Energy Efficiency of Software Transactional Memory in a Heterogeneous Architecture

Emilio Villegas, Alejandro Villegas, Angeles Navarro, Rafael Asenjo,
Yash Ukidave*, Oscar Plata

University of Malaga, Dept. Computer Architecture, Spain

*Advanced Micro Devices

e-mail: {evillegas,avillegas,magonzalez,asenjo,oplata}@uma.es, yash.ukidave@amd.com

Hardware vendors make an important effort creating low-power CPUs that keep battery duration and durability above acceptable levels. In order to achieve this goal and provide good performance-energy for a wide variety of applications, ARM designed the big.LITTLE architecture. This heterogeneous multi-core architecture features two different types of cores: big cores oriented to performance, fast and power-hungry, and little cores, slower and aimed to save energy consumption. As all the cores have access to the same memory, multi-threaded applications must resort to some mutual exclusion mechanism to coordinate the access to shared data by the concurrent threads. Transactional Memory (TM) represents an optimistic approach for shared-memory synchronization.

To take full advantage of the features offered by software TM, but also benefit from the characteristics of the heterogeneous big.LITTLE architectures, our focus is to propose TM solutions that take into account the power/performance requirements of the application and what it is offered by the architecture. In order to understand the current state-of-the-art and obtain useful information for future power-aware software TM solutions, we have performed an analysis of a popular TM library running on top of an ARM big.LITTLE processor. Experiments show, in general, better scalability for the LITTLE cores for most of the applications except for one, which requires the computing performance that the big cores offer.

1 Introduction

Mobile and embedded devices, whose use is widely extended at present, should be designed to be very energy-efficient. Most of them, and most of the time, depend on a battery for their operation. Energy-efficient components ensure a longer duration and durability of these batteries. Furthermore, the environment they are located in and the use we make of these devices require them to consume a low amount of energy and to keep their temperature as low as possible. As the market for these types of devices grows, industry focuses on designing energy-efficient low-power processors. Specifically, ARM has built an heterogeneous multi-core architecture named big.LITTLE [1]. This architecture features a set of *big* cores designed for performance and a set of *LITTLE* cores designed for energy-efficiency. We refer to these core sets as *big cluster* and *little cluster*, respectively. Both clusters share the same ISA and have access to the same main memory. Thus, an application can run in any of them. Applications with high performance requirements are intended to be attached to the big cluster, while those without these requirements are to be processed using the little cluster.

To take full advantage of these multi-core clusters, applications should be multi-threaded. As in any other multi-core processor, programmability challenges arise when writing a multi-threaded application, specially when coordination is needed when accessing shared data. Programmers should guarantee mutual exclusion for the portion of the code that is accessing shared data (i.e., critical section). Transactional Memory (TM) [8] has been designed as an optimistic model to ensure mutual exclusion, defining the concept of transaction to wrap a critical section. Many TM systems have been proposed in the last two decades for multi-core architectures [7], implemented either in hardware or software (or a combination). Modern processors and systems have started to include hardware TM support [14, 4].

Adapting both hardware and software TM solutions to a heterogeneous low-power architecture is an important commitment for the incoming years. Some efforts have been done in order to understand and propose energy-aware TM solutions [6, 11, 3, 9, 13]. However, these proposals do not take into consideration heterogeneous architectures. Furthermore, both hardware and software TM solutions are evaluated on top of simulation platforms in order to get a power consumption estimation. None of these works consider evaluating using real hardware for power measurements.

In this paper, we evaluate the TinySTM [5] software TM system running on the Odroid-XU3 platform [2], and present its performance and energy consumption using a set of benchmarks from the STAMP suite [10]. This energy/performance evaluation serves as a basis to choose among the big cluster or the little cluster for execution of the benchmark applications. The goal is to obtain enough information for future library/compiler/runtime optimization tools in order to improve the efficiency of software TM in heterogeneous low-power architectures.

Feature	Value
CPU	Samsung Exynos-5422: Cortex-A15 and Cortex-A7 big.LITTLE
Main Memory	2 Gbyte LPDDR3 RAM at 933MHz
GPU	Mali-T628 MP6
Storage	32GB Sandisk iNAND Extreme
Energy Monitor	Separated sensors to measure the power consumption of big cluster, little cluster, GPU and DRAM
OS	Linux odroid 3.10.59+

Table 1: ODROID-XU3 system setup used for evaluation

2 Energy/Performance Evaluation

2.1 Experimental setup

Our experimental evaluation was conducted in a ODROID-XU3 [2] computing device. This platform features a Samsung Exynos 5422 processor, based in the ARM big.LITTLE architecture, containing a quad-core Cortex-A15 and a quad-core Cortex-A7. Table 1 summarizes the hardware characteristics of this device. Hardware sensors are available for power measurements. Specifically, information can be obtained on energy consumption for the big and little clusters individually, the Mali GPU, and the memory subsystem. We have developed a software tool that samples power readings using the hardware sensors and integrate them through time using the real-time system clock. This way, energy consumption can be profiled through different applications stages.

Five applications from the STAMP benchmark suite were selected for evaluation: Intruder, Kmeans, Labyrinth, SCAA2, and Vacation. All of them used the ++ input parameters, as described in [10]. The high contention inputs were used for Kmeans and Vacation. Bayes was not considered as it shows unexpected behavior (this issue is also documented in [12]). In Genome, some synchronization errors were found when running multi-threaded experiments, and Yada execution usually resulted in out-of-memory errors. We are currently investigating these issues.

With respect to the TinySTM library, among the transaction management options available, the write-back policy and not commit-time locking (i.e., encounter-time locking) were selected.

By default, the STAMP codes have some instrumentation for performance evaluation. We replaced this instrumentation with calls to our own instrumentation library. This library provides access to the ODROID-XU3 energy counters: the big and little clusters, GPU, and main memory. Additionally, a time counter was included. Tests showed negligible differences with the original timing instrumentation. As we plan to measure the impact of using TinySTM on the complete chip, the power consumption of clusters, GPU, and memory was considered.

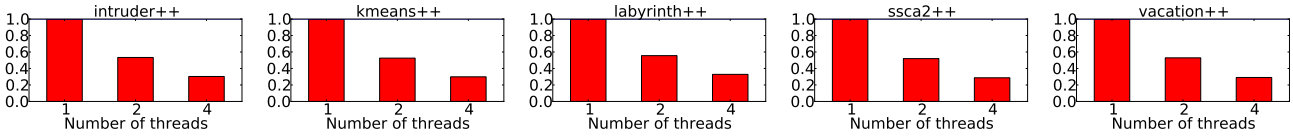
2.2 Experimental evaluation results

Two evaluation metrics were considered: the normalized execution time with respect to a single-threaded execution, and the energy consumption, again, normalized to the energy consumption of a single-threaded execution. These two metrics were combined to obtain the Energy-Delay Product (EDP). All the experiments were carried out comparing their results with respect to a single thread running TinySTM. For each application, 10 tests were run, taking as the result the average value. The experiments showed consistent results for every execution. A sequential version of the codes were not used as the intention was to provide scalability metrics for TinySTM instead of comparing TinySTM against other implementations.

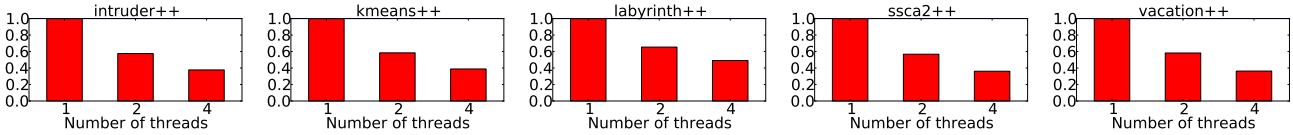
2.2.1 Little and big clusters evaluation

Fig. 1 shows the results of the evaluation of the little cluster. TinySTM achieves good scalability for the three parameters (execution time, energy consumption and EDP). The performance scalability of TinySTM together with the power efficiency of the Cortex-A7 processor permits to observe a reduction in EDP between 80% to 90% when using 4 threads.

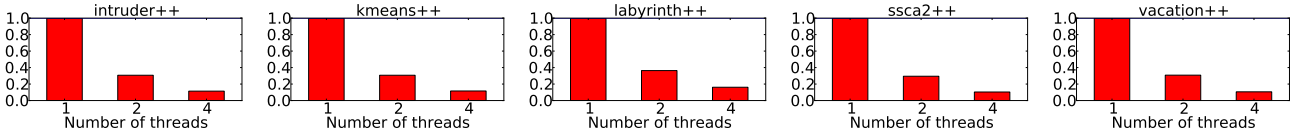
Fig. 2 shows the evaluation of the big cluster. The picture is different as compared to the little cluster. In this case, we observe scalable results for the execution time, but not as scalable as in the little cluster. In addition, as the Cortex-A15 processor is not as power efficient as the Cortex-A7, we observe that the energy consumption is higher when using 4 threads in Intruder, Kmeans, and Vacation. Despite TinySTM shows to be scalable in terms of execution time, this is not the case for energy when running in the big cluster. The scalability obtained in terms of execution time is not enough to compensate the energy increment when using 4 threads. As a result, the EDP is not always optimal for 4 threads.



a. Normalized execution time w.r.t single-thread running TinySTM

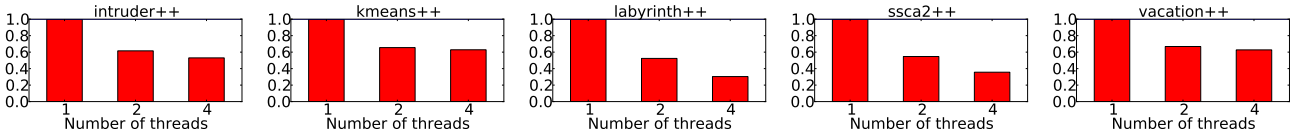


b. Normalized energy consumption w.r.t single-thread running TinySTM

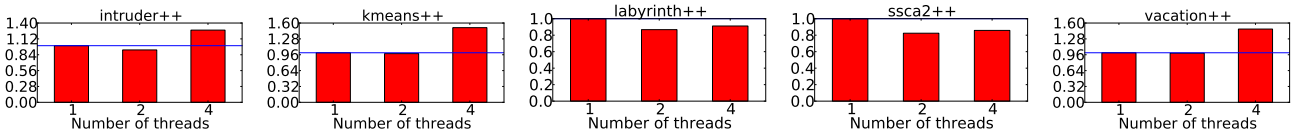


c. Normalized EDP w.r.t single-thread running TinySTM

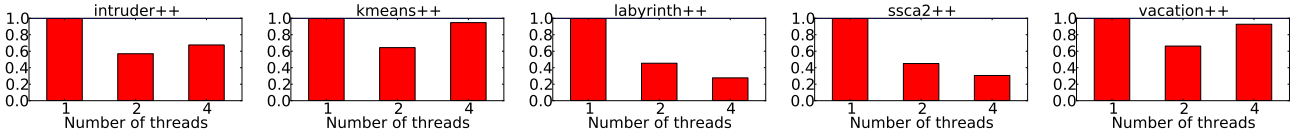
Figure 1: Little cluster evaluation



a. Normalized execution time w.r.t single-thread running TinySTM



b. Normalized energy consumption w.r.t single-thread running TinySTM



c. Normalized EDP w.r.t single-thread running TinySTM

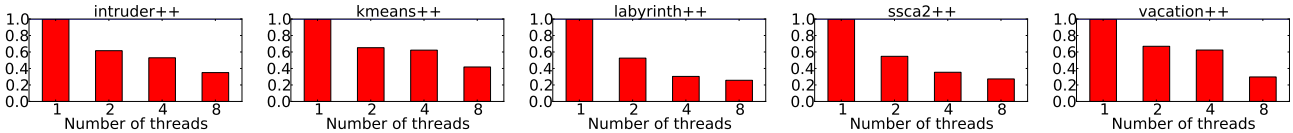
Figure 2: Big cluster evaluation

2.2.2 Full system evaluation

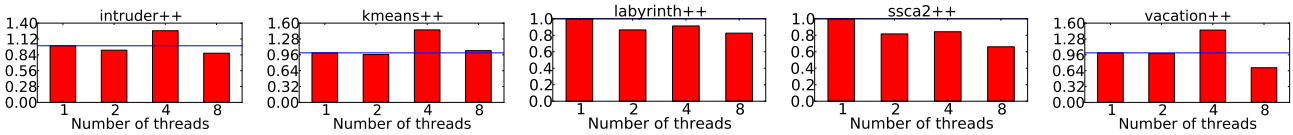
Fig. 3 shows the evaluation for the full system. Before running these experiments, we observed that the OS scheduler tries to use the big cluster as soon as it detects any overload. We decided to keep the default behavior of the OS scheduler in order to analyze if some changes should be required for an optimal scheduling. Results show that, using up to 4 threads, they are scheduled on the big cluster and the performance and energy results are similar. When adding 4 more threads, they are scheduled on the little cluster. The applications achieve good performance scalability when adding the LITTLE cores. In addition, its power-efficiency results in a reduction of the energy consumed by the applications when using 8 cores. The only exception is Kmeans. However, despite the small increment of the energy consumption as compared to the single-thread version, it is balanced by the execution time reduction achieved when using the little cluster. This results in an improvement of the EDP in all the cases.

2.2.3 Comparing little cluster with big cluster results

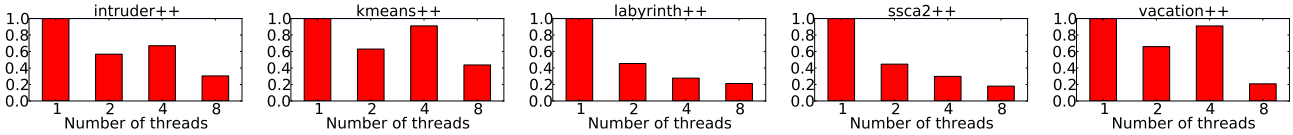
Fig. 4 shows the performance obtained in the little cluster as compared to the big cluster. Regarding execution time (Fig. 4.a), values higher than one reveal that the big cluster performs better than the little cluster. Running the applications with a single thread results in better performance for the big cluster. However, we observe that (with the exception of Labyrinth), as we keep adding more threads, more memory conflicts appear increasing the transaction abort rate, so as the big cluster is less efficient than the little cluster. Labyrinth, which features long transactions with



a. Normalized execution time w.r.t single-thread running TinySTM

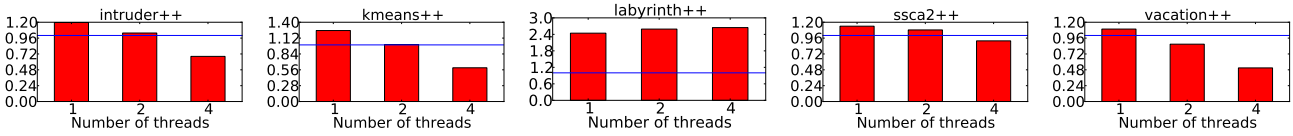


b. Normalized energy consumption w.r.t single-thread running TinySTM

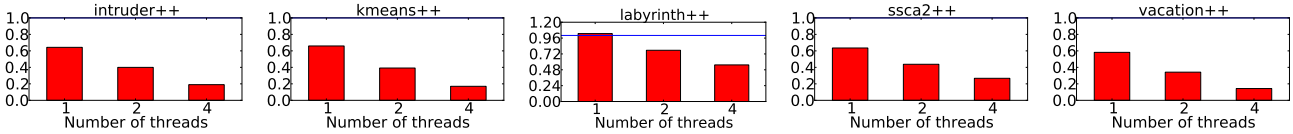


c. Normalized EDP w.r.t single-thread running TinySTM

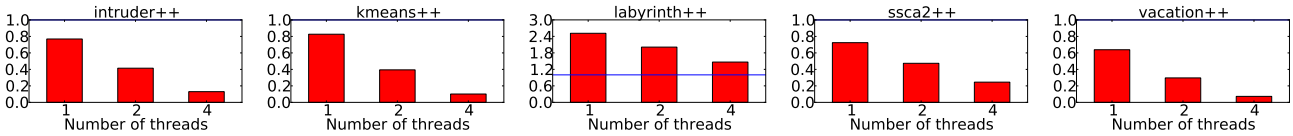
Figure 3: Full system evaluation



a. $ExTime_{little}/ExTime_{big}$



b. $Energy_{little}/Energy_{big}$



c. EDP_{little}/EDP_{big}

Figure 4: Big cluster vs little cluster evaluation

many accesses to memory and where most of the code in inside a critical section, requires of a performance-oriented multi-core processor and thus is able to get full advantage of the big cluster. A similar comparison was performed for the energy consumption (Fig. 4.b). In this case, the low-power Cortex-A7 cores perform better than the Cortex-A15 cores in terms of energy consumed. The same applies for the EDP, except for Labyrinth. For this application, as the big cluster performed (in terms of execution time) much better than the little cluster, the EDP shows improvements between 1.5X and 2.5X despite the higher energy requirements.

3 Conclusions and Current Work

In this work we present an empirical evaluation of a state-of-the-art software TM library running on an platform based on the ARM big.LITTLE architecture. Our results show that there is no an easy answer on how to schedule transactions on the cores to obtain the best combined energy-performance value (EDP), as it depends on the workload characteristics (see, for instance, Laberinth versus rest of codes).

Current research involves using this type of experiments to build an energy-aware runtime scheduler for TM aimed at exploiting the benefits of heterogeneous architectures, considering the flexible HMP (heterogeneous multi-processing) use model, that enables all physical cores at the same time.

References

[1] Advances in big.little technology for power and energy savings. `http://www.thinkbiglittle.com`.

[2] ODROID — Hardkernel. `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127`.

[3] A. Baldassin, J. P. L. de Carvalho, L. A. G. Garcia, and R. Azevedo. Energy-performance tradeoffs in software transactional memory. In *IEEE 24th Int'l. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD'12)*, pages 147–154, Oct 2012.

[4] H.W. Cain, M.M. Michael, B. Frey, C. May, D. Williams, and H. Le. Robust architectural support for transactional memory in the power architecture. In *40th Ann. Int'l. Symp. on Computer Architecture (ISCA'13)*, pages 225–236, Jun 2013.

[5] P. Felber, C. Fetzer, P. Marlier, and T. Riegel. Time-based software transactional memory. *IEEE Trans. on Parallel and Distributed Systems*, 21(12):1793–1807, 2010.

[6] E. Gaona, R. Titos-Gil, M.E. Acacio, and J. Fernandez. Dynamic serialization: Improving energy consumption in eager-eager hardware transactional memory systems. In *2012 20th Euromicro Int'l. Conf. on Parallel, Distributed and Network-based Processing (PDP'12)*, pages 221–228, Feb 2012.

[7] T. Harris, J. Larus, and R. Rajwar. *Transactional Memory, 2nd Ed.* Morgan & Claypool Publishers, USA, 2010.

[8] M. Herlihy and J.E.B. Moss. Transactional memory: Architectural support for lock-free data structures. In *20th Ann. Int'l. Symp. on Computer Architecture (ISCA'93)*, pages 289–300, Jun 1993.

[9] F. Klein, A. Baldassin, G. Araujo, P. Centoducatte, and R. Azevedo. On the energy-efficiency of software transactional memory. In *22nd Ann. Symp. on Integrated Circuits and System Design: Chip on the Dunes (SBCCI'09)*, pages 33:1–33:6, Sep 2009.

[10] C.C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IEEE Int'l. Symp. on Workload Characterization (IISWC'08)*, pages 35–46, Sep 2008.

[11] T. Moreshet, R.I. Bahar, and M. Herlihy. Energy-aware microprocessor synchronization: Transactional memory vs. locks. *4th Ann. Boston-Area Architecture Workshop (BARC'06)*, page 21, Feb 2006.

[12] W. Ruan, Y. Liu, and M. Spear. STAMP need not be considered harmful. In *9th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT'14)*, Mar 2014.

[13] S. Sanyal, S. Roy, A. Cristal, O.S. Unsal, and M. Valero. Clock gate on abort: Towards energy-efficient hardware transactional memory. In *IEEE Int'l. Symp. on Parallel Distributed Processing (IPDPS'09)*, pages 1–8, May 2009.

[14] R.M. Yoo, C.J. Hughes, K. Lai, and R. Rajwar. Performance evaluation of intel transactional synchronization extensions for high-performance computing. In *Int'l Conf. on High Performance Computing, Networking, Storage and Analysis (SC'13)*, pages 19:1–19:11, Nov 2013.