Experiences With Mobile Processors for Energy Efficient HPC

Nikola Rajovic[*†], Alejandro Rico[*†], James Vipond[*], Isaac Gelado[*], Nikola Puzovic[*], Alex Ramirez[*†]
[*]Barcelona Supercomputing Center
[†]Universitat Politècnica de Catalunya, BarcelonaTech
{first.last}@bsc.es

Abstract—**The performance of High Performance Computing (HPC) systems is already limited by their power consumption. The majority of top HPC systems today are built from commodity server components that were designed for maximizing the compute performance. The Mont-Blanc project aims at using low-power parts from the mobile domain for HPC.**

In this paper we present our first experiences with the use of mobile processors and accelerators for the HPC domain based on the research that was performed in the project. We show initial evaluation of NVIDIA Tegra 2 and Tegra 3 mobile SoCs and the NVIDIA Quadro 1000M GPU with a set of HPC micro-benchmarks to evaluate their potential for energy-efficient HPC.

I. Introduction

The performance of supercomputers has shown an exponential growth in the past years: according to the Top500 list of fastest supercomputers, the performance has increased 10x every 3.6 years [1]. According to the Green500 list [2], which ranks supercomputers according to their energy efficiency, today's most energy efficient supercomputer – Beacon (Intel Xeon E5-2670 and Intel Xeon Phi 5110P) – achieves an energy efficiency of ∼2.5 GFLOPS/W[1]. Following this performance trend, exascale performance could be achieved in 2018, but without improvements in energy efficiency, the power requirement of such a machine would be 500 MW. According to a DARPA study on the future of exascale systems, a realistic budget for such a system will be in the order of 20 MW[2] [3] which requires an energy efficiency of 50 GFLOPS/W.

Current fastest supercomputers are based either on multicore processors or on heterogeneous architectures and are at least one order of magnitude away from the required energy efficiency. The Mont-Blanc project [4] is a European initiative to explore the use of low-power parts from the mobile and embedded market for HPC. The aim of the project is to lower the total power of exascale systems by using parts with a much higher GFLOPS/W ratio than the current general purpose processors, but at a cost of lower peak performance per core. This paper presents early performance and energy efficiency results from executing HPC-specific micro-kernels on three mobile platforms: two embedded general purpose SoCs and a discrete mobile GPU in the form of a compute accelerator.

[1]Data from Green 500 list published in November 2012: http://www.green500.org/lists/green201211

[2]This is seen as a reasonable requirement to avoid having to set the supercomputing facility next to a nuclear plant

978-3-9815370-0-0/DATE13/©2013 EDAA

TABLE I
Evaluated Platforms

Platform	CPU	Memory
Tegra 2	dual-core Cortex-A9 @ 1 GHz	1 GB DDR2
Tegra 3	quad-core Cortex-A9 @ 1.3 GHz	2 GB DDR3
Quadro 1000M	96 CUDA cores @ 700 MHz	2 GB DDR3

II. Mobile Platforms

In this section we describe the mobile platforms that we use to evaluate mobile processors for HPC, both in terms of performance and energy efficiency. Table I shows the three mobile platforms under evaluation.

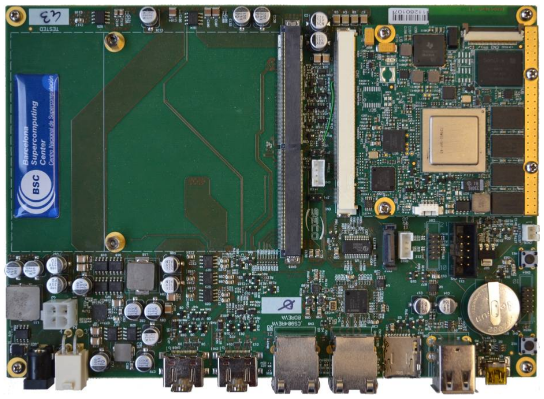
A. Tegra 2 platform

The first mobile SoC under evaluation is NVIDIA Tegra 2 [5]. It integrates a dual-core ARM Cortex-A9 [6] running at 1 GHz with 32 KB of private L1 cache and 1 MB of shared L2 cache. Alongside the CPU cores, Tegra 2 features a number of application-specific accelerators targeted at the mobile market such as GPU, video and audio encoder/decoder and image signal processor. None of these accelerators are programmable by means of CUDA, OpenCL or similar programming models which makes them unusable in HPC scenarios.

The Tegra 2 platform (See Figure 1(a)) consists of a Q7-compliant motherboard [7] and a Q7 compute card [8]. The compute card integrates an NVIDIA Tegra 2 SoC, 1 GB of DDR2-667 memory, 16 GB of eMMC storage, a 100 MbE NIC (interfaced through USB) and exposes PCIe connectivity to the motherboard. The motherboard hosts the compute card and also integrates a 1 GbE NIC (interfaced through the PCIe to the Tegra 2), μSD card adapter and exposes other connectors with related circuitry that are targeted to mobile/embedded hardware and software development (RS-232, HDMI, USB, SATA, etc.).

B. Tegra 3 platform - CARMA kit

The second system is the CARMA kit [9], which provides two platform configurations - homogeneous and heterogeneous. The first is quad-core ARM Cortex-A9 processor cluster and the second couples this cluster with a mobile discrete GPU for compute acceleration. The CARMA board has the same layout as the Tegra 2 platform (See Figure 1(b)).


(a) Tegra 2 developer board

(b) CARMA kit

Fig. 1. Platforms under evaluation.

The difference is that it contains a more powerful NVIDIA Tegra 3 SoC which (quad-core ARM Cortex-A9 running on to 1.3 GHz) [10], has 2 GB of memory and a single 1 GbE NIC (connected through USB to Tegra 3). It uses four PCIe v1 lanes to connect to a discrete mobile GPU. The GPU in this configuration is the NVIDIA Quadro 1000M, which is a mobile GPU (more precisely entry-level laptop GPU). This GPU is not well suited for applications that use double-precision floating point since the ratio between single-precision and double-precision performance is 1:8, meaning that double-precision performance is 8 times lower than the single-precision performance[3]. Tegra 3 itself brings some improvements in on-chip 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 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.

III. Benchmarks

During the lifetime of the Mont-Blanc project, we will test multiple architectures, some of which include compute accelerators, and in such cases the effort of porting real world production-level applications with thousands of lines of code is unaffordable. Hence, in order to evaluate these mobile platforms we use a number of micro-architectural benchmarks that stress different architectural features and cover a wide range of algorithms employed in HPC applications to provide a fundamental understanding of the strengths and weaknesses of mobile processors for HPC. The set includes 11 micro-benchmarks and here we list them 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 in an element-by-element basis. This workload mimics the vector operations often found in numerical simulations, and other compute intensive regular codes.

[3]NVIDIA GPUs that are commonly used for HPC have 1:2 ratio

Dense matrix-matrix Multiplication (dmmm): This code takes two dense matrices and produces an output dense matrix that is the result of a multiplying the two input matrices. Matrix multiplication is a common computation in many numerical simulations and measures the ability of the compute accelerator to exploit data reuse and compute 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 on each dimension, i.e., points with the same coordinates as the input point plus/minus an offset on only one dimension. This code evaluates the performance of strided memory accesses on the compute 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 an input matrix and produces an output matrix of the same size. Each point in the output matrix is calculated as a linear combination of the point with the same coordinates in the input matrix and the neighboring points. 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 compute 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 compute 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 compute 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. This code requires synchronizing execution threads after each merge step and serves as a good hint about the performance of barrier instructions on the compute 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 each body. This code is used to characterize the performance of irregular memory accesses on the compute 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 displacments 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 compute 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 multiplying the input matrix and vector together. This code assigns a different workload to each execution thread, and serves as a measurement of the performance of the compute accelerator when load imbalance occurs.

All micro-benchmarks are developed in three versions. The serial version is used in single-core scenarios to test the performance of a single core. The parallel version that we use for results reported here is developed using the OmpSs [11] programming model. OmpSs is a flexible programming model that supports asynchronous task-based parallelism. The version for testing the performance of GPUs is developed using NVIDIA's CUDA [12] programming model.

IV. Results

To get a comprehensive overview of the platforms we measure both the performance and power consumption of our micro-benchmarks, and we execute them in both serial version (See Figure 2) and multithreaded version (See Figure 3).

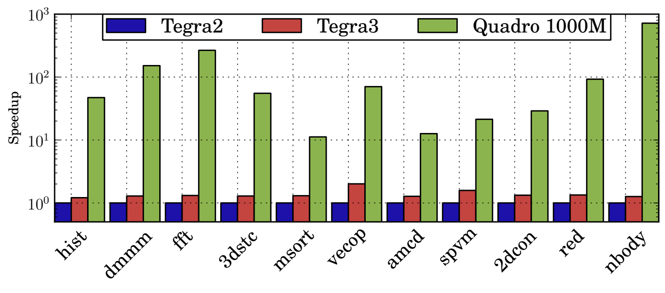
Although the core microarchitecture is the same in both Tegra 2 and Tegra 3, the higher operating frequency of the latter is reflected in the resulting performance: when utilizing only one core on both platforms, the execution time on Tegra 3 is 30% shorter on average for single precision (See Figure 2(a)). For double-precision codes the difference is slightly larger in favour of Tegra 3 (See Figure 2(b)). This is due to the higher memory bandwidth requirements of double-precision codes; these benchmarks operate on double the size of data compared to their single-precision counterparts, so they benefit from the higher memory bandwidth in Tegra 3 (DDR3 versus DDR2 in Tegra 2). This effect is also visible in those benchmarks that are memory bound, such as *vecop*, where the performance on Tegra 3 improves beyond the clock speed difference.

Figure 3 shows the performance and energy-to-solution results for all micro-benchmarks using all the available CPU cores: two on Tegra 2 and four on Tegra 3. On average, Tegra 3 completes execution two times 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 only accounts 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. The result shows that energy efficiency benefits from increasing the multicore density as long as applications scale reasonably well. Only *3D stencil* offers 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 so 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.

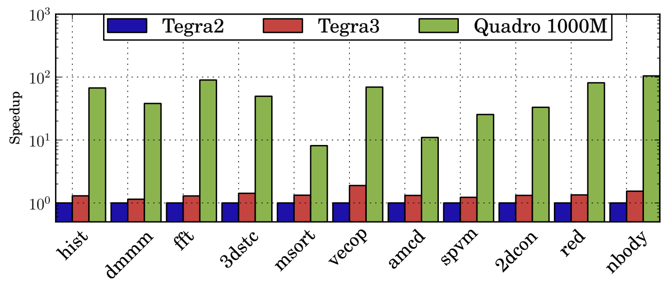
Both Figure 2 and Figure 3 also show the performance of the CUDA version of the micro-benchmarks running on the discrete GPU. These performance and energy-to-solution results are using all the processing cores in the GPU, as we do not have a way to restrict the execution to a subset of them. The GPU performance is, on average, 14x better than Tegra 2 using two cores and 7x better than Tegra 3 using four cores. Energy-to-solution is also better: 3x with respect to Tegra 2 and 2.5x with respect to Tegra 3. There are two micro-benchmarks for which the CUDA version is still under development and is not fully optimized yet: *merge-sort* and *atomic monte carlo dynamics*. In these cases, running the micro-benchmark on the GPU takes more energy than using the Tegra 3 cores only. This shows that off-loading tasks to the GPU only pays off if the speed-up is large enough to compensate for the increased platform power when running on the GPU compared to using just the CPU. Optimized versions of these two micro-benchmarks are expected to achieve higher energy efficiency numbers closer to those of their counterparts, which will confirm the benefits of the CPU+GPU combination for energy-efficient HPC also for mobile components.

V. Conclusions

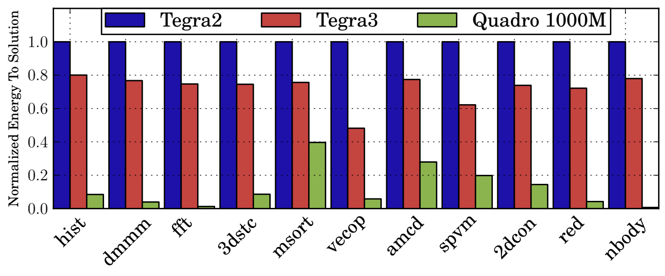
In this paper we presented initial results from our evaluation of mobile MPSoCs and their suitability for being used in the HPC domain. The evaluation with micro-benchmarks shows that more recent MPSoC (Tegra 3 vs Tegra 2) reduces the required energy to solution to 67% on average, using the same core architecture with increased multicore density.



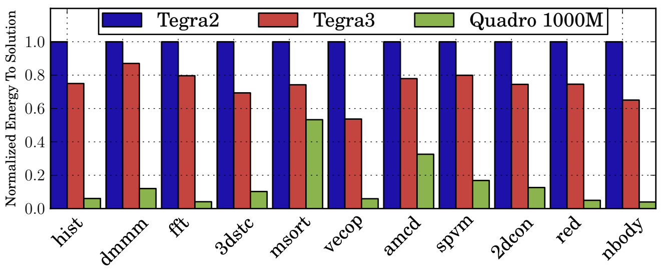
(a) Single-precision speedup.



(b) Double-precision speedup.

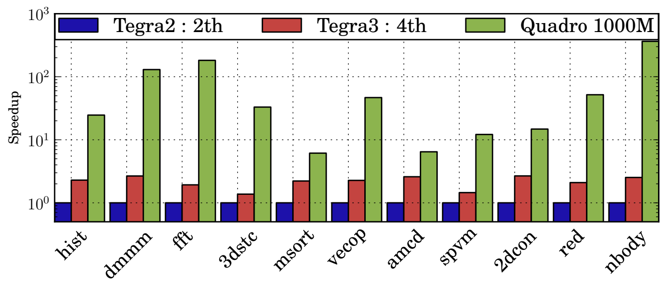


(c) Single-precision energy-to-solution.

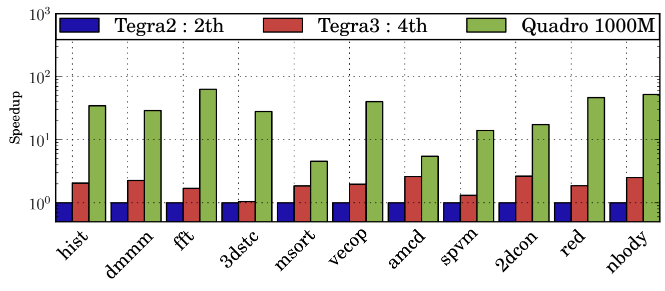


(d) Double-precision energy-to-solution.

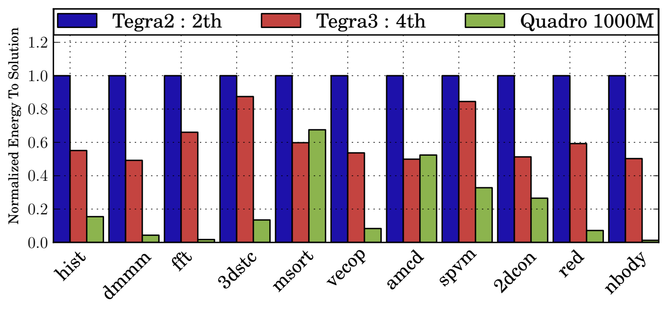
Fig. 2. Results for micro-benchmarks executed on a single core (normalized to Tegra 2 platform)



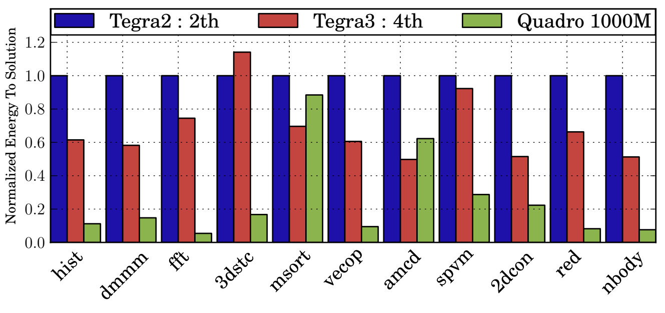
(a) Single-precision speedup.



(b) Double-precision speedup.



(c) Single-precision energy-to-solution.



(d) Double-precision energy-to-solution.

Fig. 3. Results for micro-benchmarks when executing multiple threads (normalized to Tegra 2 platform)

In our previous work, we compared an ARM Cortex-A9 (integrated in Tegra 2) to an Intel Core i7 640M processor [13], showing that a Cortex-A9 platform can be 1.2x more energy efficient on average at the cost of being 9x slower. In the light of the results shown here, we can see that the latest generation of Tegra MPSoCs (Tegra 3) could improve our previous results, and the gap in performance seen in our previous work is starting to close.

Recent interest in server-class low-power systems is pushing the evolution of ARM processors in a direction that is suitable for HPC as well: ARM is working on increasing the floating-point performance of its processors, and vendors are starting to implement better connectivity into their MPSoCs. The Cortex-A15 core [14], which is entering the market, already offers better performance than the Cortex-A9 core tested in this work. With future generations of ARM cores, such as the 64-

bit Cortex-A57, and with increased multicore density, ARM-based MPSoCs are expected to be more competitive in terms of performance while increasing energy efficiency.

Another way for increasing the usability of ARM-based mobile MPSoCs in the HPC domain is the use of integrated GPU: most ARM-based mobile MPSoCs have an integrated GPU, such as NVIDIA ULP GeForce in Tegra 2 and Tegra 3, ARM Mali [15] in Samsung Exynos 5 and PowerVR in TI's OMAP. If these GPUs are made programmable for general purpose applications, and not only for graphics, the overall energy efficiency of ARM-based MPSoCs should further increase.

Acknowledgments

This project and the research leading to these results is supported by Mont-Blanc project (European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n^o 288777) and PRACE project (European Community funding under grants RI-261557 and RI-283493).

References

[1] "TOP500: TOP 500 Supercomputer Sites," http://www.top500.org.

[2] "The Green 500 List," http://www.green500.org.

[3] K. Bergman *et al.*, "Exascale Computing Study: Technology Challenges in Achieving Exascale Systems," in *DARPA Technical Report*, 2008.

[4] "The MontBlanc project," http://montblanc-project.eu.

[5] NVIDIA, "The Benefits of Multiple CPU Cores in Mobile Devices," White Paper, 2010.

[6] ARM Ltd., "Cortex-A9 Processor," http://www.arm.com/products/processors/cortex-a/cortex-a9.php.

[7] SECO, "SECOCQ7-MXM," http://www.seco.com/en/item/secocq7-mxm.

[8] ——, "QuadMo747-X/T20 - Qseven® Rel.1.20 Compliant Module based on NVIDIA® Tegra® T20 Processor," http://www.seco.com/en/item/quadmo747-x_t20.

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

[10] ——, "Variable SMP (4-PLUS-1™) – A Multi-Core CPU Architecture for Low Power and High Performance," White Paper, 2011.

[11] A. Duran *et al.*, "OmpSs: A Proposal for Programming Heterogeneous Multi-Core Architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011.

[12] NVIDIA, "Compute Unified Device Architecture Programming Guide," 2007.

[13] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, and A. Ramirez, "The Low Power Architecture Approach Towards Exascale Computing," *Journal of Computational Science*, To Appear.

[14] J. Turley, "Cortex-A15 "Eagle" flies the coop," *Microprocessor Report*, vol. 24, no. 11, pp. 1–11, Nov. 2010.

[15] ARM Ltd., "Mali Graphics," http://www.arm.com/products/multimedia/mali-graphics-hardware/index.php.