 7.6 Conclusions

In this chapter, we have presented in detail the architecture of the Mont-Blanc prototype, and compared it to a production supercomputer. Our results show that Mont-Blanc is 3.5× slower than MareNostrum III for the same number of MPI processes. However applications can weak scale to 3.5× more nodes to compensate for that, and still run in approximately the

same time and same energy. Since applications must scale to a higher number of nodes, load balancing is a critical design issue for the applications.

Cost savings due to the use of commodity embedded SoCs in Mont-Blanc is impossible to evaluate at this point. The Mont-Blanc prototype is dominated by Non-Recurring Engineering (NRE) costs, while the cost of MareNostrum III is the result of a negotiation and a competitive bid.

However, we do not necessarily advocate for using off-the-shelf mobile processors like the Exynos 5250, just like we do not use off-the-shelf desktop i7 cores for MareNostrum III. We advocate for building workload-specific SoCs based on the IP developed for the embedded and mobile segments, adding the missing features required by HPC, such as a lossless network (as indicated by our results in Section 7.5), ECC memory protection, and the set of accelerators that the workload will exploit. Just like the Intel Xeon used in MareNostrum III builds on the desktop i7 SandyBridge processor.

From the time when Mont-Blanc specifications were fixed, there have been many developments in the embedded computing space: increased multicore counts (4 and 8 cores per SoC), 64-bit ARM processors (Cortex-A72 and Cortex-A57), CUDA capable embedded GPUs (NVIDIA Tegra K1 and X1 SoCs), and on-chip PCIe controllers are all available. Our projections simulating CoMD, LULESH, and miniFE on an upgrade of the Exynos 5250 dual-core Cortex-A15 @ 1.7 GHz to the Tegra X1 quad-core Cortex-A57 @ 1.9 GHz show a 1.6-1.7× performance improvement while still relying only on the CPUs.

Based on our analysis in this chapter, a next-generation Mont-Blanc system should have a lossless interconnection network, and a higher per-node core count to better amortize shared infrastructure costs such as cooling and power supply. Then, the burden falls on the applications, which should fully utilize the SoC resources, such as the embedded GPU[7], and focus on load balancing to scale to a higher number of lower-performance nodes.

Under these conditions, Mont-Blanc type systems would offer equivalent performance to contemporary systems, while potentially saving 45% energy.

[7]The use of OpenCL for HPC code acceleration has not ramped up since 2013 when the ARM Mali GPU was selected for the Mont-Blanc prototype. CUDA and OpenMP with SIMD annotations seem to be the preferred way to use HPC accelerators today.

# 8 Mont-Blanc Next-Generation

The purpose of this application-driven study is to identify bottlenecks of the Mont-Blanc prototype design and to propose a set of design recommendations for next generation systems. In addition, we would like to project the performance and power, if we could have used more recent and future commodity mobile/embedded hardware building blocks. We consider 32-bit ARM SoCs that were not available to the project consortium in time to enter the final prototype design cycle. In addition, we show predictions for systems built with 64-bit ARM based mobile SoCs that appeared at the final phase of the project. At the end, we give an estimate looking into the future, by extrapolating power and performance envelopes for systems built around the recently announced 64-bit ARM Cortex-A72 processor core IP.

## 8.1 Methodology

For this study we rely on execution traces of MPI applications taken on the Mont-Blanc prototype. To predict application performance, while doing design space exploration, we apply a methodology similar to the one used in a previous work [63]. We extend this methodology with an adaptation of the work of IBM [117] that allows for performance prediction of compute phases using a linear combination of benchmarks, instead of doing fully detailed instruction trace simulations.

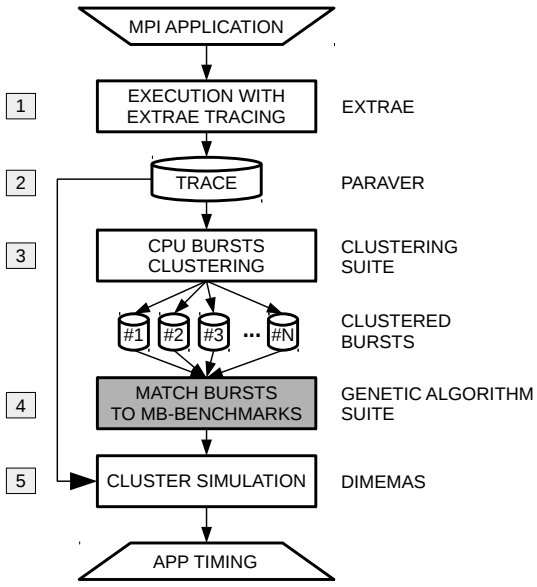



**Figure 8.1:** Illustration of the methodology for performance prediction of potential Mont-Blanc prototype upgrades.

### 8.1.1 Description

Figure 8.1 depicts the performance prediction methodology. The basic idea is to estimate the performance of a real application on a target machine based on the performance of a set of kernels. The premise is that the real, full size, MPI application cannot be executed in the target platform, whereas the kernels do. Moreover, a detailed instruction-level simulation of the real application on a target machine model would be too time consuming. We execute both the kernels and the real applications on a base machine (in our case, the Mont-Blanc prototype), and discover which kernels are representative (and with which weights) of each part of the real application. Then, we run the kernels on the target machine and measure their performance. Using the representative kernel performance on the target machine (and their weights) we project the performance for the real application running on the target machine. As an input we take an MPI application execution trace which holds information about the duration of each computational part (CPU burst), MPI calls and their corresponding communication patterns. In addition, we record the available performance counters data for each CPU burst. Having performance counters data allows for CPU burst classification and grouping by similarity (clustering). We perform clustering on all the CPU bursts, e.g. by grouping bursts with the same performance counters statistics and du-

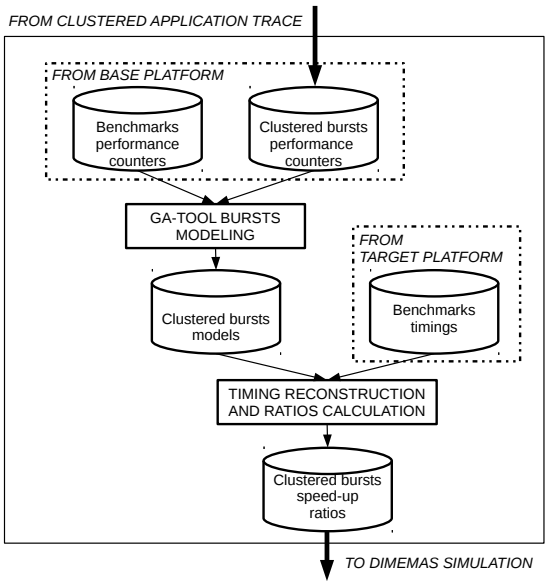



**Figure 8.2:** Computational phases performance modelling scheme.

ration, resulting in a few burst types. Then, we map each burst type into a linear combination of kernels using a tool developed within the project. Once the mapping is done, we can reconstruct the per burst type execution time using the execution time of the kernels. With the new execution time we calculate per burst type speedup ratios with respect to the base machine that are fed into the Dimemas cluster simulator. Dimemas replays the entire MPI application execution using the speedup ratios and network parameters.

### Burst Mapping

In Figure 8.2 we show the procedure of mapping a burst to a set of kernels and how to compute the speedup ratio for Dimemas simulation. Each burst type from an MPI application is modelled as a linear combination of kernels by matching the burst performance counters statistics to the linear combination of performance counters statistics of kernels. We do this on *the base platform*, in this case, the Mont-Blanc prototype. Having this model for each burst type, we reconstruct its execution time on the *target platform* by multiplying the execution times of the corresponding kernels on the *target platform* with their corresponding weights obtained on the base machine. Once we have the execution time of each burst type on both *base* and *target* machines, we compute the corresponding speedup ratio (base exectime/target exectime) and feed it into Dimemas.

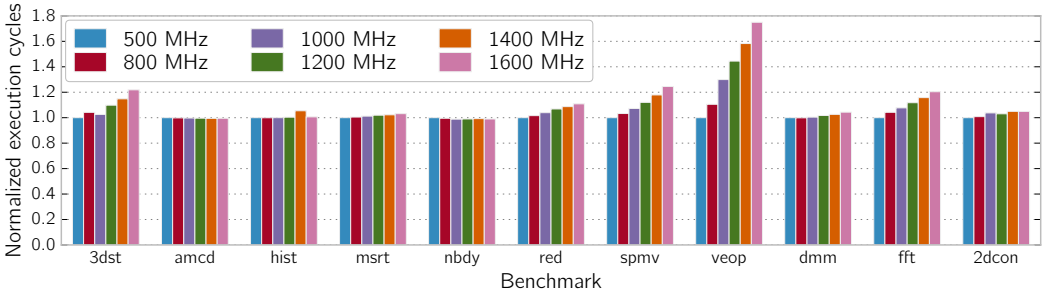



**Figure 8.3:** Mont-Blanc benchmarks: execution cycles vs. operational frequency on the Mont-Blanc node. This figure shows compute/memory bound nature of the workloads. Compute bound benchmarks have all bars of the same height, while the memory bound ones increase the height with frequency. Per benchmark data is normalized to the execution at 500 MHz.

### 8.1.2 Benchmarks

As the benchmark suite, we employ the Mont-Blanc benchmarks, developed for the purpose of our research. The Mont-Blanc benchmarks cover a mix of benchmarks commonly found in HPC applications. Their behaviour covers the compute and memory-bound characteristics of applications. Figure 8.3 depicts the compute/memory bound nature of the Mont-Blanc benchmarks while sweeping the operating frequency of the Mont-Blanc prototype node. In an ideal situation increase in frequency should be followed by the corresponding increase in the performance (reduction of execution time) equal to the ratio of the frequency increase. Five benchmarks show memory-bound behaviour, as it is observed by monotonic increase in the number of execution cycles[1]. Compute-bound kernels require the same number of cycles at different core frequencies, as memory speed (which remains unchanged) is irrelevant for them. This means the speedup is equal to the frequency increase. Vector operation (*vecop*) is the extremely memory-bound case: when frequency is increased by a factor of 3.2, the number of cycles increases by a factor of 1.7, achieving a speedup of just 1.88 compared to the base case of 500 MHz. In addition to the Mont-Blanc benchmarks, we extend our benchmarks set with the STREAM suite (*copy*, *scale*, *add*, *triad*), well-known as memory characterization kernels.

[1]Increase in the number of execution cycles does not necessarily means longer execution time due to the shorter cycle time with an increased operating frequency.

**Table 8.1:** List of target platforms used for performance and power predictions of the potential Mont-Blanc prototype upgrades.

| Platform | Core type | Frequency | Cores per SoC |
|---|---|---|---|
| Mont-Blanc | ARM Cortex-A15 | 1.7 GHz | 2 |
| NVIDIA Jetson | ARM Cortex-A15 | 2.3 GHz | 4 |
| ARM Juno | ARM Cortex-A57 | 1.1 GHz[1] | 4 |
| NG[2]Node | ARM Cortex-A72 | 2.5 GHz | 8 |

[1] We extrapolate operating frequency to 2.3 GHz in our study.
[2] Next Generation

### 8.1.3 Applications

Our performance and power consumption prediction methodology requires collecting the execution and power traces from the Mont-Blanc prototype. Due to stability issues with the prototype we have only a few small application traces that are too small for a large-scale study. At the moment of writing about this study, we managed to get one large trace from the CoMD proxy application [58] with 1080 processes running on 540 nodes (half of the Mont-Blanc prototype), and we use it to conduct our study.

### 8.1.4 Base and Target Architectures

In this study we utilize the Mont-Blanc prototype to build extrapolation models, from both performance and power perspectives. We analyze what could have been the Mont-Blanc prototype and how it could look like in the near future. Here we focus on strictly homogeneous designs - all cores being of the same type and without GPU acceleration. We also investigate the effects of interconnect bandwidth and latencies on a per application basis. Table 8.1 lists the platforms we use in our study, covering both existing and hypothetical systems. Our performance projections assume the same number of cores on the all systems, therefore configurations with more cores per SoC have a proportionally lower number of nodes.

### 8.1.5 Validation

In order to show the error margins of our methodology for modelling single MPI process bursts, we present validation results against three platforms listed in Table 8.2 for HPL and the CoMD proxy application [58]. CoMD has two representative burst types and we list validation results for the both.

**Table 8.2:** List of platforms used for methodology validation.

| Platform | SoC | CPU | Frequency |
|---|---|---|---|
| MB-Node (base) | Exynos 5250 | Cortex-A15 | 1.7 GHz |
| NVIDIA Jetson | Tegra K1 | Cortex-A15 | 2.3 GHz |
| NVIDIA Carma | Tegra 3 | Cortex-A9 | 1.3 GHz |
| Raspberry Pi 2 | BCM2836 | Cortex-A7 | 0.9 GHz |

**Table 8.3:** Methodology validation for HPL and CoMD on different target platforms. All data is normalized to the Mont-Blanc node.

| | **MB-Node** | **Jetson** | **Carma** | **RPi2** |
|---|---|---|---|---|
| *Workload* | *Relative error [%]* | | | |
| **HPL** | 0.00 | 3.12 | 7.42 | 3.55 |
| **CoMD Cluster 1** | 0.00 | 1.31 | 17.47 | 22.61 |
| **CoMD Cluster 2** | 0.00 | 1.48 | 20.81 | 23.07 |

Table 8.3 presents the performance prediction results on the aforementioned platforms. Errors are calculated compared to real execution times. In the case of HPL, the method keeps the prediction error below 10 % across all the platforms, even though we exploit different CPU architectures – starting with an out-of-order CPU (Cortex-A15) and ending with an in-order CPU (Cortex-A7). In the case of CoMD application bursts, prediction error is negligible when modelling execution time on the Jetson platform, whilst for Carma and RPi2 has an acceptable level of 20%. These results are within the margins reported by IBM [117]. We have to underline that all errors produce optimistic predictions - we always predict faster, predicting shorter execution time.

## 8.2 Performance Projections

In this section, we present performance projections for CoMD application running on alternative node platforms and their related configurations listed in Table 8.1. We show the results obtained when varying different system parameters such as core types, number of cores in an SoC, and network parameters - bandwidth and latency.

Table 8.4 shows that, given the total execution time, there are two predominant clusters, namely Cluster 1 and Cluster 2, taking 99% of the total computation time together. To simplify the analysis, we incorporate Cluster

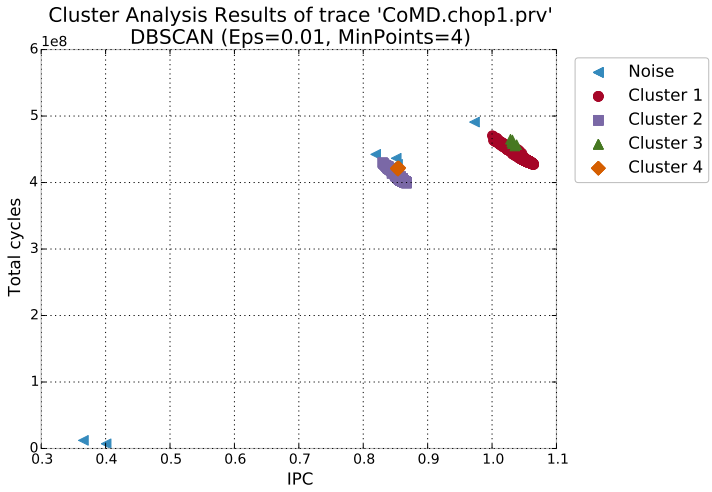



**Figure 8.4:** Example of computational bursts clustering analysis of CoMD application.

**Table 8.4:** Clustering statistics of CoMD computational bursts. Durations are in ms.

| Cluster Name | Density | Tot. Duration | Avg. Duration | % Total Duration |
|---|---|---|---|---|
| Noise | 6 | 1 188 | 198 | 0.0233 |
| Cluster 1 | 10793 | 2 759 066 | 255 | 54.386 |
| Cluster 2 | 9713 | 2 310 236 | 237 | 45.539 |
| Cluster 3 | 6 | 1 628 | 271 | 0.03 |
| Cluster 4 | 4 | 995 | 248 | 0.02 |

3 into Cluster 1, and Cluster 4 into Cluster 2 given their similarities.

Table 8.5 presents the clusters performance models obtained using the methodology described in the Section 8.1. Each cluster is modelled by a set of benchmarks. In this case, both clusters map to Atomic Monte-Carlo Dynamics (amcd) and 2D Convolution (2dcon) benchmarks. Using these benchmarks and corresponding weights, we project the performance for the target platforms in Table 8.1 and show the resulting per cluster speedup ratios in Table 8.6.

ARM Juno is a test platform built with test technology node meant for early CPU IP technology adopters. The Cortex-A57 cores in this platform have the same architecture but operate at a lower frequency than that of a real mobile SoC product implementation. Then, we extrapolated ARM Juno's operating frequency from 1.1 to 2.3 GHz by a linear factor equal to

**Table 8.5:** Performance model of CoMD application: clusters modeling with kernels.

| ClusterID # | Kernels | Weights |
|---|---|---|
| Cluster 1 | amcd 2dcon | 0.156 0.1002 |
| Cluster 2 | amcd 2dcon | 0.138 0.1015 |

**Table 8.6:** Performance model of CoMD application: per-cluster speedup ratios for the target platforms.

| Platform | Core type | Frequency | Per cluster speedup ratios |
|---|---|---|---|
| Mont-Blanc | ARM Cortex-A15 | 1.7 GHz | 1.0, 1.0 |
| NVIDIA Jetson | ARM Cortex-A15 | 2.3 GHz | 1.322, 1.319 |
| ARM Juno | ARM Cortex-A57 | 2.3 GHz | 1.50948, 1.5051 |
| NG Node | ARM Cortex-A72 | 2.5 GHz | 3.5, 3.5 |

the frequency increase, and applied it to the per cluster speedup ratios. This way, we can predict performance for ARM Cortex-A57 CPU implementations on a real mobile SoC implementations such as Samsung Exynos 5433 and Qualcomm Snapdragon 810, from the studies on the ARM Juno board.

For the projections for a NG Node, we use publicly available performance numbers [12] for the highest-performance ARM's CPU core IP – ARM Cortex-A72 core. The public ratio claimed by ARM is a speedup of 3.5 over ARM Cortex-A15 on average. We use this ratio for our study, since there were no existing mobile SoC developers platforms featuring this processor at our disposal.

### 8.2.1 Mont-Blanc Prototype

Figure 8.5 shows the sensitivity to interconnect bandwidth and latency of the CoMD applicatin running on the Mont-Blanc prototype. The measured node bandwidth of Mont-Blanc prototype is 640Mb/s with a latency of 145$\mu$s, which is far from optimal in both cases. It is evident that reaching the theoretical 1Gb/s bandwidth would improve the performance of CoMD of approximately 8% in this configuration. A further increase in bandwidth would saturate at 32% performance improvement at 3.2Gb/s of network bandwidth. Further, analysis reveals that CoMD is latency insensitive on the Mont-Blanc platform.

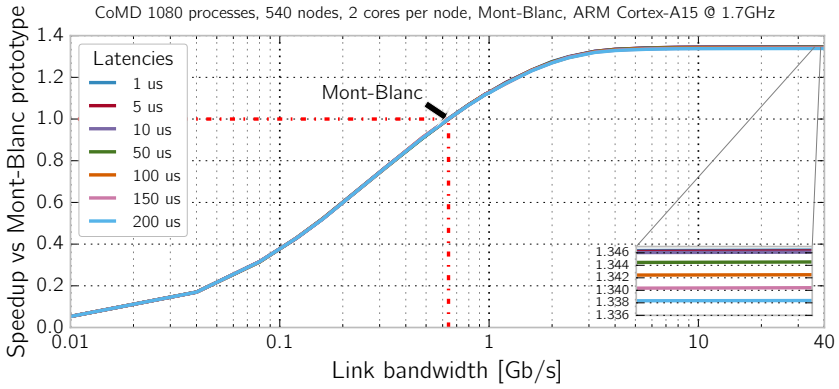



**Figure 8.5:** Performance projection for CoMD application on the Mont-Blanc prototype with different interconnect bandwidths and latencies.

### 8.2.2 NVIDIA Jetson

Figure 8.6 shows the speedup of CoMD on a hypothetical Mont-Blanc prototype powered by Jetson-like nodes, while sweeping interconnect bandwidth and latency. Note that the results are normalized to the execution on the real Mont-Blanc prototype architecture.

Interconnect parameters measurement on the Jetson platform shows a sustained interconnect bandwidth of 960Mb/s while providing a latency of 50$\mu$s. A one-to-one replacement of the Mont-Blanc node for a Jetson-like node would improve the performance of CoMD by 35%. If we could further improve interconnect elements of such a platform, mainly bandwidth, we could achieve a 72% performance increase compared to the Mont-Blanc prototype. We would need approximately 5Gb/s of sustained node bandwidth to achieve this level of performance.

### 8.2.3 ARM Juno

Figure 8.7 shows the performance prediction for CoMD in a hypothetical prototype powered by nodes with four-core Cortex-A57 SoCs running at 2.3GHz. Using a more advanced core IP technology further improves performance. Assuming the same bandwidth as in the Jetson platform, we could achieve a 42% performance increase over the base line Mont-Blanc prototype. With a better interconnect, a maximum of 96% performance improvement could be achieved with an interconnect bandwidth of approximately 6.5Gb/s.

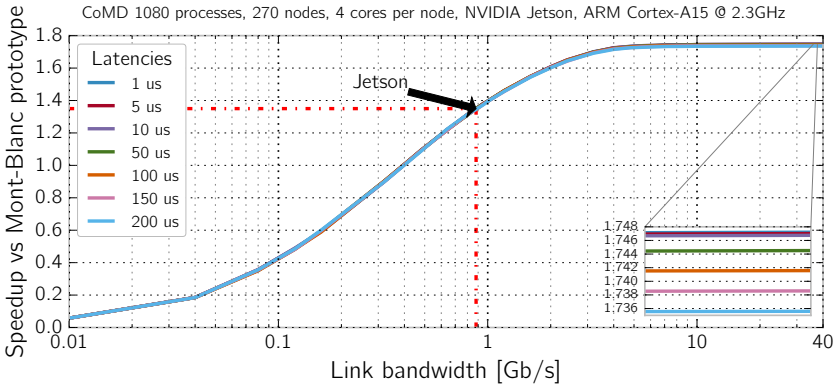



**Figure 8.6:** Performance projection for CoMD application on a hypothetical prototype powered by NVIDIA Jetson-like nodes with different interconnect bandwidths and latencies compared to the Mont-Blanc prototype.

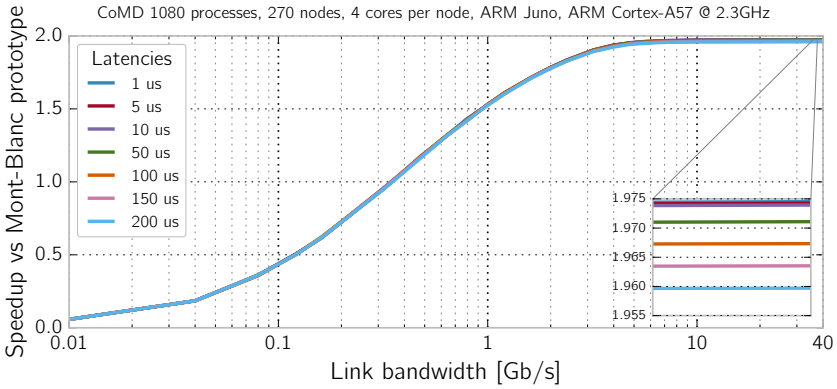



**Figure 8.7:** Performance projection for CoMD application on a hypothetical prototype powered by ARM Juno-like nodes with different interconnect bandwidths and latencies compared to the Mont-Blanc prototype.

### 8.2.4 NG Node

Figure 8.8 shows the performance prediction for CoMD running on a hypothetical platform featuring hypothetical nodes with eight of the recently-announced ARM Cortex-A72 cores. If we choose a 1Gb/s interconnect bandwidth, we could not use the full potential of such a platform. We would gain just a 150% speedup over the baseline Mont-Blanc prototype. To fully unleash the platform's potential, we should integrate a 10Gb/s interconnect interface, such as 10Gb Ethernet, and potentially achieve a 4x performance

improvement over the Mont-Blanc prototype.

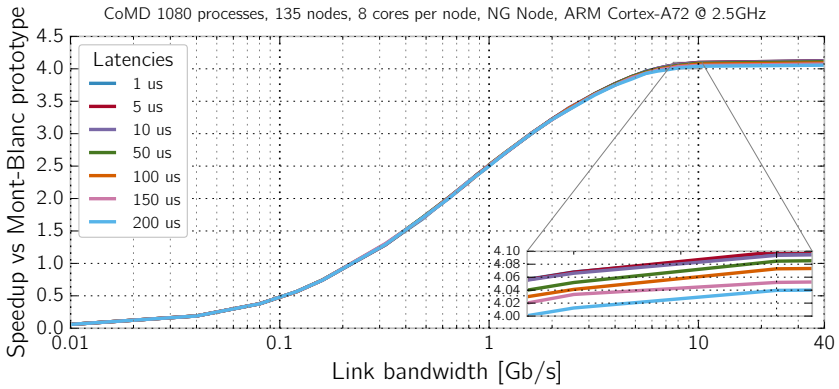



**Figure 8.8:** Performance projection for CoMD on a hypothetical prototype powered by NG Nodes with different interconnect bandwidths and latencies compared to the Mont-Blanc prototype.

To make a summary of the interconnect bandwidth effect and a comparison of the different target platforms for CoMD performance, Figure 8.9 shows per platform application execution speedup compared to the existing Mont-Blanc prototype. The different platform performances are shown for the case when we could fully utilise the capacity of 1Gb and 10Gb Ethernet interconnects. Using 10 Gb Ethernet improves application performance for every configuration compared to 1 Gb Ethernet. The improvement grows as we increase per node compute capability, by either increasing the number of cores per node and/or improving cores IP. For future systems, based on ARM Cortex-A72 cores and using commodity interconnect technology, 10 Gb Ethernet is a real necessity in order to achieve a balanced system design.

## 8.3 Power Projections

In this section, we present power projections for the target platforms listed in Table 8.1. We show results when varying different system parameters such as core types, number of cores in a SoC and network parameters.

### 8.3.1 Methodology

For the purpose of power projections we use the Mont-Blanc prototype system architecture as the base case. More precisely, we build a system power

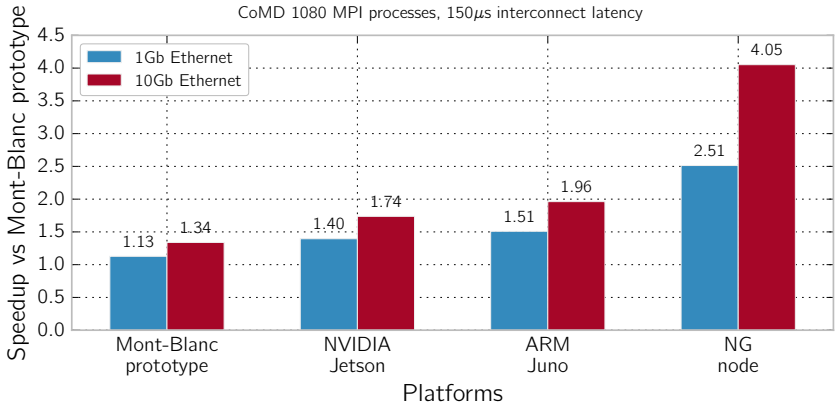



**Figure 8.9:** Achievable speedup of CoMD with upgraded node architecture, using commodity 1Gb and 10Gb Ethernet.

model around the Mont-Blanc architecture, taking into account power consumption from as many hardware components as possible.

In the prototype, we can measure power consumption on a per blade and per node basis. Thus, we do not have mechanisms to track power with finer granularity. For Ethernet controllers and voltage regulators, we use power figures provided in the components' datasheets. For CPU and memory, we use power consumption numbers obtained from a similar platform with power sensors for those components.

### CPU and Memory Power

We use the Hardkernel Odroid-XU3 board [67] which integrates the Samsung Exynos 5422 SoC [112] and offers power probes for CPUs and memory. The SoC integrates ARM Cortex-A15 and ARM Cortex-A7 both in quad core configurations, and 2 GB of LPDDR3. Figure 8.10 shows how power consumption varies with frequency and number of cores while running CoMD with one, two, and four processes. Sweeping the frequency allows for extrapolation of power figures beyond the available frequencies of the Odroid-XU3 board.

From the graph, we deduce a power of 2.51 W at 1.7 GHz for a dual core configuration of the Mont-Blanc node CPU. We use this figure in the power breakdown to build the node and system power models.

To model CPU power consumption of the Jetson platform, we extrapolate the power consumption shown in Figure 8.10 with a exponential fitting curve. From the graph We read the power consumption of 11.287W at 2.3 GHz and use this in our projections with the Jetson platform.

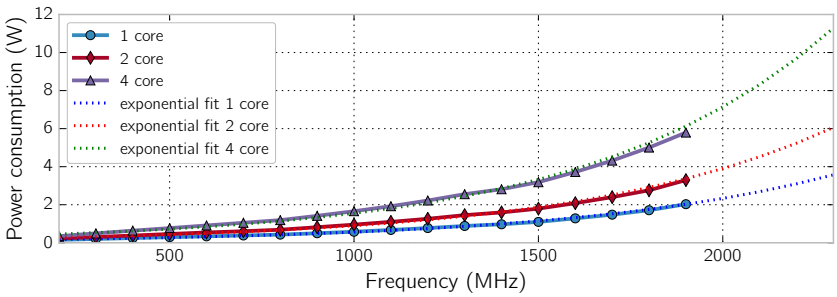



**Figure 8.10:** ARM Cortex-A15 power consumption vs. operational frequency: single and multi-core frequency sweep.

For the Juno powered system, with a quad core ARM Cortex-A57 CPU, we use CPU power figures obtained from a 3rd party benchmarking website [5]. The authors had an access and benchmarked a mobile platform (tablet) integrating a Samsung Exynos 7 Octa (5433) with an octa-core configuration [113] - quad core ARM Cortex-A57 and quad core ARM Cortex-A53 CPUs. Figure 8.11 shows those numbers. We again extrapolate power consumption beyond the maximum frequency to project the power consumption for a quad-core ARM Cortex-A57 CPU on the Juno platform. At 2.3 GHz we read 13.526 W for the quad-core ARM Cortex-A57 CPU integrated on the Samsung Exynos 7 Octa mobile SoC and use this in our projections with the Juno platform.

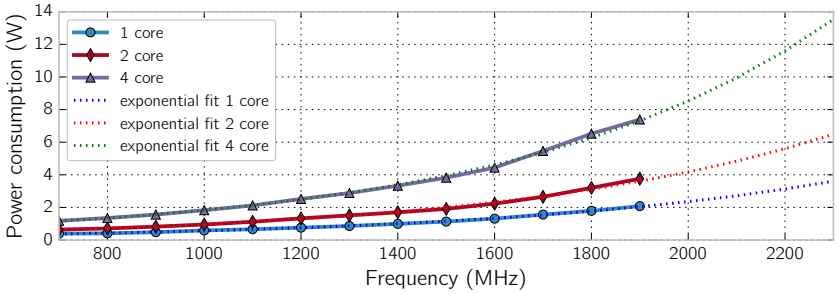



**Figure 8.11:** ARM Cortex-A57 power consumption vs. operational frequency: single and multi-core frequency sweep.

**Memory** For memory power consumption estimation, we repeat the same previous experiments as for the Cortex-A15 CPU power on the ODROID-XU3 board. Figure 8.12 depicts how memory power consumption varies with

different core operating frequencies, in both single and multi-core configurations. There is a linear relationship between core frequency and memory power. This makes sense as faster cores imply a higher memory load. The slope of the increase may vary when reaching a point of memory congestion, as it seems to be the case for four cores at high frequency[2].

For a dual-core configuration at 1.7 GHz (as in the Mont-Blanc prototype), we measure 53.4 mW of memory power consumption. In the Mont-Blanc prototype, we have double the memory capacity compared to the ODROID-XU3, so we multiply this number by a factor of two and use it for the power breakdown shown in the next section.

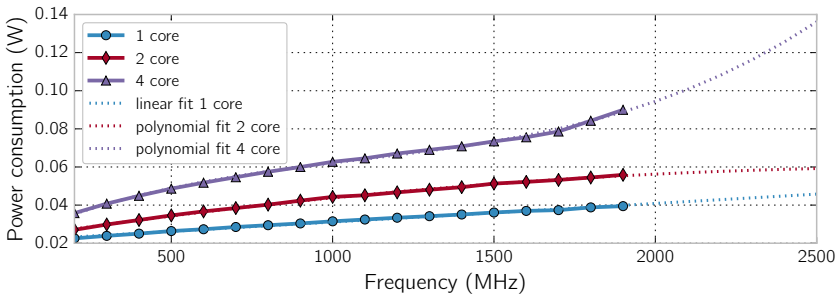



**Figure 8.12:** Memory power consumption as a function of core frequency.

In the case of Jetson- and Juno-like nodes, we take the power consumption from a quad-core execution, extrapolated to 2.3GHz, and multiply it by the factor of four to account for the increased memory capacity. Note that we assume 2GB of memory per core as we increase the number of cores, as it a standard in the present HPC systems.

Finally, for NG Node configuration, we use the extrapolated memory power consumption for a quad-core SoC at 2.5GHz, multiplied by a factor of 8 to account for the increased memory capacity.

### Blade Power Breakdown

The average measured power of a Mont-Blanc prototype *blade* when running CoMD with 1080 MPI processes is 235.98W. In the Figure 8.13 we show the power breakdown for the different components using our model. The 1080 MPI processes are scheduled on 540 nodes occupying 36 blades in total.

We build a blade power model after the blade power breakdown, as follows:

[2]This is the reason why we fit dual and quad-core results with polynomials of the 2$^{nd}$ and 3$^{rd}$ order respectively.

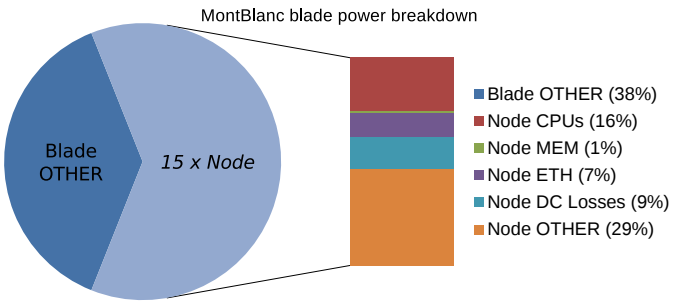



**Figure 8.13:** Mont-Blanc prototype blade power breakdown while running CoMD on 540 nodes, 2 MPI processes per node. Total blade average power consumption is 235.98 W.

$$P_{blade} = P_{bov} + \frac{N_{npb}}{\eta_{DC}} \times (P_{nov} + p_{cpu} + p_{mem} + p_{eth}) \tag{8.1}$$

where $P_{bov}$ (blade overhead) is a constant representing part of the blade power consumption we were unable to determine further, including cooling fans and network switch. We keep this parameter fixed across the experiments. $N_{npb}$ is a constant representing the *number of nodes per blade*. $\eta_{DC}$ is a constant modelling per node main linear voltage regulator efficiency. $P_{nov}$ (node overhead) models part of the node power we were unable to breakdown further, and is kept constant across the experiments. Variables $p_{cpu}$, $p_{mem}$, and $p_{eth}$ model CPU cores, memory and Ethernet controller average power consumption.

Finally, we model the entire system with the following equation:

$$P_{total} = n_{blades} \times P_{blade} = \frac{N_{PMPI}}{N_{npb} \times n_{cpn}} \times P_{blade} \tag{8.2}$$

where $N_{PMPI}$ is a constant representing the *total number of MPI processes* within an application execution. $N_{npb}$ is the same constant we use in Equation 8.1. The variable $n_{cpn}$ holds the information about the number of cores per SoC (or node, since we assume a single socket node architecture).

**Power Prediction Parameters** Table 8.7 presents the list of parameters we use to calculate Equation 8.1 and Equation 8.2 in order to project average power consumption and potential energy savings when running CoMD on the target platforms listed in Table 8.1

**Table 8.7:** Power consumption model of CoMD: list of used parameters for different node architectures.

| | Platforms | | | |
|---|---|---|---|---|
| **Parameter** | *Mont-Blanc* | *NVIDIA Jetson* | *ARM Juno* | *NG Node* |
| $n_{cpn}$ | 2 | 4 | 4 | 8 |
| $p_{cpu}$ [W] | 2.51 | 11.287 | 13.526 | 6.508 |
| $p_{mem}$ [mW] | 106.8 | 464.8 | 464.8 | 1089.3 |
| $N_{PMPI}$ | 1080 | | | |
| $N_{npb}$ | 15 | | | |
| $\eta_{DC}$ | 0.85 | | | |
| $P_{bov}$ [W] | 89.49 | | | |
| $P_{nov}$ [W] | 4.54 | | | |
| $p_{eth}$ [W] | {1.15, 5.1 [79]} for {1GbE, 10GbE} | | | |

### Results

In this section, we present power and energy figures for alternative Mont-Blanc node designs and 10GbE interconnect technology. Figure 8.14 shows a comparison between power consumption of hypothetical Mont-Blanc blades built around the platforms listed in Table 8.1. All the alternative blades experience an increase in power consumption due to increased multi-core density and improved interconnect technology.

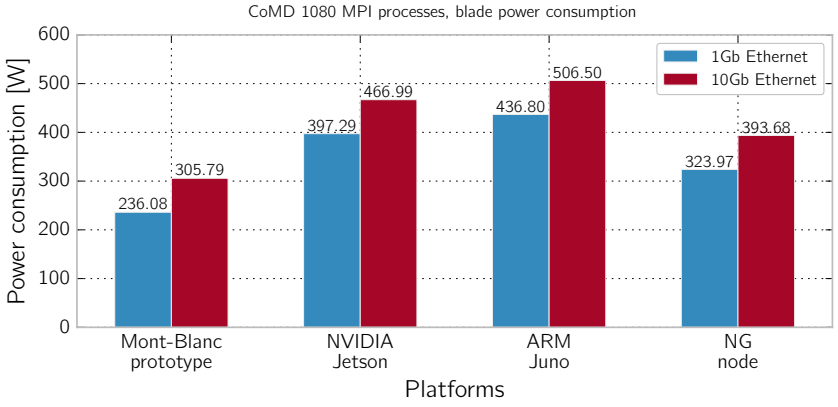



**Figure 8.14:** Power consumption comparison of alternative Mont-Blanc blades.

The increase in blade power consumption does not necessary lead to an

increased power consumption of the whole system, for the same total number of cores. Figure 8.15 depicts this behaviour. As the number of cores per node is increased, the number of blades is reduced. This way, savings are achieved by reducing the total power overheads of the blades. One interesting case is the Jetson platform which shows that with an increase in the number of cores from two to four, followed by an increase in their operating frequency from 1.7 to 2.3 GHz, and doubling the memory capacity to 8GB, we could afford to integrate 10 Gb Ethernet and stay within the same power budget of the current Mont-Blanc prototype. However, if the power consumption is the major concern, we should consider the NG Node which could potentially reduce power consumption by 59% for the same number of cores as in the Mont-Blanc prototype.

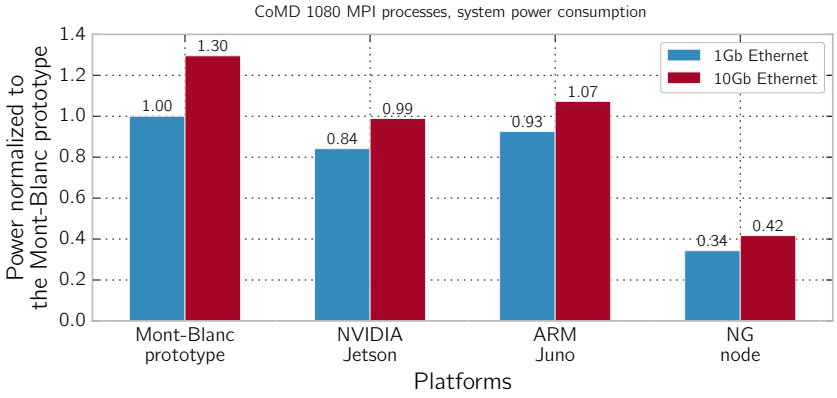



**Figure 8.15:** Power consumption comparison of alternative Mont-Blanc systems.

At the end, we show energy consumption comparison in Figure 8.16. As expected, all the alternative configurations consume less energy than the Mont-Blanc prototype. In terms of energy consumption NVIDIA Jetson and ARM Juno platforms are almost even. However, the Juno alternative could achieve potentially better speedup compared to Jetson - 1.96 vs 1.74. The NG Node alternative would be the best choice given both energy consumption and speedup over the Mont-Blanc prototype. It could achieve 88% energy savings and 4.05× speedup over the base case, for the same number of cores and the same system integration.

## 8.4 Conclusions

From the performance and power projection studies we can draw a couple of guidelines for next generation systems design.

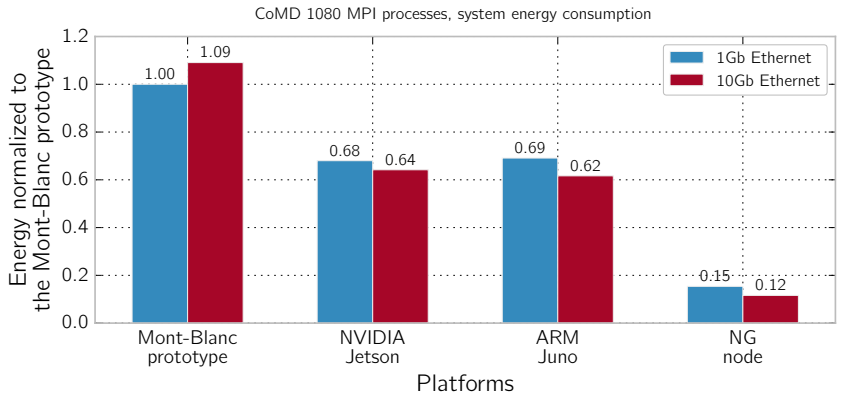



**Figure 8.16:** Energy consumption comparison of alternative Mont-Blanc systems.

The performance study shows that each generation of mobile cores keeps improving the performance. We have shown that using the current most commonly used mobile cores IP, namely ARM Cortex-A57, we could achieve a performance improvement over the Mont-Blanc prototype system by a factor of 1.96. Even more promising are the current state-of-the-art mobile IP cores, namely ARM Cortex-A72, which would further improve the performance of an HPC cluster built around mobile SoCs by a factor of 4.05 over the Mont-Blanc prototype.

In line with the performance study is the power projection study, showing that the new silicon processes which allow for higher per SoC core count, together with new core types, improve on the overall system energy efficiency. Increasing core count from two to eight and using current state-of-the-art mobile cores, could potentially save 88% of the energy compared to a current Mont-Blanc prototype.

However, all those CPU performance improvements require improvement on the side of interconnect. As we have demonstrated there is a need for higher bandwidth interconnect technologies as 1 Gb Ethernet would not suffice. More precisely, we have to provide at least 10 Gb of per node bandwidth for the systems built with the current state-of-the-art CPU core IPs in order to have a balanced system. Relying on the commodity networking technology, 10 Gb Ethernet would be a viable candidate for the node interconnect.

# 9 Conclusions

In this thesis we have shown that building HPC systems using mobile SoCs, capable of running production level HPC applications is feasible. However, due to a lower per socket performance and limited I/O resources of a mobile SoC, there are significant challenges. Throughout our work, one can notice the evolution of mobile SoCs – both in terms of compute performance and available memory bandwidth which needs to closely follow the performance increase.

One of the findings is that mobile SoCs need to offer 4× higher aggregate performance in order to match a production level supercomputer. This can be achieved using higher core count i.e. eight, next-generation IP cores, and exploiting on-chip GPUs for accelerating the computation.

Further, in order to interconnect them together, they need to provide low-latency high-bandwidth interfaces. USB3.0 is not suitable for HPC since it introduces an additional level of latency, but it however the most common peripheral interface found in mobile SoCs. Currently, the only viable solution for this purpose if the use of PCIe, as found in most NVIDIA Tegras SoCs.

Unlike server processors, mobile SoCs do not provide a support for multi-socket systems. Technology such as Intel QPI, AMD HyperTransport, would be beneficial for mobile SoCs - allowing for multisocket solutions.

Sizing the memory system to a rule-of-a-thumb specification of 2-4 GB/core is finally possible with the 64-bit ARM architecture. But, providing enough of memory bandwidth would be a challenging task, since it requires additional memory channels. Also, those memory channels have to support ECC

since it is of a paramount importance for high-performance systems.

The bright side is the software eco-system. When we began our study with mobile SoCs, the HPC software stack for ARM platform was almost non-existent. Today ARM provides a set of HPC optimized libraries, just like Intel and other HPC vendors do. This is already recognized by Fujitsu who recently, as of the latest ISC'16 conference, revealed their plans on building exascale supercomputer based on ARMv8 ISA, while relying on their proprietary microarchitecture.

It is unlikely that mobile SoC will takeover the lead from well established HPC architectures but, there is a opportunity for the next-generation departments size machines be built from mobile SoCs. In order this to happen, all flaws must be addressed, and there is enough space to balance cost vs performance in mobile domain.

## 9.1 Future Work

A logical next step in this study is to actually make a case for HPC-ready mobile SoC, and software eco-system. Author's vision is as follows:

- Exploring the possibility of using heterogeneous multicore to offload OS, runtime, and interrupt handling to designated low-power low-performance cores – making a case for big.Little platform use for HPC.
- Even further exploration of heterogeneity in mobile SoCs, and enable programmability of on-chip accelerators beyond the GPU i.e. DSPs.
- Taking the advantage of dark-silicon and conduct a design space exploration for integrating the bare minimum set of on-chip resources to support efficient HPC.
- Enabling multi-mode operation of an HPC-ready mobile SoC – exploration of resource managements techniques, both in hardware and software, in order to enable the proper operation depending on the need, and power and thermal budget.

# List of publications

Main contributions:

- N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. O. Vilarrubi, C. Gomez, L. Backes, D. Nieto, H. Servat, X. Martorell, J. Labarta, E. Ayguade, C. Adeniyi-Jones, S. Derradji, H. Gloaguen, P. Lanucara, N. Sanna, J.-F. Mehaut, K. Pouget, B. Videau, E. Boyer, M. Allalen, A. Auweter, D. Brayford, D. Tafani, V. Weinberg, D. Brömmel, R. Halver, J. H. Meinke, R. Beivide, M. Benito, E. Vallejo, M. Valero, and A. Ramirez, **The Mont-Blanc prototype: An Alternative Approach for HPC Systems**, In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'16, Salt Lake City, UT, USA, 2016, IEEE/ACM.

  🏆***Nominated for The Best Paper Award.***

- N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez. **Tibidabo: Making the case for an ARM-based HPC system**. *Future Generation Computer Systems*, vol. 36, pp. 322 – 334, 2014, Elsevier.

- N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, **Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?** In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'13, Denver, CO, USA, 2013, IEEE/ACM.

  🏆***Winner of The Best Student Paper Award.***

- N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, and A. Ramirez. **The Low-power Architecture Approach Towards Exascale Computing**. *Journal of Computational Science*, vol. 4, no. 6, pp. 439 – 443, 2013, Elsevier

- N. Rajovic, A. Rico, J. Vipond, I. Gelado, N. Puzovic, and A. Ramirez. **Experiences with Mobile Processors for Energy Efficient HPC**. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE'13, Grenoble, France, 2013. EDA Consortium.

- N. Rajovic, N. Puzovic, L. Vilanova, C. Villavieja, and A. Ramirez. **The Low-power Architecture Approach Towards Exascale Computing**. In *Proceedings of the Second Workshop on Scalable Algorithms for Large-scale Systems*, ScalA'11, Seattle, WA, USA, 2011. ACM.

Related side publications:

- I. Grasso, P. Radojkovic, N. Rajovic, I. Gelado, and A. Ramirez. **Energy Efficient HPC on Embedded SoCs: Optimization Techniques for ARM Mali GPU**. In *Proceedings of International Parallel and Distributed Processing Symposium*, IPDPS'14, Phoenix, AZ, USA, 2014, IEEE.

- D. Göddeke, D. Komatitsch, M. Geveler, D. Ribbrock, N. Rajovic, N. Puzovic, and A. Ramirez. **Energy efficiency vs. performance of the numerical solution of PDEs: An application study on a low-power ARM-based cluster**. *Journal of Computational Physics*, vol. 237, pp. 132 – 150, 2013, Elsevier.

# Bibliography


[1] Abdurachmanov, D., Bockelman, B., Elmer, P., Eulisse, G., Knight, R., and Muzaffar, S. Heterogeneous high throughput scientific computing with APM X-gene and Intel Xeon Phi. In *Journal of Physics: Conference Series* (2015), vol. 608, IOP Publishing, p. 012033. [page 14]

[2] Adiga, N. R., Almási, G., Almasi, G. S., Aridor, Y., Barik, R., Beece, D., Bellofatto, R., Bhanot, G., Bickford, R., Blumrich, M., et al. An overview of the BlueGene/L supercomputer. In *Supercomputing, ACM/IEEE 2002 Conference* (2002), IEEE. [page 13]

[3] Alam, S., Barrett, R., Bast, M., Fahey, M. R., Kuehn, J., McCurdy, C., Rogers, J., Roth, P., Sankaran, R., Vetter, J. S., Worley, P., and Yu, W. Early evaluation of IBM BlueGene/P. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), IEEE Press. [page 13]

[4] AMD. AMD Opteron A1100 SoC Series. https://web.archive.org/web/20161127212702/https://www.amd.com/Documents/A-Hierofalcon-Product-Brief.pdf. [page 13]

[5] Anandtech. Cortex-A57 - Performance and Power - ARM A53/A57/T760 investigated - Samsung Galaxy Note 4 Exynos Review. http://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/6. Accessed: 2015-06-09. [page 113]

[6] Applied Micro. APM "X-Gene" Launch Press Briefing. https://web.archive.org/web/20120813151248/http://www.apm.com/global/x-gene/docs/X-GeneOverview.pdf, 2012. [page 13]

[7] ARM Connected Community. What is exact double precision performance for Mali T628 MP6 (Arndale Octa Board)? https://web.archive.org/web/20160303143357/https://community.arm.com/message/18218, 4 2014. [page 83]

[8] ARM Ltd. ARM Announces 2GHz Capable Cortex-A9 Dual Core Processor Implementation. http://www.arm.com/about/newsroom/25922.php. [page 42]

[9] ARM Ltd. ARM Holding Annual Report & Accounts 2012. http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9MTczODc5fENoaWxkSUQ9LTF8VHlwZT0z&t=1. [page 6]

[10] ARM Ltd. CoreLink CCN-504 Cache Coherent Network. http://www.arm.com/products/system-ip/interconnect/corelink-ccn-504-cache-coherent-network.php. [pages 7, 59, and 60]

[11] ARM Ltd. CoreTile Express$^{TM}$A15×2 A7×3 Technical Reference Manual. https://web.archive.org/web/20160509134259/http://infocenter.arm.com/help/topic/com.arm.doc.ddi0503h/DDI0503H_v2p_ca15_a7_tc2_trm.pdf. [page 53]

[12] ARM Ltd. Cortex-A72 processor. http://www.arm.com/products/processors/cortex-a/cortex-a72-processor.php. Accessed: 2015-06-09. [page 108]

[13] ARM Ltd. Cortex-A9 Processor. http://www.arm.com/products/processors/cortex-a/cortex-a9.php. [page 5]

[14] ARM Ltd. Mali Graphics plus GPU Compute. http://www.arm.com/products/multimedia/mali-graphics-plus-gpu-compute/index.php. [page 8]

[15] ARM Ltd. Mali-T658. http://www.arm.com/products/mali-t658.php. [page 8]

[16] ARM Ltd. The ARM® NEON$^{TM}$general purpose SIMD engine. http://www.arm.com/products/processors/technologies/neon.php. [page 6]

[17] ARM Ltd. VFPv3 Floating Point Unit. http://www.arm.com/products/processors/technologies/vector-floating-point.php. [page 6]

[18] ARM Ltd. Virtualization Extensions and Large Physical Address Extensions. http://www.arm.com/products/processors/technologies/virtualization-extensions.php. [page 7]

[19] ARM Ltd. Virtualization Extensions and Large Physical Address Extensions. http://www.arm.com/products/processors/technologies/virtualization-extensions.php. [page 54]

[20] ARM Ltd. Mali-T604 GPU Architecture. http://www.arm.com/products/multimedia/mali-graphics-plus-gpu-compute/mali-t604.php, 2013. [page 60]

[21] Samsung Exynos 5 Dual Arndale Board. http://www.arndaleboard.org/, 2013. [page 68]

[22] Badia, R. M., Labarta, J., Gimenez, J., and Escale, F. Dimemas: Predicting mpi applications behavior in grid environments. In *Workshop on Grid Applications and Programming Tools (GGF8)* (2003), vol. 86, pp. 52–62. [pages 22, 52, and 92]

[23] Barcelona Supercomputing Center. Basic analysis suite. https://ftp.tools.bsc.es/basicanalysis/basicanalysis-0.2-revised.tar.gz. [page 25]

[24] Barcelona Supercomputing Center. Clustering suite. https://ftp.tools.bsc.es/clusteringsuite/ClusteringSuite-2.6.6-linux-x86-64.tar.gz. [page 25]

[25] Barcelona Supercomputing Center. Extrae tracing infrastructure. https://tools.bsc.es/extrae. [page 24]

[26] Barcelona Supercomputing Center. MareNostrum III (2013) System Architecture. https://web.archive.org/web/20160303114630/https://www.bsc.es/marenostrum-support-services/mn3. [page 84]

[27] Barker, K. J., Davis, K., Hoisie, A., Kerbyson, D. J., Lang, M., Pakin, S., and Sancho, J. C. Entering the petaflop era: The architecture and performance of roadrunner. In *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2008), pp. 1–11. [page 11]

[28] Berendsen, H., van der Spoel, D., and van Drunen, R. Gromacs: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications 91*, 1 (1995), 43–56. [pages 21 and 41]

[29] Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., nneau, M. D., Franzon, P., Harrod, W., Hill, K., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R. S., Yelick, K., and Kogge, P. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. In *DARPA Technical Report* (2008). [page 4]

[30] The BigDFT Scientific Application. http://bigdft.org/, 2015. [pages 21 and 90]

[31] Blem, E., Menon, J., and Sankaralingam, K. Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures. In *19th IEEE International Symposium on High Performance Computer Architecture (HPCA 2013)* (2013). [page 14]

[32] Bolaria, J. Cortex-A57 Extends ARM's Reach. *Microprocessor Report* (Nov. 2012), 1–5. [page 7]

[33] Byrne, J. ARM CoreLink Fabric Links 16 CPUs. *Microprocessor Report* (Oct. 2012), 1–3. [pages 7 and 59]

[34] Calxeda. Calxeda Quad-Node EnergyCard. http://web.archive.org/web/20131029200758/http://www.calxeda.com/wp-content/uploads/2012/06/EnergyCard-Product-Brief-612.pdf. [pages 13 and 60]

[35] Carrington, L., Komatitsch, D., Laurenzano, M., Tikir, M. M., Michéa, D., Le Goff, N., Snavely, A., and Tromp, J. High-frequency simulations of global seismic wave propagation using specfem3d_globe on 62k processors. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), IEEE Press, p. 60. [page 47]

[36] Casas, M., Badia, R. M., and Labarta, J. Automatic analysis of speedup of MPI applications. In *Proceedings of the 22nd Annual International Conference on Supercomputing, ICS 2008* (2008), pp. 349–358. [pages 23, 25, and 97]

[37] Case, L. Tegra X1 Doubles GPU Performance. *Microprocessor Report* (Jan. 2015). [page 8]

[38] Cavium. ThunderX$^{TM}$. https://web.archive.org/web/20160310114848/http://www.cavium.com/pdfFiles/ThunderX_PB_p12_Rev1.pdf, 2013. [page 13]

[39] Cepulis, D. ISC16 Recap - Fujitsu Takes the Stage. https://community.arm.com/groups/processors/blog/2016/06/27/isc16-recap-fujitsu-takes-the-stage. [page 15]

[40] Charles Zhang, Phytium Technology Co., Ltd. Mars: A 64-core ARMv8 Processor. https://web.archive.org/web/20160310155325/http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.24-Monday-Epub/HC27.24.30-HP-Cloud-Comm-Epub/HC27.24.321-64core-Zhang-phytium-v1.0.pdf, 2015. [page 13]

[41] Chen, T., Rghavan, R., Dale, J., and Iwata, E. Cell Broadband Engine Architecture and its first implementation—A performance view. *IBM Journal of Research and Development 51*, 5 (sep 2007), 559 –572. [page 11]

[42] Coral collaboration benchmark codes. https://asc.llnl.gov/CORAL-benchmarks/, 2013. [page 90]

[43] Data Center Knowledge. PayPal Deploys ARM Servers in Data Centers. https://web.archive.org/web/20160310160416/http://www.datacenterknowledge.com/archives/2015/04/29/paypal-deploys-arm-servers-in-data-centers/, 2015. [page 14]

[44] de Verdière, G. Hydrobench. https://github.com/HydroBench/Hydro, 2014. [page 21]

[45] Demler, M. Tegra K1 Adds PC Graphics to Mobile. *Microprocessor Report* (Jan. 2014). [page 8]

[46] Dhodapkar, A., Lauterbach, G., Lie, S., Mallick, D., Bauman, J., Kanthadai, S., Kuzuhara, T., Shen, G., Xu, M., and Zhang, C. SeaMicro SM10000-64 server: Building datacenter servers using cell phone chips. In *2011 IEEE Hot Chips 23 Symposium (HCS)* (Aug 2011), pp. 1–18. [page 13]

[47] Dongarra, J., Luszczek, P., and Petitet, A. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience 15*, 9 (2003), 803–820. [pages 21, 32, and 40]

[48] Duran, A., Ayguadé, E., Badia, R. M., Labarta, J., Martinell, L., Martorell, X., and Planas, J. Ompss: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters 1*, 2 (2011), 173–193. [pages 19 and 82]

[49] Eisenmenger, F., Hansmann, U. H. E., Hayryan, S., and Hu, C.-K. [SMMP] A modern package for simulation of proteins. *Computer Physics Communications 138*, 2 (2001), 192–212. [pages 21 and 90]

[50] Eisenmenger, F., Hansmann, U. H. E., Hayryan, S., and Hu, C.-K. An enhanced version of SMMP—open-source software package for simulation of proteins. *Computer Physics Communications 174*, 5 (2006), 422–429. [pages 21 and 90]

[51] Energy Price Statistics. https://web.archive.org/web/20160906232718/http://ec.europa.eu/eurostat/statistics-explained/index.php/Energy_price_statistics. [page 4]

[52] ExMatEx. CoMD Proxy Application. http://www.exmatex.org/comd.html. [pages 21 and 90]

[53] Feng, W.-c., and Cameron, K. The green500 list: Encouraging sustainable supercomputing. *Computer 40*, 12 (2007). [page 3]

[54] Frigo, M., and Johnson, S. G. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing* (1998), vol. 3, IEEE, pp. 1381–1384. [page 39]

[55] Fürlinger, K., Klausecker, C., and Kranzlmüller, D. The appletv-cluster: Towards energy efficient parallel computing on consumer electronic devices. *Whitepaper, Ludwig-Maximilians-Universitat* (2011). [page 13]

[56] Fürlinger, K., Klausecker, C., and Kranzlmüller, D. Towards energy efficient parallel computing on consumer electronic devices. In *Information and Communication on Technology for the Fight against Global Warming*. Springer, 2011, pp. 1–9. [page 13]

[57] Genovese, L., Videau, B., Ospici, M., Deutsch, T., Goedecker, S., and Méhaut, J.-F. Daubechies Wavelets for High Performance Electronic Structure Calculations: the BigDFT Project. In *Compte-Rendu de l'Académie des Sciences, Calcul Intensif.* (2010), Académie des Sciences. [pages 21 and 90]

[58] Germann et al, T. C. Codesign Molecular Dynamics (CoMD) Proxy App Deep Dive. *Exascale Research Conference* (2011). [page 105]

[59] Giannozzi, P., Baroni, S., Bonini, N., Calandra, M., Car, R., Cavazzoni, C., Ceresoli, D., Chiarotti, G. L., Cococcioni, M., Dabo, I., Corso, A. D., de Gironcoli, S., Fabris, S., Fratesi, G., Gebauer, R., Gerstmann, U., Gougoussis, C., Kokalj, A., Lazzeri, M., Martin-Samos, L., Marzari, N., Mauri, F., Mazzarello, R., Paolini, S., Pasquarello, A., Paulatto, L., Sbraccia, C., Scandolo, S., Sclauzero, G., Seitsonen, A. P., Smogunov, A., Umari, P., and Wentzcovitch, R. M. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter 21*, 39 (2009), 395502. [pages 21 and 90]

[60] Girona, S., Labarta, J., and Badia, R. M. Validation of Dimemas Communication Model for MPI Collective Operations. In *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface* (2000), pp. 39–46. [page 23]

[61] Goglin, B. Design and implementation of Open-MX: High-performance message passing over generic Ethernet hardware. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (2008), IEEE, pp. 1–7. [pages 44 and 90]

[62] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. [page 25]

[63] Gonzalez, J., Gimenez, J., Casas, M., Moreto, M., Ramirez, A., Labarta, J., and Valero, M. Simulating whole supercomputer applications. *IEEE Micro* (2011), 32–45. [page 101]

[64] Grass, T., Allande, C., Armejach, A., Rico, A., Ayguadé, E., Labarta, J., Valero, M., Casas, M., and Moreto, M. Musa: A multi-level simulation approach for next-generation hpc machines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2016), SC '16, pp. 45:1–45:12. [page 58]

[65] Gwennap, L. Broadcom Bares Muscular ARM: Quad-Issue ARMv8 CPU Targets Xeon-Class Performance. *Microprocessor Report* (Oct. 2013), 1–4. [page 13]

[66] Göddeke, D., Komatitsch, D., Geveler, M., Ribbrock, D., Rajovic, N., Puzovic, N., and Ramirez, A. Energy efficiency vs. performance of the numerical solution of pdes: An application study on a low-power arm-based cluster. *Journal of Computational Physics 237* (2013), 132 – 150. [pages 4 and 45]

[67] Hardkernel. ODROID-XU3. https://web.archive.org/web/20160324175751/http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127. Accessed: 2016-04-25. [page 112]

[68] Henning, J. L. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News 34*, 4 (Sept. 2006), 1–17. [pages 18 and 32]

[69] Heroux, M. A., Doerfler, D. W., Crozier, P. S., Willenbring, J. M., Edwards, H. C., Williams, A., Rajan, M., Keiter, E. R., Thornquist, H. K., and Numrich, R. W. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574 3* (2009). [pages 21 and 90]

[70] HPCwire. New Mexico to Pull Plug on Encanto, Former Top5 Supercomputer. https://web.archive.org/web/20160602162230/http://www.hpcwire.com/2012/07/12/new_mexico_to_pull_plug_on_encanto_former_top_5_supercomputer, 7 2012. [page 4]

[71] HPCwire. Requiem for Roadrunner. https://web.archive.org/web/20130511112654/http://www.hpcwire.com/hpcwire/2013-04-01/requiem_for_roadrunner.html, 4 2013. [page 4]

[72] IBM Systems and Technology. IBM System Blue Gene/Q Data Sheet, November 2011. [page 13]

[73] IHS iSuppli News Flash. Low-End Google Nexus 7 Carries $152 BOM, Teardown Reveals. https://web.archive.org/web/20160527073453/https://technology.ihs.com/408150/low-end-google-nexus-7-carries-152-bom-teardown-reveals. [page 6]

[74] Imagination Technologies Limited. PowerVR Graphics. http://www.imgtec.com/powervr/powervr-graphics.asp. [page 8]

[75] Intel. Intel ATOM S1260. https://web.archive.org/web/20160601151758/http://ark.intel.com/products/71267/Intel-Atom-Processor-S1260-1M-Cache-2_00-GHz. [page 6]

[76] Intel. Intel MPI Benchmarks 3.2.4. http://software.intel.com/en-us/articles/intel-mpi-benchmarks. [pages 20 and 44]

[77] Intel. Intel Xeon Processor E5-2670. https://web.archive.org/web/20160601144459/http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI. [page 6]

[78] Intel. Intel® 82574 GbE Controller Family. https://web.archive.org/web/20160315003153/http://www.intel.com/content/dam/doc/datasheet/82574l-gbe-controller-datasheet.pdf. [page 42]

[79] Intel. Intel® 82599 10 Gigabit Ethernet Controller: Datasheet. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/82599-10-gbe-controller-datasheet.pdf. Accessed: 2015-06-09. [page 116]

[80] Kandadio, S. N., and He, X. Performance of HPC Applications over Infiniband, 10 Gb and 1 Gb Ethernet. http://www.chelsio.com/assetlibrary/whitepapers/HPC-APPS-PERF-IBM.pdf. [page 90]

[81] Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B. L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., and Still, C. Exploring Traditional and Emerging Parallel Programming Models using a Proxy Application. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing* (Boston, USA, May 2013). [pages 21 and 90]

[82] Karlin, I., Keasler, J., and Neely, R. Lulesh 2.0 updates and changes. Tech. Rep. LLNL-TR-641973, Lawrence Livermore National Laboratory, August 2013. [pages 21 and 90]

[83] Komatitsch, D., and Tromp, J. Introduction to the spectral element method for three-dimensional seismic wave propagation. *Geophysical Journal International 139*, 3 (1999), 806–822. [page 41]

[84] Lang, W., Patel, J., and Shankar, S. Wimpy node clusters: What about non-wimpy workloads? In *Proceedings of the Sixth International Workshop on Data Management on New Hardware* (2010), ACM, pp. 47–55. [page 12]

[85] Lavallée, P.-F., de Verdière, G. C., Wautelet, P., Lecas, D., and Dupays, J.-M. Porting and optimizing hydro to new platforms and programming paradigms–lessons learnt, 2012. [page 21]

[86] Li, S., Lim, K., Faraboschi, P., Chang, J., Ranganathan, P., and Jouppi, N. P. System-level integrated server architectures for scale-out datacenters. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* (2011), ACM, pp. 260–271. [page 14]

[87] The LINPACK 1000x1000 benchmark program. http://www.netlib.org/benchmark/linpackc. [page 30]

[88] Marjanović, V., Labarta, J., Ayguadé, E., and Valero, M. Overlapping communication and computation by using a hybrid mpi/smpss approach. In *Proceedings of the 24th ACM International Conference on Supercomputing* (2010), ACM, pp. 5–16. [page 61]

[89] Mattson, T., and Henry, G. An Overview of the Intel TFLOPS Supercomputer. *Intel Technology Journal 2*, 1 (1998). [pages 3 and 11]

[90] McCalpin, J. D. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25. [pages 18 and 32]

[91] Meinke, J. H., Mohanty, S., Eisenmenger, F., and Hansmann, U. H. E. [SMMP] v. 3.0—Simulating proteins and protein interactions in Python and Fortran. *Computer Physics Communications 178*, 6 (2008), 459–470. [pages 21 and 90]

[92] Micron. DDR2 SDRAM System-Power Calculator. https://web.archive.org/web/20100513031210/http://micron.com/support/dram/power_calc.html. [page 42]

[93] Ming Lei. USBNET: increase max rx/tx qlen for improving USB3 throuput. https://web.archive.org/web/20160308154732/https://github.com/torvalds/linux/commit/452c447a497dce3c9faeb9ac7f2e1ff39232876b, 2013. [page 89]

[94] The Mont-Blanc project website. http://montblanc-project.eu. [pages 18 and 62]

[95] The MQTT Protocol. http://mqtt.org, 2014. [page 83]

[96] Munshi, A., Ginsburg, D., and Shreiner, D. *OpenGL ES 2.0 programming guide*. Pearson Education, 2008. [page 8]

[97] Nakamura, Y., and Stüben, H. BQCD-Berlin quantum chromodynamics program. *arXiv preprint arXiv:1011.0199* (2010). [pages 21 and 90]

[98] Nakanishi, H., Nishigaki, N., Tachibana, K., and Shinagawa, T. WT210/WT230 Digital Power Meters. https://web.archive.org/web/20130728084210/http://c418683.r83.cf2.rackcdn.com/uploaded/WT210.pdf, 2003. [pages 22 and 40]

[99] Nakashima, H., Nakamura, H., Sato, M., Boku, T., Matsuoka, S., Takahashi, D., and Hotta, Y. Megaproto: 1 tflops/10kw rack is feasible even with only commodity technology. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference* (2005), IEEE, pp. 28–28. [page 12]

[100] NVIDIA. NVIDIA Tegra mobile processor. http://www.nvidia.com/object/tegra-4-processor.html. [page 8]

[101] NVIDIA. Tegra2 die photo. https://web.archive.org/web/20160503180038/http://images.anandtech.com/reviews/gadgets/LG/Optimus2X/SoC.jpg. [page 43]

[102] NVIDIA. Compute Unified Device Architecture Programming Guide. [page 19]

[103] NVIDIA. Variable SMP (4-PLUS-1™) – A Multi-Core CPU Architecture for Low Power and High Performance. *White Paper* (2011). [page 64]

[104] NVIDIA. CARMA - CUDA® Development Kit For ARM®. http://www.nvidia.com/object/carma-devkit.html, 2012. [page 64]

[105] PCWorld. Qualcomm enters server CPU market with 24-core ARM chip. http://www.pcworld.com/article/2990868/qualcomm-enters-server-cpu-market-with-24-core-arm-chip.html, 2015. [page 13]

[106] Pillet, V., Labarta, J., Cortes, T., and Girona, S. Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: Transputer and occam Developments* (1995), vol. 44, pp. 17–31. [pages 23 and 25]

[107] Planas, J., Badia, R. M., Ayguade, E., and Labarta, J. Self-adaptive ompss tasks in heterogeneous environments. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on* (2013), IEEE, pp. 138–149. [page 82]

[108] Quanta. STRATOS S900 Series, S900-X31A: Extremely Low Power, High Density 3U Microserver. http://www.qct.io/index.php?route=account/download/download&order_download_id=40. [page 13]

[109] Rajovic, N., Rico, A., Vipond, J., Gelado, I., Puzovic, N., and Ramirez, A. Experiences with mobile processors for energy efficient HPC. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2013), EDA Consortium, pp. 464–468. [page 84]

[110] Ramirez, A., Prat, O., Labarta, J., and Valero, M. Performance Impact of the Interconnection Network on MareNostrum Applications. In *1st Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip* (2007). [page 23]

[111] Rosas, C., Giménez, J., and Labarta, J. Scalability prediction for fundamental performance factors. *Supercomputing frontiers and innovations 1*, 2 (2014). [pages 23, 25, and 97]

[112] Samsung. Exynos 5 Octa (5422). https://web.archive.org/web/20140727080227/http://www.samsung.com/global/business/semiconductor/product/application/detail?productId=7978&iaId=2341. [page 112]

[113] Samsung. Samsung Exynos 7 Octa. http://www.samsung.com/global/business/semiconductor/minisite/Exynos/w/solution.html#?v=7octa. [page 113]

[114] Samsung. Samsung Exynos5 Dual White Paper. https://web.archive.org/web/20151026121220/http://www.samsung.com/global/business/semiconductor/minisite/Exynos/data/Enjoy_the_Ultimate_WQXGA_Solution_with_Exynos_5_Dual_WP.pdf. [page 60]

[115] SECO. QuadMo747-X/T20. https://web.archive.org/web/20120613172914/http://www.seco.com/en/item/quadmo747-x_t20. [page 38]

[116] SECO. SECOCQ7-MXM. https://web.archive.org/web/20120524090006/http://www.seco.com/en/item/secocq7-mxm. [page 38]

[117] Sharkawi, S., Desota, D., Panda, R., Indukuru, R., Stevens, S., Taylor, V., and Wu, X. Performance projection of hpc applications using spec cfp2006 benchmarks. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2009). [pages 25, 101, and 106]

[118] SMSC. LAN9514/LAN9514i: USB 2.0 Hub and 10/100 Ethernet Controller. https://web.archive.org/web/20110716100101/http://www.smsc.com/media/Downloads_Public/Data_Sheets/9514.pdf. [page 42]

[119] Stanley-Marbell, P., and Cabezas, V. C. Performance, power, and thermal analysis of low-power processors for scale-out systems. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)* (2011), pp. 863–870. [page 12]

[120] Sutmann, G., Westphal, L., and Bolten, M. Particle based simulations of complex systems with mp2c : Hydrodynamics and electrostatics. *AIP Conference Proceedings 1281*, 1 (2010), 1768–1772. [pages 21 and 90]

[121] Texas Instruments. AM5K2E04/02 Multicore ARM KeyStone II System-on-Chip (SoC) Data Manual. https://web.archive.org/web/20160530133009/http://www.ti.com/lit/ds/symlink/am5k2e04.pdf, November 2012. [page 45]

[122] TOP500. Top500®supercomputer cites. http://www.top500.org/. [pages 1, 2, and 3]

[123] Turek, S., Göddeke, D., Becker, C., Buijssen, S. H. M., and Wobker, H. Feast—realization of hardware-oriented numerics for hpc simulations with finite elements. *Concurrency and Computation: Practice and Experience 22*, 16 (2010), 2247–2265. [pages 21 and 47]

[124] Turley, J. Cortex-A15 "Eagle" flies the coop. *Microprocessor Report* (Nov. 2010), 1–7. [pages 7, 68, and 74]

[125] van Dyk, D., Geveler, M., Mallach, S., Ribbrock, D., Göddeke, D., and Gutwenger, C. Honei: A collection of libraries for numerical computations targeting multiple processor architectures. *Computer Physics Communications 180*, 12 (2009), 2534–2543. [pages 21 and 47]

[126] Vasudevan, V., Andersen, D., Kaminsky, M., Tan, L., Franklin, J., and Moraru, I. Energy-efficient cluster computing with fawn: Workloads and implications. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking* (2010), ACM, pp. 195–204. [page 12]

[127] Vázquez, M., Arís, R., Aguado-Sierra, J., Houzeaux, G., Santiago, A., López, M., Córdoba, P., Rivero, M., and Cajas, J. C. *Selected Topics of Computational and Experimental Fluid Mechanics*. Springer International Publishing, Cham, 2015, ch. Alya Red CCM: HPC-Based Cardiac Computational Modelling, pp. 189–207. [pages 21 and 90]

[128] Vazquez, M., Houzeaux, G., Koric, S., Artigues, A., Aguado-Sierra, J., Aris, R., Mira, D., Calmet, H., Cucchietti, F., Owen, H., et al. Alya: towards exascale for engineering simulation codes. *arXiv preprint arXiv:1404.4881* (2014). [pages 21 and 90]

[129] Warren, M. S., Weigle, E. H., and Feng, W.-C. High-density computing: A 240-processor beowulf in one cubic meter. In *Supercomputing, ACM/IEEE 2002 Conference* (Nov 2002), pp. 61–61. [page 12]

[130] Weicker, R. Dhrystone: a synthetic systems programming benchmark. *Communications of the ACM 27*, 10 (1984), 1013–1030. [page 32]

[131] Whaley, R. C., and Dongarra, J. Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing* (1998). [pages 31 and 39]

[132] Winkel, M., Speck, R., Hübner, H., Arnold, L., Krause, R., and Gibbon, P. A massively parallel, multi-disciplinary barnes–hut tree code for extreme-scale n-body simulations. *Computer Physics Communications 183*, 4 (2012), 880 – 889. [pages 21 and 41]

[133] Yoo, A. B., Jette, M. A., and Grondona, M. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003. Revised Paper* (2003), Springer, pp. 44–60. [page 40]