Position Paper: OpenMP scheduling on ARM big.LITTLE architecture

Anastasiia Butko, Louisa Bessad, David Novo,
Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli,
Lionel Torres, and Michel Robert

LIRMM (CNRS and University of Montpellier), Montpellier, France
firstname.lastname@lirmm.fr

Abstract. Single-ISA heterogeneous multicore systems are emerging as a promising direction to achieve a more suitable balance between performance and energy consumption. However, a proper utilization of these architectures is essential to reach the energy benefits. In this paper, we demonstrate the ineffectiveness of popular OpenMP scheduling policies executing Rodinia benchmark on the Exynos 5 Octa (5422) SoC, which integrates the ARM big.LITTLE architecture.

1 Introduction

Traditional CPUs consume just too much power and new solutions are needed to scale up to the ever-growing demand on computational complexity. Accordingly, major efforts are focusing on achieving a more holistic balance between performance and energy consumption. In this context, heterogeneous multicore architectures are firmly established as the main gateway to higher energy efficiency. Particularly interesting is the concept of single-ISA heterogeneous multicore systems [1], which is an attempt to include heterogeneity at the microarchitectural level while preserving a common abstraction to the software stack. In single-ISA heterogeneous multicore systems, all cores execute the same machine code and thus, any core can execute any part of the code. Such model makes it possible to execute the same OS kernel binary implemented for symmetric *Chip Multi-Processors (CMPs)* with only minimal configuration changes.

In order to take advantage of single-ISA heterogeneous multicore architectures, we need an appropriate strategy to manage the distribution of computation tasks—also known as efficient thread scheduling in multithreading programming models. OpenMP [2] is a popular programming model that provides a shared memory parallel programming interface. It features a thread-based fork-join task allocation model and various loop scheduling policies to determine the way in which iterations of a parallel loop are assigned to threads.

This paper measures the impact of different loop scheduling policies in a real state-of-the-art single-ISA heterogeneous multicore system. We use the Exynos 5 Octa (5422) System-on-Chip (SoC) [3] integrating the ARM big.LITTLE architecture [4], which couples relatively slower, low-power processor cores (LITTLE)

with relatively more powerful and power-hungry ones (big). We provide insightful performance and energy consumption results on the Rodinia OpenMP benchmark suite [5] and demonstrate the ineffectiveness of typical loop scheduling policies in the context of single-ISA heterogeneous multicore architectures.

2 The Exynos 5 Octa (5422) SoC

2.1 Platform Description

We run our experiments on the Odroid-XU3 board, which contains the Exynos 5 Octa (5422) SoC with the ARM big.LITTLE architecture. ARM big.LITTLE technology features two sets of cores: a low performance energy-efficient cluster that is called "LITTLE" and power hungry high performance cluster that is called "big". The Exynos 5 Octa (5422) SoC architecture and its main parameters are presented in Figure 1. It contains: (1) a cluster of four out-of-order superscalar Cortex-A15 cores with 32kB private caches and 2MB L2 cache, and (2) a cluster of four in-order Cortex-A7 cores with 32kB private caches and 512KB L2 cache. Each cluster operates at independent frequencies, ranging from 200MHz up to 1.4GHz for the LITTLE and up to 2GHz for the big. The SoC contains 2GB LPDDR3 RAM, which runs at 933MHz frequency and with 2x32 bit bus achieves 14.9GB/s memory bandwidth. The L2 caches are connected to the main memory via the 64-bit Cache Coherent Interconnect (CCI) 400 [6].

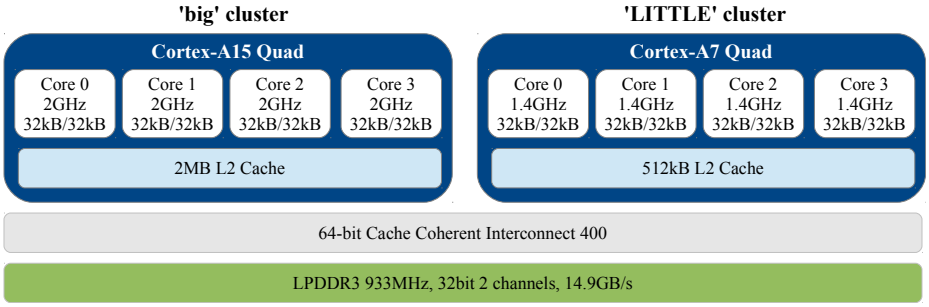


Fig. 1: Exynos 5 Octa (5422) SoC.

2.2 Software Support

Execution models. ARM big.LITTLE processors have three main software execution models [4]. The first and simplest model is called *cluster migration*. A single cluster is active at a time, and migration is triggered on a given workload threshold. The second mode named *CPU migration* relies on pairing every "big" core with a "LITTLE" core. Each pair of cores acts as a virtual core in which

only one actual core among the combined two is powered up and running at a time. Only four physical cores at most are active. The main difference between cluster migration and CPU migration models is that the four actual cores running at a time are identical in the former while they can be different in the latter. The *heterogeneous multiprocessing (HMP)* mode—also known as *Global Task Scheduling (GTS)*–allows using all of the cores simultaneously. Clearly, HMP provides the highest flexibility and consequently it is the promising mode to achieve the best performance/energy trade-offs.

Benchmarks. We consider the Rodinia benchmark suite for heterogeneous computing [5]. It is composed of applications and kernels of different nature in terms of workload, from domains such as bioinformatics, image processing, data mining, medical imaging and physics simulation. It also includes classical algorithms like LU decomposition and graph traversal. In our experiments, the OpenMP implementations are configured with 4 or 8 threads, depending on the number of cores that are visible to the thread scheduling algorithm. Due to space constraints, we selected the following subset of benchmarks: *backprop*, *bfs*, *heartwall*, *hotspot*, *kmeans openmp/serial*, *lud*, *nn*, *nw* and *srad v1/v2*.

Thread scheduling algorithms. OpenMP provides three loop scheduling algorithms, which allows determining the way in which iterations of a parallel loop are assigned to threads. The *static* scheduling is the default loop scheduling algorithm, which divides the loop into equal or almost equal chunks. This scheduling provides the lowest overhead, but, as we will show in the results, the potential load imbalance can cause significant synchronization overheads. The *dynamic* scheduling assignings chunks at runtime once threads complete previously assigned iterations. An internal work queue of chunk-sized blocks is used. By default, the chunk size is '1' and this can be explicitly specified by a programmer at compile time. Finally, the *guided* scheduling is similar to dynamic scheduling, but the chunk size exponentially decreases from the value calculated as $\#interations/\#threads$ to '1' by default or to a value explicitly specified by a programmer at compile time.

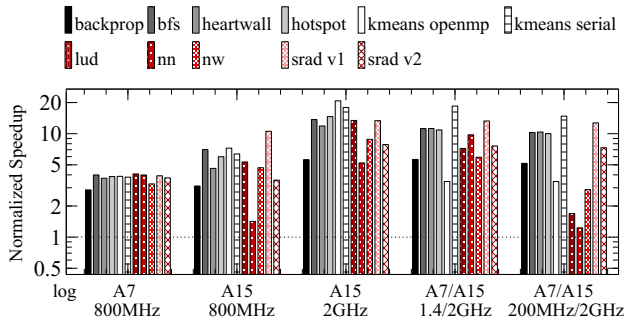
In the next section, we consider these three loop scheduling policies with the default chunk size. Furthermore, the experiments are run with the following software system configuration: the Ubuntu 14.04 Linux kernel LTS 3.10, the GCC 4.8.2 compiler and the OpenMPI 3.1 libraries.

3 Experimental Results

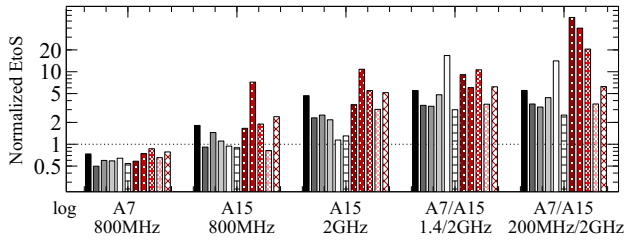
In this section we present a detailed analysis of the OpenMP implementation of the Rodinia benchmark suite running on the ARM big.LITTLE architecture. We consider the following configurations:

- Cortex-A7 cluster running at 200 MHz, 800 MHz and 1.4 GHz;
- Cortex-A15 cluster running at 200 MHz, 800 MHz and 2GHz;
- Cortex-A7/A15 clusters running at 200/200 MHz, 800/800 MHz, 1.4/2 GHz, 200 MHz/2 GHz and 1.4 GHz/200 MHz.

Static Thread Scheduling. Figure 2(a) shows in logarithmic scale the measured execution time of different configurations using the static scheduling algorithm. The results are normalized with respect to the slowest configuration, i.e., Cortex-A7 running at 200MHz. As expected, the highest performance is typically achieved by the Cortex-A15 running at 2GHz. For example, a speedup of 21x is observed when running the *kmeans openmp* in the big cluster. When using the HMP mode to simultaneously run on the big and LITTLE clusters (i.e., A7/A15 in the figure), the execution time is usually slower to that of the big cluster alone, despite using four additional actives cores. An even higher penalty is observed when operating the LITTLE cluster at a lower frequency, especially so for the *lud*, *nn* and *nw* applications.



(a) Execution time speedup comparison



(b) EtoS comparison

Fig. 2: Normalized speedup using Static scheduling (reference A7 at 200MHz).

Figure 2(b) shows the normalized *Energy to Solution (EtoS)* measured with the on-board power monitors present in the Odroid-XU3 board. Results are again normalized against the reference Cortex-A7 running at 200MHz. We observe, that the Cortex-A7 cluster is generally more energy-efficient than the Cortex-A15. Furthermore, the best energy efficiency is achieved when operating at 800MHz. Besides, we also observe that for a few applications (i.e., *bfs*, *kmeans serial*, and *srad v1*) the Cortex-A15 running at 800MHz provides slightly better EtoS than the reference Cortex-A7 cluster. These applications benefit the most from the A15 out-of-order architecture achieving the largest speedups. This leads

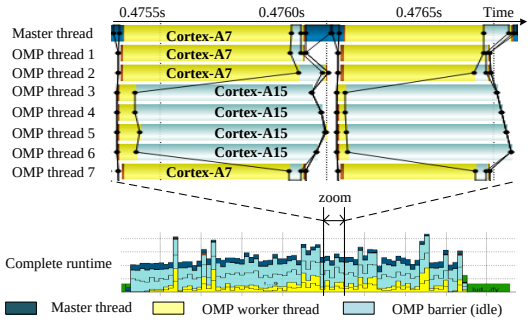


Fig. 3: *lud* on HMP big.LITTLE at 200MHz/2GHz.

to a higher energy efficiency despite running on a core of higher power consumption. When using the HMP mode, some application exhibit a very high EtoS. Particularly high are the EtoS of the *lud* and *nn* applications executed in the configuration Cortex-A7/A15 running at 200MHz/2GHz. Our experiments also show that HMP is less energy efficient than the big cluster running at maximum frequency (i.e., A15 2GHz). In conclusion, static thread scheduling achieves a highly suboptimal use of our heterogeneous architecture, which turns out to be slower and less energy efficient than a single big cluster.

Further investigations were carried out with Scalasca [7] and Vampir [8] software tools that permit instrumenting the code and visualizing low-level behavior based on collected execution traces. Figure 3 shows a snapshot of the execution trace of the *lud* application alongside a zoom on two consecutive *parallel-for* loop constructs. It is clearly visible that the OpenMP runtime spawned eight threads, which got assigned to the eight cores. The four threads assigned to the Cortex-A15 cores completed execution of their chunks significantly faster than the Cortex-A7 cores. As a result, the execution critical path is affected by the slowest cores, which slows down system performance.

Dynamic and Guided Thread Scheduling. Figures 4(a-b) respectively illustrate the execution time using dynamic and guided thread scheduling normalized by the static scheduling discussed previously. The dynamic scheduling is able to achieve good speedups for some applications (e.g., *nn*) but also degrades the performance of some others (e.g., *nw*). Something very similar happens with the guided scheduling but with different application/configuration sets. For example, the *heartwall* is now degraded for the 1.4GHz/200MHz configuration while the *nn* achieves a 1.8x speedup.

Figures 4(c-d) respectively show the EtoS of the dynamic and guided scheduling normalized by the static scheduling. We observe a very high correlation with respect to the corresponding execution time graphs. Accordingly, we can conclude that there is no existing policy that is generally superior. The best policy will depend on the application and on the architecture configuration. However, we believe that none of the policies is able to fully leverage the heterogeneity of our architecture and that more intelligent thread scheduling policies are needed

to sustain the energy efficiency promised by single-ISA heterogeneous multicore systems.

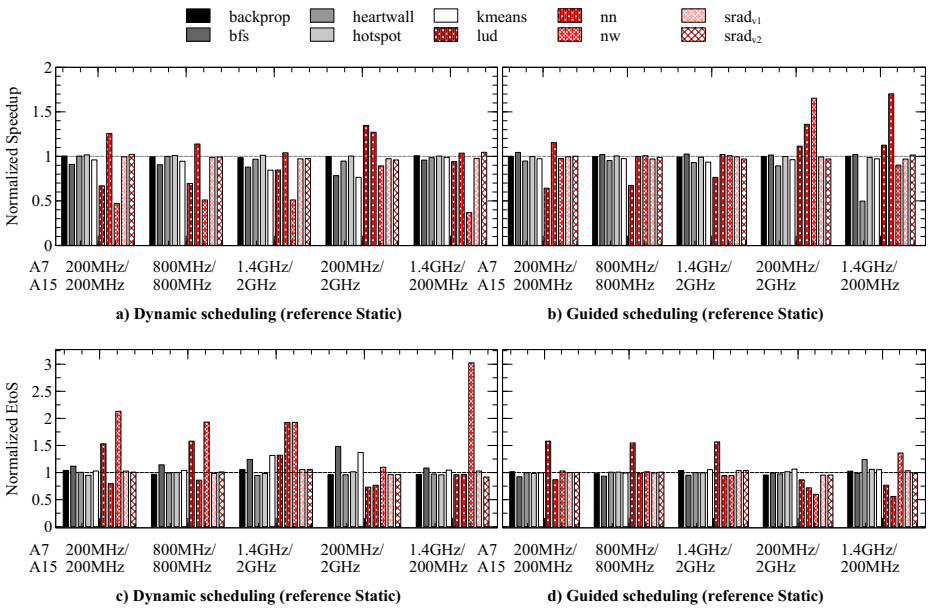


Fig. 4: Normalized execution time speedup and EtoS.

4 Conclusion

In this paper, we evaluate performance and energy trade-offs of single-ISA heterogeneous multicore system. The investigations were conducted on the Odroid XU3 board including an ARM big.LITTLE Exynos 5 Octa (5422) chip. We provided performance and energy results on the Rodinia OpenMP benchmark suit using typical loop scheduling policies, i.e. static, dynamic and guided. The results show that the given policies are inefficient in the use of heterogeneous cores.

Therefore, we conclude that further research is required to propose suitable scheduling policies able to leverage the superior energy efficiency of LITTLE cores while maintaining the faster execution times of big cores.

5 Acknowledgement

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2016) under the Mont-Blanc 2 Project: http://www.montblanc-project.eu, grant agreement n^{o} 610402.

References

1. R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ISCA '04, (Washington, DC, USA), pp. 64–, IEEE Computer Society, 2004.
2. O. A. R. Board, "The openmp api specification for parallel programming." http://openmp.org/wp/, November 2015.
3. Samsung, "Exynos Octa SoC." https://http://www.samsung.com/, November 2015.
4. B. Jeff, "big.little technology moves towards fully heterogeneous global task scheduling." http://www.arm.com/files/pdf/, November 2013.
5. S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 44–54, Oct 2009.
6. ARM, *CoreLink CCI-400 Cache Coherent Interconnect Technical Reference Manual*, November 16 2012. Revision r1p1.
7. "Scalasca." http://www.scalasca.org/, November 2015.
8. "Vampir - performance optimization." https://www.vampir.eu/, November 2015.