POSTER: Exploiting Asymmetric Multi-Core Processors with Flexible System Sofware

Kallia Chronaki[*‡], Miquel Moretó[*‡], Marc Casas[*], Alejandro Rico[+],
Rosa M. Badia[*†], Eduard Ayguadé[*‡],Jesus Labarta[*‡] and Mateo Valero[*]
[*]Barcelona Supercomputing Center, Barcelona, Spain
[‡]Universitat Politècnica de Catalunya - BarcelonaTech, Barcelona, Spain [+]ARM, Austin, Texas
[†]Research Institute (IIIA) - Spanish National Research Council (CSIC), Barcelona, Spain

ABSTRACT

Energy efficiency has become the main challenge for high performance computing (HPC). The use of mobile asymmetric multi-core architectures to build future multi-core systems is an approach towards energy savings while keeping high performance. However, it is not known yet whether such systems are ready to handle parallel applications.

This paper fills this gap by evaluating emerging parallel applications on an asymmetric multi-core. We make use of the PARSEC benchmark suite and a processor that implements the ARM big.LITTLE architecture. We conclude that these applications are not mature enough to run on such systems, as they suffer from load imbalance.

Furthermore, we explore the behaviour of dynamic scheduling solutions on either the Operating System (OS) or the runtime level. Comparing these approaches shows us that the most efficient scheduling takes place in the runtime level, influencing the future research towards such solutions.

1 Introduction

Energy efficiency has become the main challenge for future parallel computing designs [14], motivating prolific research to face the *Power Wall*. An interesting approach towards energy efficiency is the use of asymmetric multi-core architectures [2, 17] with different types of cores targeting different performance and power optimization points. Such systems have been successfully deployed in the mobile domain, where simple in-order cores (*little*) have been combined with aggressive out-of-order cores (*big*) to build these systems.

Many researchers are pushing towards building future parallel systems with asymmetric multi-cores [9, 10, 12, 13, 23] and even mobile chips [20]. However, it is unclear if current parallel applications will benefit from these asymmetric platforms. Load balancing and scheduling are two of the main challenges in utilizing such heterogeneous platforms, as the programmer has to consider them from the very beginning to obtain an efficient parallelization.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

In this work we evaluate the suitability of modern asymmetric multi-core platforms for highly parallel applications. First, we demonstrate that out-of-the-box parallel applications do not run efficiently on asymmetric multi-cores as the asymmetry of the system leads to load imbalance.

When load-balancing techniques are not included in the original application, we evaluate alternative solutions that, without relying on the programmer, can leverage the opportunities that asymmetric systems offer. These solutions consist of dynamic schedulers on either the OS or on the runtime system level. We use a state of the art dynamic OS scheduler that is aware of the asymmetry of the platform. Another approach is the use of a runtime system that is responsible to schedule the workload to the appropriate idle cores dynamically. This is done with the use of a modern task-based programming model that allows the specification of inter-task dependences and lets the runtime system to track dependences between tasks. We compare and analyse the outcome of this evaluation in terms of performance, power and energy.

Although there has been remarkable research on asymmetric systems, we consider that their experimental evaluation is limited compared to our work. First, there are many works that base their evaluation on a simulated or emulated environment [1, 2, 11–13, 15–17, 19, 21–23, 25, 26], in contrast to our real asymmetric system. Furthermore, many works use either random task dependency graph generators or scientific kernels instead of real scientific applications [5, 7, 22, 24]. Finally, many of the existing works do not present power and energy results [5, 11, 17, 22, 25, 26].

2 The ARM big.LITTLE Architecture

The ARM big.LITTLE [6,10] is a modern asymmetric multi-core architecture that has been successfully deployed in the mobile market. In this work, we make use of one of the commercially available development boards featuring a big.LITTLE architecture: the Hardkernel Odroid-XU3 development board. The Odroid-XU3 includes an 8-core Samsung Exynos 5422 chip with four ARM Cortex-A15 (big) cores and four Cortex-A7 (little) cores. For the remainder of the paper, we refer to Cortex-A15 cores as *big* and to Cortex-A7 cores as *little*.

Scheduling a set of processes on an asymmetric multi-core system is more challenging than the traditional process scheduling on symmetric multi-cores. An efficient OS scheduler has to take into account the different characteristics of the core types of the system. The ARM big.LITTLE systems provide three mainstream OS schedulers: *cluster*

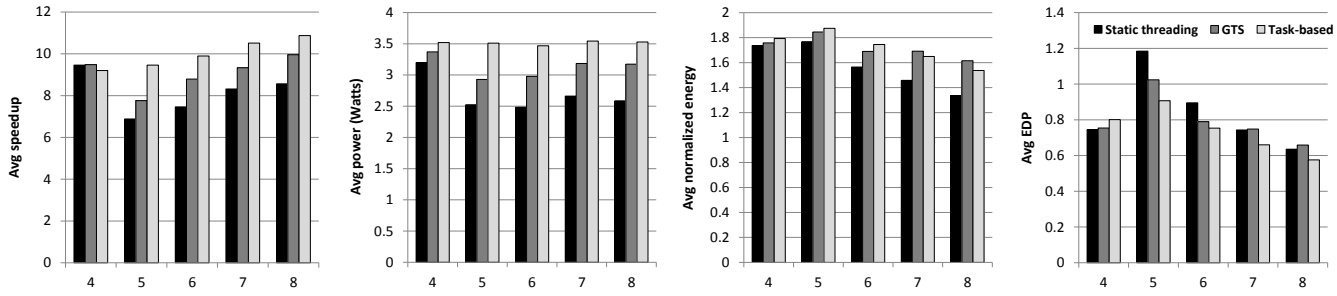

Figure 1: Average results when running on 4 to 8 cores with 4 of them big. Speedup is over 1 little core, static threading on 4 little cores is the baseline of energy consumption and EDP

switching [6], *in-kernel switch* [18] and *global task scheduling* (GTS) [6]. GTS allows running applications on all cores in the asymmetric multi-core and is considered as the most sophisticated scheduler. In GTS, all cores are available and visible to the OS scheduler, and this scheduler is aware of the characteristics and each core type. GTS tracks the CPU utilization of each process and dynamically schedules high CPU utilization processes to big cores and low CPU utilization processes to little cores. As a result, cores are active according to the characteristics of the running processes.

3 Evaluation

The experiments of this work are performed on the Hardkernel Odroid XU3 described in Section 2. In these experiments, we set the big cores to run at 1.6GHz and the little cores at 800MHz through the `cpufreq` driver.

We measure the performance, power and energy of the original PARSEC codes [3] together with a task-based implementation of nine benchmarks[1] of the suite [4]. For space purposes we only show the average results among these benchmarks. The original codes make use of the `pthreads` library for all the selected benchmarks, while the task-based implementation is done using the OmpSs programming model [8]. The OmpSs applications follow the same parallelization strategy implemented with OpenMP 4.0 task annotations.

In the search of the optimal solution, we compare three different scenarios of parallel execution when transferring the scheduling responsibility to different levels of the software stack: *(a) Static Threading:* the scheduling responsibility is on the application level. *(b) OS Scheduling (GTS):* the OS is responsible for performing the scheduling. Specifically we use the GTS provided by ARM described in Section 2. *(c) Task-based:* the runtime system is responsible for the dynamic scheduling.

Figure 1 shows the average results among the evaluated benchmarks. We refer to the system configurations as B+L where B is the number of big cores and L is the number of little cores. The speedup chart of Figure 1 shows that the *Static threading* approach does not benefit from adding little cores to the system. In fact, this approach brings an average 15% slowdown when adding four little cores (configuration `4+4`). This is a result of the static thread scheduling; the same amount of work is assigned to each core, so when the big cores finish the execution of their part, they become idle and under-utilized. GTS achieves a limited speedup of 5% with the addition of four little cores to the `4+0` configuration. The addition of a single little core brings a 22% slowdown (from `4+0` to `4+1`) and requires three additional little cores to reach the performance of the symmetric configuration (configuration `4+3`). Finally, the *Task-based* approach always benefits from the extra computational power as the runtime automatically deals with load imbalance. Performance improvements keep growing with the additional little cores, reaching an average improvement of 16% over the symmetric configuration when 4 extra cores are added.

The power chart of Figure 1 shows oppositional benefits among the three approaches. We can see that *Static threading* and *GTS* benefit from asymmetry, effectively reducing average power consumption. *Static threading* reduces power consumption when moving from the `4+0` to the `4+4` system by 23% while *GTS* does so by 6.2%. On the other hand, the *task-based* approach keeps the big cores busy for most of the time so it maintains the average power nearly constant. By keeping the power stable, the energy consumption of the task-based approach shown in the third chart of Figure 1 is minimized since it only depends on the execution time.

To see the impact on both performance and energy efficiency we plot the average energy delay product (EDP) on the rightmost chart of Figure 1. In this chart the lower values are the better. We observe that the *task-based* approach is the one that has the best performance-energy combination for the asymmetric configurations since it maintains the lowest EDP for all cases. *Static threading* manages to reduce the average EDP by 7% while *GTS* and *task based* approaches do so by 10% and 37% respectively.

4 Conclusions

In this paper we examine the maturity of asymmetric multicore systems to support emerging parallel applications, showing results for performance, power, energy and EDP. Through our comparison of the three scheduling approaches, we conclude that the task-based approach is the optimal solution to dynamically balance the load among the asymmetric resources. Contrarily to the static and OS scheduling approaches, the task-based approach constantly improves performance as asymmetry is increasing. Moreover, relying on the runtime system for the efficient scheduling keeps power static which results in energy savings. Finally, according to the EDP results, the conclusion is that the task-based approach offers the optimal performance and energy trade-off for asymmetric systems.

[1]The benchmarks used are: blackscholes, bodytrack, canneal, dedup, facesim, ferret, fluidanimate, streamcluster and swaptions.

5 References

[1] A. Agarwal and P. Kumar. Economical Duplication Based Task Scheduling for Heterogeneous and Homogeneous Computing Systems. In *IACC*, 2009.

[2] S. Balakrishnan, R. Rajwar, M. Upton, and K. K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *ISCA*, pages 506–517, 2005.

[3] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[4] D. Chasapis, M. Casas, M. Moreto, R. Vidal, E. Ayguade, J. Labarta, and M. Valero. PARSECSs: Evaluating the Impact of Task Parallelism in the PARSEC Benchmark Suite. *Trans. Archit. Code Optim.*, 2015.

[5] K. Chronaki, A. Rico, R. M. Badia, E. Ayguadé, J. Labarta, and M. Valero. Criticality-aware dynamic task scheduling for heterogeneous architectures. In *ICS*, pages 329–338, 2015.

[6] H. Chung, M. Kang, and H.-D. Cho. Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM big.LITTLE Technology. Technical report, Samsung Electronics Co., Ltd., 2013.

[7] M. Daoud and N. Kharma. Efficient Compile-Time Task Scheduling for Heterogeneous Distributed Computing Systems. In *ICPADS*, 2006.

[8] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas. Ompss: a Proposal for Programming Heterogeneous Multi-Core Architectures. *Parallel Processing Letters*, 21, 2011.

[9] A. Fedorova, J. C. Saez, D. Shelepov, and M. Prieto. Maximizing Power Efficiency with Asymmetric Multicore Systems. *Communications of the ACM*, 52(12), 2009.

[10] P. Greenhalgh. big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. *ARM White Paper*, 2011.

[11] M. A. Iverson, F. Özgüner, and G. J. Follen. Parallelizing Existing Applications in a Distributed Heterogeneous Environment. In *HCW*, 1995.

[12] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt. Bottleneck identification and scheduling in multithreaded applications. In *ASPLOS*, pages 223–234, 2012.

[13] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt. Utility-based acceleration of multithreaded applications on asymmetric CMPs. In *ISCA*, pages 154–165, 2013.

[14] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, and Others. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical report, University of Notre Dame, CSE Dept., 2008.

[15] D. Koufaty, D. Reddy, and S. Hahn. Bias scheduling in heterogeneous multi-core architectures. In *EuroSys*, pages 125–138, 2010.

[16] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO*, pages 81–92, 2003.

[17] 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 *ISCA*, pages 64–75, 2004.

[18] Mathieu Poirier. In Kernel Switcher: A solution to support ARM's new big.LITTLE technology. Embedded Linux Conference 2013, 2013.

[19] T. Y. Morad, U. C. Weiser, A. Kolodny, M. Valero, and E. Ayguade. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *IEEE Comput. Archit. Lett.*, 5(1):4–17, Jan. 2006.

[20] 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 *SC*, 2013.

[21] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu. Scalable thread scheduling in asymmetric multicores for power efficiency. In *SBAC-PAD*, pages 59–66, 2012.

[22] G. Sih and E. Lee. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2), 1993.

[23] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS*, pages 253–264, 2009.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3), 2002.

[25] K. Van Craeynest, S. Akram, W. Heirman, A. Jaleel, and L. Eeckhout. Fairness-aware Scheduling on single-ISA Heterogeneous Multi-cores. In *PACT*, 2013.

[26] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *ISCA*, pages 213–224, 2012.

Acknowledgments

This work has been supported by the Spanish Government (SEV2015-0493), by the Spanish Ministry of Science and Innovation (contracts TIN2015-65316-P), by Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272), by the RoMoL ERC Advanced Grant (GA 321253) and the European HiPEAC Network of Excellence. The Mont-Blanc project receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 610402 and from the EU's H2020 Framework Programme (H2020/2014-2020) under grant agreement number 671697. M. Moretó has been partially supported by the Ministry of Economy and Competitiveness under Juan de la Cierva postdoctoral fellowship number JCI-2012-15047. M. Casas is supported by the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the Cofund programme of the Marie Curie Actions of the 7th R&D Framework Programme of the European Union (Contract 2013 BP_B 00243).