Characterizing the Energy Consumption of Software Transactional Memory

Alexandro Baldassin, Felipe Klein, Guido Araujo, Rodolfo Azevedo, and Paulo Centoducatte
Institute of Computing, University of Campinas (UNICAMP), Brazil
{alebal, klein, ducatte, rodolfo, guido}@ic.unicamp.br

Abstract—The well-known drawbacks imposed by lock-based synchronization have forced researchers to devise new alternatives for concurrent execution, of which transactional memory is a promising one. Extensive research has been carried out on Software Transaction Memory (STM), most of all concentrated on program performance, leaving unattended other metrics of great importance like energy consumption. This letter presents a thorough evaluation of energy consumption in a state-of-the-art STM. We show that energy and performance results do not always follow the same trend and, therefore, it might be appropriate to consider different strategies depending on the focus of the optimization. We also introduce a novel strategy based on dynamic voltage and frequency scaling for contention managers, revealing important energy and energy-delay product improvements in high-contended scenarios. This work is a first study towards a better understanding of the energy consumption behavior of STM systems, and could prompt STM designers to research new optimizations in this area, paving the way for an energy-aware transactional memory.

Index Terms—Parallel Architectures, Multiprocessor Systems, Transactional Memory, Power Management, Energy Consumption.

1 Introduction

The shift towards multicore processors and their subsequent mainstream adoption have drastically increased the need for better and effortless methods for building concurrent software. The prevalent parallel shared-memory programming model employs locks and condition variables as the standard synchronization primitives. The well-known drawbacks imposed by lock-based synchronization [13] have forced researchers to devise new alternatives of which transactional memory [5] is a promising one. The implementation substrate for transactional memory can be realized entirely via software (STM), through hardware components (HTM) or as a combination of both hardware and software (hybrid approach). In this letter, we shall focus our attention specifically on STM.

Extensive research has been carried out on the design space of STM. To evaluate a proposed implementation, researchers have invariably concentrated on performance, usually by measuring the system throughput in the form of transactions per second. We argue that, if STM is to become mainstream, other factors should be equally taken into account when devising a new design. In particular, energy-efficiency is of great importance and must be traded off with performance. This is specially true for embedded systems, where the energy consumption is closely related to battery lifetime. It is also of increasing interest in the non-mobile arena, such as data centers and desktop environments [4], [1].

This work is a first step towards a better understanding of the energy behavior in STM systems. We make the following contributions: (i) first characterization of a state-of-the-art STM system based on the TL2 algorithm [2] and the STAMP suite [11]; (ii) a novel strategy based on dynamic voltage and frequency scaling (DVFS) for contention managers, intended to enhance both performance and energy consumption of applications with high abort rates. Our results reveal average improvements of 45% on energy consumption for those applications, achieving maximum reductions of up to 87%.

Manuscript submitted: 02-Jul-2009. Manuscript accepted: 23-Jul-2009. Final manuscript received: 05-Aug-2009. This work was supported in part by FAPESP (2005/02565-9).

2 Methodology

The energy characterization methodology herein described attempts to profile the energy consumption of an STM system in terms of its basic components. A number of transactional memory approaches have been proposed in the literature [5] and, though distinct, they are mostly built upon the same primitives, namely, *TxStart*, *TxCommit*, *TxLoad*, and *TxStore*. In addition to these primitives, we distinguish the costs due to the re-execution of aborting transactions (*rollback*) as well as the costs related to the contention management scheme. For convenience, a category labeled *Other* is introduced and include other minor transactional operations such as memory management (transactional allocs and frees). Notice that we include *TxStart* in this category in the experiments.

2.1 Simulation Platform

A cycle-accurate MPSoC simulation platform [9] is used to collect accurate energy and performance numbers. Its main components are: (i) a variable number of ARMv7 processors, each with a 8KB, 4-way I-cache and a 4KB, 4-way D-cache; (ii) their private memory (12MB each); (iii) a shared memory (16MB); (iv) a hardware semaphore module, providing support for *test-and-set* operations; and (v) an AMBA AHB bus interconnect. The platform models have been characterized on a 0.13-μm technology by STMicroelectronics and validated on silicon implementations of the various components. It has also been used in numerous papers in the literature (see [7], for instance). Two important observations are worth mentioning. Firstly, the memory architecture is based on SRAM and, hence, has lower latency and is more energy-efficient as compared to DRAM [10]. Secondly, cache coherency is not enforced by hardware: private reads and writes are cacheable, while shared accesses are not. As showed by others in [8], this approach appears to be competitive in terms of performance and energy with respect to hardware-based cache coherence. Although the platform is more suited for embedded systems, we believe the results shown in this letter still hold for general CMP architectures.

Posted to the IEEE & CSDL on 8/27/2009
DOI 10.1109/L-CA.2009.47 1556-6056/09/$25.00 © 2009 Published by the IEEE Computer Society

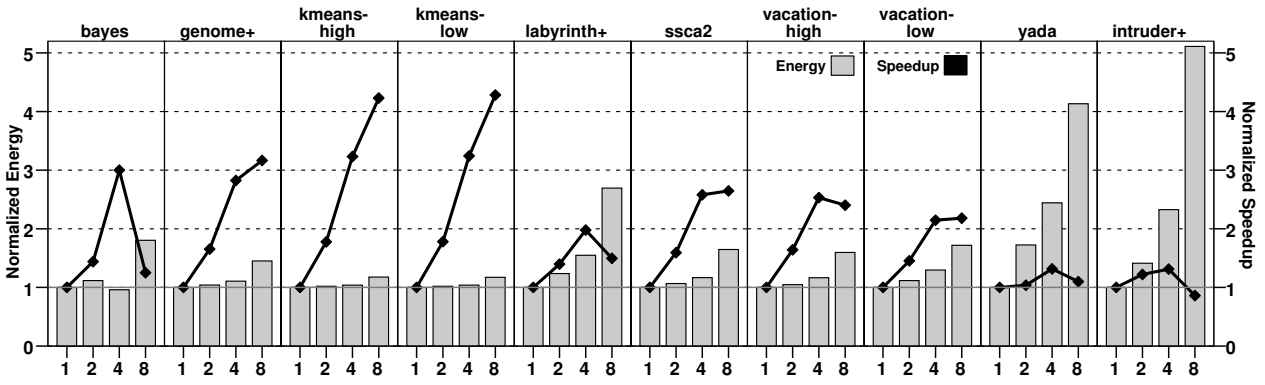

Fig. 1. Energy and speedup figures for the STAMP applications using the lazy version of TL2 with linear backoff. Both energy and speedup are normalized to that of the transactional single-core case.

2.2 Energy Profiling Procedure

In order to proceed with the characterization process, we need to distinguish the energy consumed by the STM infrastructure from the energy consumed by the application itself. The energy profiling phase is performed as follows:

- **API profiling**: after entering the STM code, through a call to its API, the energy measurement of the primitive is activated. During this period, the energy consumed by all components of the platform (processor, caches, main memory and bus) are monitored and recorded for each primitive. Prior to returning to the application code, the measurement for that primitive is deactivated and recorded.
- **Application profiling**: the application starts and energy measurement is initially activated. Prior to any call to the STM API the measurement is deactivated, recorded, and then reactivated when returning to the application code. Similar as above, all platform's components are taken into account and the resulting aggregate energy is recorded.
- **Rollback profiling**: whenever a transaction aborts, all energy recorded since the last *TxStart*, including the primitives and application code is subsumed under the term *rollback*. The only exception is the energy due, specifically, to the contention manager, which is tagged differently (*backoff*).

The procedure aforementioned allow us to assess individually the inherent energy costs of the application and also the overhead for each of the STM primitives. This facilitates the identification of possible bottlenecks in STM systems and could prompt STM designers to devise optimizations and improve their implementations, so as to reduce the energy footprint imposed by the STM approach. Also notice that the methodology is general enough and could be seamlessly applied to any STM implementation exposing a compatible API.

2.3 TL2 and STAMP

We make use of the TL2 implementation distributed with the STAMP benchmark suite [11]. Besides the original lazy version (*TL2-lazy*), it also features an eager version (*TL2-eager*) wherein locks are acquired during *TxStore* and memory is updated in place. Two contention management strategies based on backoff are provided (*linear* and *exponential*), being triggered after three consecutive aborts. Porting the x86 TL2 implementation and the STAMP applications to the simulation platform was mostly straightforward, requiring attention when mapping shared data into the platform's shared space. The only significant modification made to the TL2 code was that we reduced the size of the hash table to 256KB (originally 4MB). The *CAS* operation required by the STM algorithm is built upon the *test-and-set* primitive provided by the platform. All 8 STAMP applications are used in our experiments, characterizing different transactional scenarios with regard to transaction length, read and write set sizes, transaction time and contention level. The input parameters are the recommended ones for running in simulation environments. We show the results for 10 application variants, following the nomenclature used in the original STAMP paper [11].

3 Energy Characterization & Analysis

The evaluation of STM designs has traditionally focused on performance. If energy consumption could directly be predicted from run time there would be no motivation not to continue evaluating STM systems purely on performance. Therefore, it is of the utmost importance to provide evidence that disassociates energy from speedup, as suggested by Figure 1. As can be seen from this figure, increasing the number of cores invariably causes more energy to be consumed (except for bayes with 4 cores). However, speedup does not always follow the same trend: while it does increase for some applications (such as `genome+` and `kmeans`), it actually decreases for others (most notably, `intruder+`). The main reason for the latter, as will be seen in more detail shortly, is the excessive number of aborts and consequent rollback time wasted as the number of cores increases. Although not shown due to lack of space, we observed the same behavior with *TL2-eager*. Altogether, Figure 1 offers a compelling evidence that energy consumption cannot be predicted directly from performance.

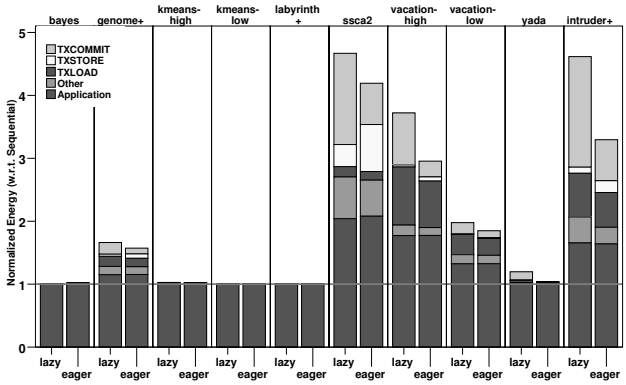

Fig. 2. STM energy breakdown for both *TL2-lazy* and *TL2-eager* in a single-core configuration. Energy is normalized to that of the sequential code.

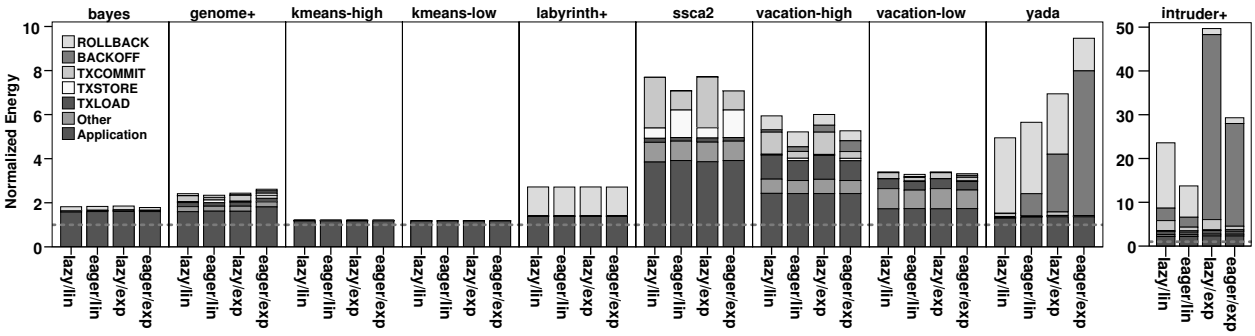

Fig. 3. STM energy breakdown for both *TL2-lazy* and *TL2-eager* in an 8-core configuration, with linear and exponential backoff policies. Energy is normalized to that of the sequential code.

Software transactions are known for introducing performance overhead due to the data versioning and conflict detection schemes. In order to analyze the energy impact, Figure 2 shows the energy breakdown with a single processor. A couple of observations can be made regarding this figure. Firstly, some applications seem not to have any significant overhead. More specifically, `bayes`, `labyrinth+`, and `yada` have long transactions and the overhead due to transactional primitives is low (a small *TxCommit* overhead can be noticed in `yada-lazy`). The scenario is different for `kmeans`, however. This particular application makes extensive use of floating-point operations, which are emulated via software in our platform. As it turns out, the overhead due to these operations tends to hide the cost one would normally see from the transactional primitives (this very same behavior happens, although in a much lesser extent, to `yada`). Secondly, the application code overhead is significant (about 2x) for `ssca2`, `vacation` and `intruder+`. The main reason for this is that the compiler does a very good job in optimizing the sequential code, whereas it cannot do much in the transactional case due to the insertion of the primitives. Lastly, it can be noticed that the *TxStore* operation is cheaper in the lazy configuration, while *TxCommit* is cheaper in the eager version (recall that locking is performed during *TxStore* in *TL2-eager*, and during *TxCommit* in *TL2-lazy*). Also, the cost of the *TxLoad* operation is cheaper for *TL2-eager*, since the read value does not need to be looked up in the write set (*TL2-lazy* uses a Bloom filter to avoid iterating over the write set).

While providing some insight into the energy consumption of STM, the single-core breakdown does not exhibit the percentage of energy spent with aborts and rollbacks. To remedy the situation, Figure 3 shows the energy breakdown for an 8-core configuration. First off, notice that the energy overhead for `kmeans` is still negligible. This is due to its low transaction time, short transactions and good load balancing. Again, the excessive use of floating-point operations obscure the appearance of any overhead caused by the transactional primitives. The low abort rate displayed by applications `bayes` (∼7.4%), `genome+` (∼2.2%), `ssca2` (∼0.2%), and `vacation-low` (∼4.4%) precludes most of the overheads caused by backoff and rollback from showing off. Despite having a relatively high abort rate (∼30%), application `labyrinth+` does not have significant backoff time since its long-running transactions and consequent rollback time do not allow the backoff mechanism to be activated (recall that it is only activated after 3 retries). The remaining 3 applications (`vacation-high`, `yada`, and `intruder+`) show both backoff and rollback costs in different proportions, according to their abort rates. We observe that, most notably in `yada` and `intruder+`, the energy spent while backing off and rolling back dominates the overall energy consumption, even though no useful work is done during those periods. These two operations usually correlate to each other and depend heavily on the policies adopted for contention management. Thus, as the results show, there is considerable room for improvement, which is exploited by the strategy introduced in the next section.

4 DVFS-BASED STRATEGY

We leveraged TL2's original contention management policies in order to exploit the slack available in applications displaying high contention. For that purpose, we adopted a strategy based on dynamic voltage and frequency scaling [4]. Since power depends quadratically (linearly) on voltage (frequency), the power-efficiency could be theoretically improved cubically.

Any contention manager causing transactions to wait are eligible to use this technique. The strategy, which is a simple, but effective one is as follows: prior to entering the backoff mode, the processor is put in a low-power mode by reducing both the frequency and voltage. Then, the processor stalls for an amount of time proportional to the number of retries of the transaction attempting to commit. Hence, the energy wastage is reduced without degrading performance significantly, given that such periods are considered *idle time* and perform no useful work. Upon completion of the backoff period, the processor frequency/voltage is rescheduled to full speed so as to avoid impacting on the overall performance. Since the DVFS strategy requires switching the processor between different states, one should be aware of the extra overhead when applying the mechanism so as not to degrade the overall performance. Switching to and from low-power mode incurs, in our simulation environment, a 2-cycle penalty, causing the frequency to be scaled between 200MHz and 1.56MHz.

Figure 4 presents the results achieved by the proposed scheme for STAMP. In order to correlate the impact on both energy and performance, the numbers are exposed in terms of energy and energy-delay product (EDP). We make four major observations about this figure. Firstly, for those applications displaying medium-to-high contention, namely, `intruder+`, `yada`, and `vacation-high`, the scheme effectively reduced the energy consumption. On average, the energy was reduced by a ∼45% factor, and up to 87% for `intruder+`. As a positive side-effect of this optimization, the abort rate was reduced, since the aborted transactions stayed longer (due to lower frequencies) in backoff mode thus avoiding a premature re-execution which was doomed to fail. Consequently, performance was increased by ∼13% (on average) and the resulting

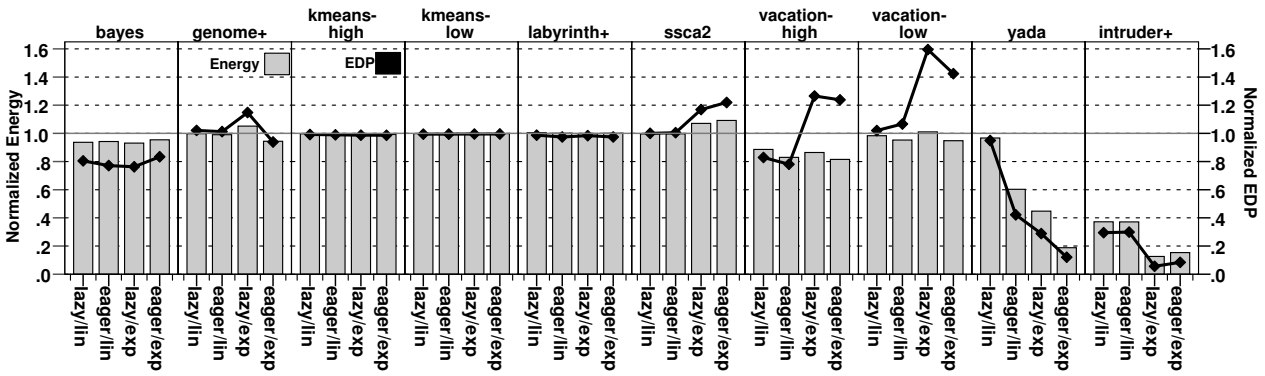

Fig. 4. Energy and EDP figures for the proposed DVFS-based strategy in an 8-core configuration with both linear and exponential backoffs. Results are normalized to those presented in Figure 3.

EDP decreased substantially. Secondly, some applications experienced negative results with some configurations of the DVFS-based exponential backoff. In `genome+-lazy`, `ssca2-lazy` and `ssca2-eager`, there is very low contention, which incurs entering the backoff mode just a handful of times. However, due to the low abort rates ($\leq$3%), the time spent backing off is increased more than necessary, counteracting the benefits of running it in low-power mode. Moreover, since there is no slack to be exploited, both energy and EDP are slightly but adversely affected by this behavior. A similar explanation holds for the EDP increase in `vacation-low` and `vacation-high` (both with exponential backoff). But, contrary to the previous applications, there is a little slack available, which is properly exploited, as it can be perceived by the decrease in the total energy. Notice that the same applications are not influenced by the mentioned behavior when using the linear backoff scheme. Thirdly, for those applications displaying low STM energy overhead, namely, `kmeans-high`, `kmeans-low`, and `labyrinth+`, the achieved results were virtually the same. This happens due to the small number of retries per transaction, which precludes the processor from entering in backoff mode. Finally, for `bayes`, even though there is only a thin margin of *rollback* and *backoff* available, the proposed scheme nearly halved that amount, resulting in an average improvement of $\sim$6% and $\sim$20%, in total energy and EDP, respectively.

5 Related Work

As previously mentioned, current evaluation of STM designs primarily addresses performance improvements over traditional locks. The usual performance metric is given by the number of transactions executed per unit of time (i.e., throughput). We are not aware of any methodology for estimating the energy consumption of STM systems. The works on power dissipation in HTM systems are the closest to ours. Moreshet et al. [12] initially investigated the energy consumed by transactions in a typical multiprocessor environment. While their results suggested that HTM has an advantage in terms of energy consumption over locks, one should notice that only micro-benchmarks were employed in the experiments. More recently, Ferri et al. [3] evaluated the impact of energy consumption in an embedded setting. They also proposed the use of a scratchpad memory for transaction checkpointing and a technique that shuts down the transactional cache in case of under-utilization. Our DVFS-based approach resembles the strategy used by the thrifty barrier of Li et al. [6], in which a processor is forced into a low-power state when spinning on synchronization barriers.

6 Conclusions and Future Work

This letter presented, for the first time, the energy characterization of a state-of-the-art STM using the STAMP benchmark suite. We thoroughly evaluated the impact on energy consumption due to STM and quantified the energy costs of the primitives in the adopted time-based STM implementation. Furthermore, we proposed a novel energy-aware DVFS-based strategy for contention managers in order to exploit the slack available in applications. For those displaying high-contention total energy (EDP) was reduced, on average, by 45% (45%), achieving maximum improvements of 87% (96%). This work is a first study towards a better understanding of the energy consumption behavior of STM systems. Future research will address how the STM algorithm itself could be changed in order to provide even better energy efficiency. We also plan to devise an energy macromodel for STM in order to allow energy analysis with the help of more abstract simulation models.

References

[1] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.
[2] D. Dice, O. Shalev, and N. Shavit, "Transactional locking II," in *Proc. of the 20th DISC*, 2006.
[3] C. Ferri, A. Viescas, T. Moreshet, R. I. Bahar, and M. Herlihy, "Energy efficient synchronization techniques for embedded architectures," in *Proc. of the 18th GLSVLSI*, 2008, pp. 435–440.
[4] S. Kaxiras and M. Martonosi, *Computer Architecture Techniques for Power-Efficiency*. Morgan & Claypool Publishers, 2008.
[5] J. R. Larus and R. Rajwar, *Transactional Memory*. Morgan & Claypool Publishers, 2007.
[6] J. Li, J. F. Martinez, and M. C. Huang, "The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors," in *Proc. of the HPCA-10*, 2004, pp. 14–23.
[7] M. Loghi, L. Benini, and M. Poncino, "Analyzing power consumption of message passing primitives in a single-chip multiprocessor," in *Proc. of the ICCD'04*, 2004, pp. 393–396.
[8] M. Loghi, M. Poncino, and L. Benini, "Cache coherence tradeoffs in shared-memory MPSoCs," *ACM TECS*, vol. 5, no. 2, pp. 383–407, 2006.
[9] M. Loghi, M. Poncino, and L. Benini, "Cycle-accurate power analysis for multiprocessor systems-on-a-chip," in *Proc. of the 14th GLSVLSI*, 2004, pp. 410–406.
[10] A. Macii, L. Benini, and M. Poncino, *Memory Design Techniques for Low Energy Embedded Systems*. Springer, 2002.
[11] C. C. Minh, J. W. Chung, C. Kozyrakis, and k. Olukotun, "STAMP: Stanford transactional applications for multi-processing," in *Proc. of the IEEE IISWC*, 2008, pp. 35–46.
[12] T. Moreshet, R. I. Bahar, and M. Herlihy, "Energy reduction in multiprocessor systems using transactional memory," in *Proc. of ISLPED*, 2005, pp. 331–334.
[13] H. Sutter and J. R. Larus, "Software and the concurrency revolution," *Queue*, vol. 3, no. 7, pp. 54–62, 2005.