2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing

Energy-Performance Tradeoffs in Software Transactional Memory

Alexandro Baldassin, João P. L. de Carvalho
UNESP – Univ Estadual Paulista, Rio Claro, Brazil
alex@rc.unesp.br, jaopaulolc@gmail.com

Leonardo A. G. Garcia
Linux Technology Center - IBM, Brazil
lagarcia@br.ibm.com

Rodolfo Azevedo
Campinas State University, Brazil
rodolfo@ic.unicamp.br

Abstract

Transactional memory (TM) is a new synchronization mechanism devised to simplify parallel programming, thereby helping programmers to unleash the power of current multicore processors. Although software implementations of TM (STM) have been extensively analyzed in terms of runtime performance, little attention has been paid to an equally important constraint faced by nearly all computer systems: energy consumption. In this work we conduct a comprehensive study of energy and runtime tradeoffs in software transactional memory systems. We characterize the behavior of three state-of-the-art lock-based STM algorithms, along with three different conflict resolution schemes. As a result of this characterization, we propose a DVFS-based technique that can be integrated into the resolution policies so as to improve the energy-delay product (EDP). Experimental results show that our DVFS-enhanced policies are indeed beneficial for applications with high contention levels. Improvements of up to 59% in EDP can be observed in this scenario, with an average EDP reduction of 16% across the STAMP workloads.

1. Introduction

Excessive power consumption and microarchitectural limitations have reshaped the microprocessor industry, bringing multicore architecture and parallel programming into the mainstream. Despite the fact that multicore hardware is everywhere, successfully writing concurrent applications that can make the most out of the available parallelism is still a gift reserved to a few. The common belief is that existing languages and tools are inadequate, and new programming models and tools must be developed with parallelism in mind [28].

Transactional Memory (TM) [14] is a novel abstraction that has the potential of simplifying parallel synchronization, thereby helping programmers to harness the power of multicore processors. While standard synchronization methods rely on locks and condition variables, the transactional model offers the concept of a *transaction*: a sequence of code that is executed atomically. As a result, programmers are freed from the burden of devising complicated locking protocols and are primarily responsible for identifying the atomic blocks. TM systems can be implemented entirely in software (STM), using hardware resources (HTM), or as a combination of both (HyTM). The rest of this paper focuses on STM.

There has been a lot of research on STM algorithms and implementations over the last few years, with performance being a major concern [3, 7]. Although execution time is an important constraint, power consumption has also become a primary restriction for nearly all computer systems. In embedded computing it directly affects battery lifetime, while thermal issues are a key limitation in desktop and server systems [16]. However, little is known about energy consumption in STM systems.

In this work we present a thorough investigation of energy and performance tradeoffs in STM systems. Using a cycle-accurate MPSoC simulation platform [22] and the STAMP benchmark [23], we perform a detailed study of runtime and energy consumption in STM systems by covering a large portion of the transactional design space (encounter-time versus commit-time locking, eager versus lazy versioning) and contention management policies (suicide, backoff and delay). We also show how dynamic voltage and frequency scaling (DVFS) [16] can be combined with contention managers in order to improve both energy efficiency and performance.

More specifically, the main contributions of this paper are:

- a detailed characterization of three state-of-the-art lock-based STM algorithms and three conflict resolution policies using the STAMP benchmark suite. Our experiments show that the encounter-time locking with lazy versioning algorithm, along with the suicide policy, provided the best overall energy/performance compromise for the STAMP workloads;

- a DVFS-based methodology that can be integrated into conflict resolution policies to improve both energy consumption and runtime performance. We evaluated our methodology with the backoff and delay policies, and both provided significant gains for applications with high contention levels. Our results show an EDP reduction of 16% across the STAMP applications with regard to the best previously analyzed policy (suicide).

The rest of this paper is organized as follows. Section 2 describes the STMs and resolution policies employed in our study. Section 3 introduces the simulation platform and transactional applications and configurations, while Section 4 presents the characterization of the STM algorithms and policies. Section 5 explains our motivation for using DVFS and shows the experimental results obtained with its use. Finally, Section 6 discusses related works and Section 7 concludes the paper.

2. STM Design Space

There have been a plethora of STM designs proposed since the seminal work of Shavit and Touiou [27]. On a more abstract level, the algorithms can be divided into two broad categories: blocking and non-blocking. In this work we concentrate on blocking designs since they are very popular and considered to have superior performance than non-blocking alternatives [6, 7]. Harris et al. [14] provide a comprehensive study on the design space of transactional memory in general.

Blocking algorithms rely on some sort of locking protocol to control the access to shared memory. The granularity of the data protected by a lock may be a single location or an entire object. The main choices that need to be considered in the design of a lock-based STM are: (i) the moment in which a location is locked; (ii) how data versioning is handled. Commit-time locking (CTL) algorithms acquire the locks for the accessed locations during the commit phase, while in encounter-time locking (ETL) systems the lock is acquired when a location is first accessed. A transactional system may also store the speculative data in a local buffer (lazy versioning) or directly in shared memory (eager versioning).

In this paper the following designs are considered:

- CTL – commit-time locking with lazy versioning [1]. The TL2 algorithm [5] is the main representative of this design class;
- ETL_l – encounter-time locking with lazy versioning, popularized by TinySTM [8];
- ETL_e – encounter-time locking with eager versioning, first analysed by Saha et al. [25].

[1]Notice that CTL cannot provide eager versioning, since the locks are only acquired during the commit operation.

Another important design choice for any STM system is the resolution policy employed in case of conflicts. A conflict occurs when two or more transactions access the same location and at least one of them performs a write access. The contention manager is the STM module responsible for choosing a strategy to resolve the conflict. The following policies are used in our evaluation:

- `SUICIDE` – the transaction that detected the conflict is aborted and immediately restarted;
- `BACKOFF` – an adaptive backoff mechanism [1] is employed to delay the restart of a transaction after it is aborted. The delay is increased exponentially after every restart;
- `DELAY` – before restarting, the aborted transaction waits for the lock that caused the conflict to be released.

Common to all the studied policies is the fact that the transaction that detects the conflict is aborted. They mostly differ in the actions taken before the transaction is restarted. While `SUICIDE` does not perform any further action (the transaction is restarted immediately), `BACKOFF` waits an exponentially increasing time before restarting and `DELAY` waits for the contention on the lock responsible for the conflict to disappear. The main idea behind both `BACKOFF` and `DELAY` is that, by waiting for certain events to happen, the probability of finding another conflict upon retry is reduced.

3. Simulation Platform and Applications

The results presented in this paper have been collected using a cycle-accurate MPSoC simulation platform [22], whose overall organization is depicted in Figure 1. The simulation platform allows a variable number of ARMv7 processors to be instantiated, each one with segregated instruction and data caches. Each core has also access to a external private memory by means of a interconnection network, which is also connected to a global shared memory and a hardware semaphore module. The platform provides accurate power models for each component, characterized on a 0.13-μm technology by STMicroelectronics and validated on silicon. It has also been used by others to show

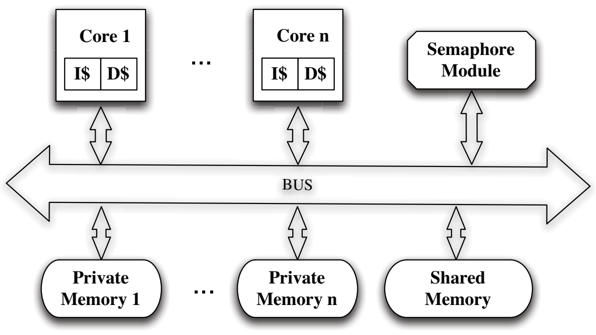

Figure 1. Simulation platform.

Component	Configuration
Processor	ARMv7 @200MHz
L1 instruction cache	8KB, 4-way set associative
L1 data cache	4KB, 4-way set associative
Private and shared memory	16MB each, SRAM
Bus	AMBA AHB

Table 1. Platform configuration used in the experiments.

the energy and performance impact of multiprocessor systems [21, 9, 2].

The precise configuration for the platform used in our experiments is given in Table 1. Since the simulated ARMv7 processors do not provide the compare-and-swap (CAS) instruction needed by the STM algorithms, it is implemented through a semaphore-based test-and-set operation (provided by the semaphore hardware module). Also, the memory architecture of the platform is based on SRAM and, consequently, has lower latency and is more energy-efficient compared to DRAM. As can be noticed, the platform configuration is typical of embedded systems.

The transactional applications used in our evaluation are taken from the STAMP benchmark suite [23]. They characterize different transactional scenarios with regard to transaction length, read and write set sizes, transaction time and contention level. The arguments for each application are the recommended for running in simulation environments, and are reproduced in Table 2.

The STM implementation used in our experiments is based on the publicly available distribution of TinySTM [8], version 1.0.3. The algorithm variations (CTL, ETL_l, and ETL_e) and basic resolution policies (`BACKOFF`, `DELAY`, and `SUICIDE`) are already provided with TinySTM. To validate our experiments, the results produced by each application in the simulation environment are checked against the output generated by the same application (and input set) on an x86 machine.

Application	Arguments
`Bayes`	-v16 -r1024 -n2 -p20 -i2 -e2
`Genome`	-g512 -s32 -n32768
`Intruder`	-a10 -l16 -n4096 -s1
`Kmeans`	-m15 -n15 -t0.05 -i random-n2048-d16-c16
`Labyrinth`	-i random-x48-y48-z3-n64
`SSCA2`	-s13 -i1.0 -u1.0 -l3 -p3
`Vacation`	-n2 -q90 -u98 -r16384 -t4096
`Yada`	-a20 -i 633.2

Table 2. Application configurations.

4. Characterization

In order to present the performance and energy characteristics of the different STM designs and policies we proceed as follows. First, we study the scalability and energy efficiency as the number of cores is increased from 1 to 8. For this study we focus on the ETL_l design with `SUICIDE`, as it is a popular configuration (for instance, it is the default configuration for TinySTM 1.0.3). Second, we perform a comparative analysis showing the advantages and disadvantages of each design and policy, for each STAMP application.

4.1 Energy versus Speedup

Transactional memory systems have been traditionally evaluated using performance metrics such as transaction throughput and execution time. Although energy consumption is proportional to execution time, it is not always trivial to infer one from the other. In order to understand the relationship between energy and performance of transactional applications, we present in this section an analysis using an STM with the ETL_l design and the `SUICIDE` policy, whose behavior is characterized in Figure 2.

Notice that, while energy consumption invariably increases with the number of execution cores, the speedup numbers depend on the application. For instance, applications `Genome`, `Kmeans`, `Labyrinth`, `SSCA2`, and `Vacation` display better performance as more cores are added. On the other hand, `Bayes`, `Intruder`, and `Yada` exhibit a performance loss when the number of cores is increased from 4 to 8. This behavior is mostly due to a high contention level reached by these applications when 8 cores are used, causing many transactions to abort repeatedly.

Furthermore, observe that the decrease in performance experienced by `Bayes`, `Intruder`, and `Yada` is closely related to the growth in the energy consumed. For example, `Intruder` speedup decreased by 50% (from 2x to 1.5x) while its energy consumption increased by 100% (from 1.5x to 3x). The configuration with 2 cores provides roughly the same performance with less than half the energy consumed. Conversely, applications with good speedups provided excellent energy savings, as can be noticed by

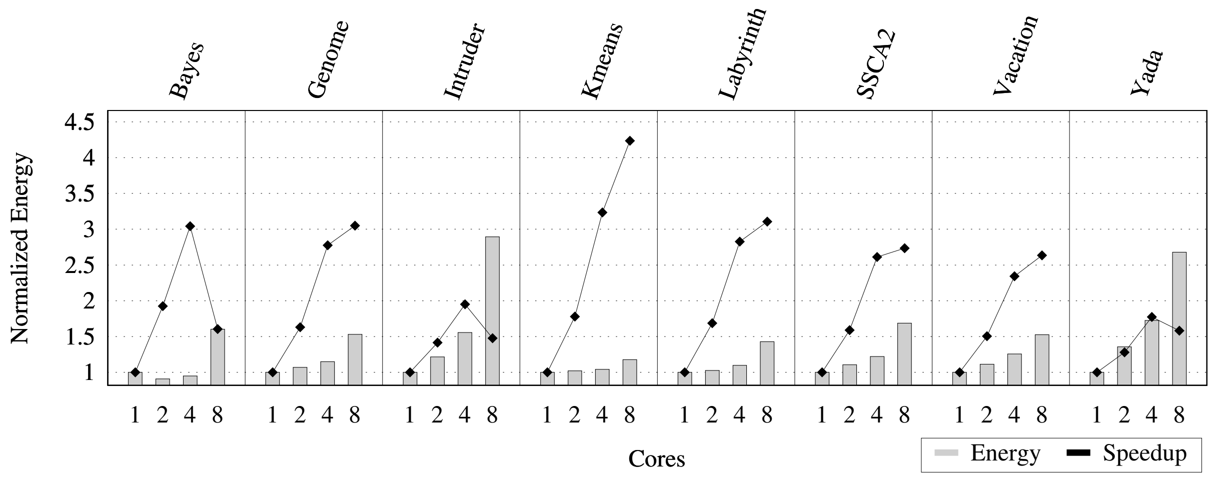

Figure 2. Energy and speedup. The values are for ETL_l with `SUICIDE` and are normalized wrt the respective transactional single-core execution.

`Kmeans`: the 8-core configuration displays a 4.2x speedup with only 15% of additional energy.

As especially evidenced by Intruder and Yada, when the contention level increases the amount of energy wasted is high. We revisit this problem in Section 5, offering a DVFS-based mechanism to alleviate the pressure on energy consumption and increase the performance.

4.2 Comparative Study

In the previous Section (4.1) we analyzed the energy and speedup behavior of the STAMP applications using a single STM configuration. In order to gain a better understanding of the STM design space, we conduct in this section a comparative investigation of the different transactional algorithms and policies presented in Section 2.

Figure 3 shows the energy and execution time for the three STM algorithms (CTL, ETL_l, and ETL_e) and policies (`BACKOFF`, `DELAY`, and `SUICIDE`). We only present results for the 8-core configuration for two main reasons: (i) to avoid cluttering the figure, thus simplifying the presentation; (ii) as Figure 2 shows, it is with 8 cores that applications start to exhibit contention and, consequently, it is in this scenario that the influence of a given algorithm and resolution policy is more evident. The energy and execution time values are also normalized to the ETL_l `SUICIDE` configuration. Therefore, if the normalized value for a given design is smaller than 1, the design is better than ETL_l `SUICIDE`. Otherwise (bigger than 1), it is worse.

We start by analyzing the relationship among the algorithms ETL_l, CTL, and ETL_e, highlighting some key differences among the resolution policies when necessary. For applications `Bayes`, `Kmeans`, `Labyrinth`, `Genome`, `SSCA2`, and `Vacation`, it is not possible to select a clear winner, since their results are similar (overall differences are smaller than 5%). Nonetheless, some observations are noteworthy. First, `SSCA2` suggests that ETL_e is more efficient than both ETL_l and CTL in terms of energy and execution time. Second, a small energy and performance loss can be observed in `Genome` with the CTL design, although the execution time of `Vacation` seems to benefit from it.

The advantages and drawbacks of the transactional algorithms are more evident in `Intruder` and `Yada`. In `Intruder`, the CTL design shows the worst performance, while ETL_l and ETL_e exhibit comparable results. It is evident from Figure 3 that `BACKOFF` is by far the less attractive policy for `Intruder`. For this application, a simple policy such as `SUICIDE` provided the best results, since the transactions are very small and it is preferable to restart a transaction immediately than providing a more sophisticated resolution scheme.

The results for `Yada` indicate that the `BACKOFF` policy did not provide any benefit for ETL_l and ETL_e, although it did perform very well with the CTL algorithm. Since transactions in `Yada` are large, postponing the acquisition of the locks until commit time has the potential to reduce the number of conflicts. As the results show, all CTL policies responded very well. Once more, the `SUICIDE` scheme presented the best results for `Yada`.

Overall, it is safe to say that the ETL_l algorithm and the `SUICIDE` resolution policy provided the best results. One exception is `SSCA2`, in which the ETL_e algorithm provided a small advantage in terms of energy and performance. Due to the results presented in this section, we focus our attention exclusively on the ETL_l algorithm in the rest of this paper.

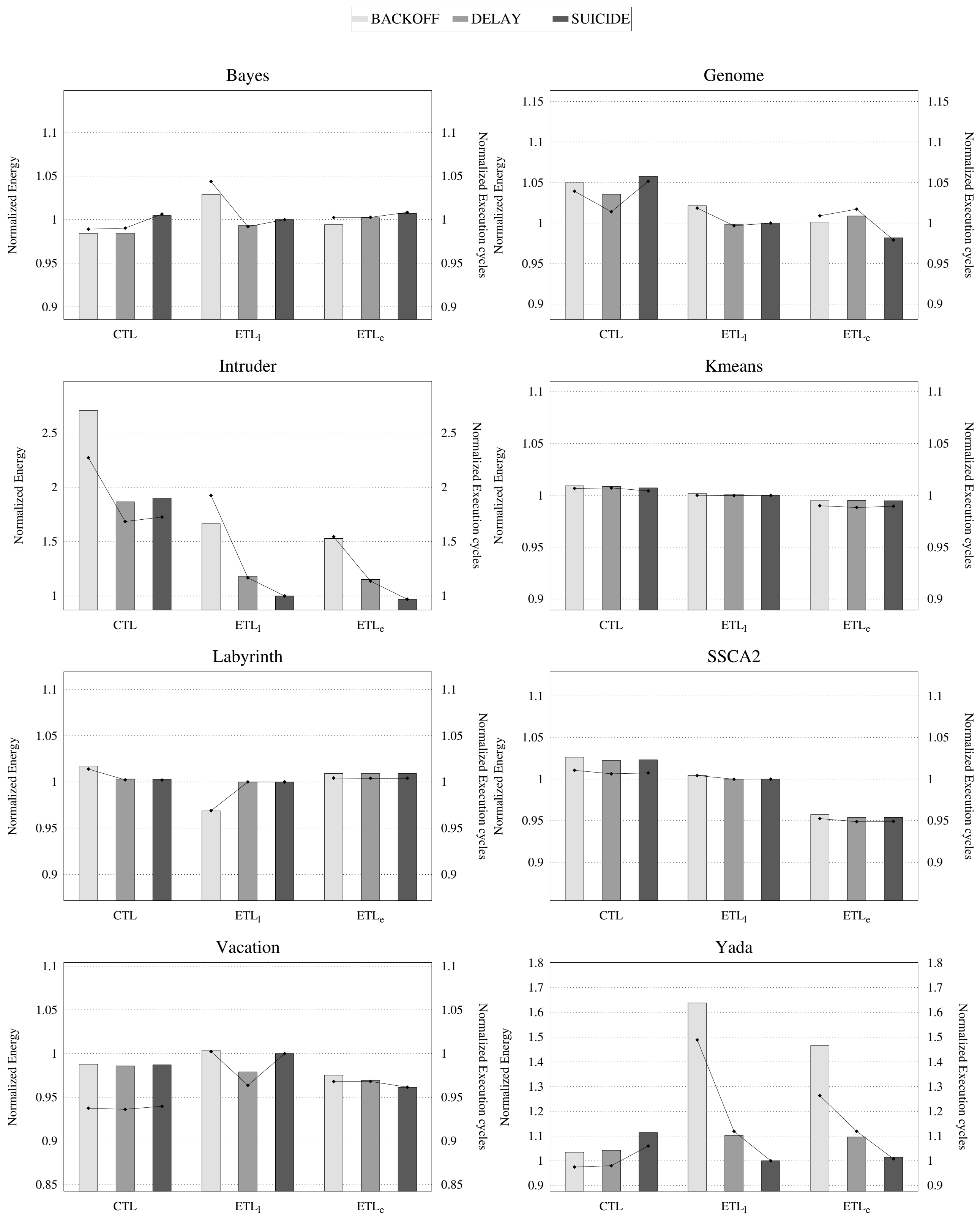

Figure 3. Relative energy and execution time of the STAMP applications. The results are for the 8-core configuration, normalized wrt ETL_l SUICIDE (8-core).

5. DVFS-Based Contention Management

The main goal of a contention management policy is to reduce the number of conflicts, thereby providing transaction throughput and allowing the transactional system to scale under heavy contention. As showed in the previous Section (4.2), the SUICIDE scheme provided the best overall behavior. On close inspection, it can be noticed that SUICIDE is not actually a *real* policy, in the sense that all it does it to restart the transaction immediately. More sophisticated policies (BACKOFF and DELAY) have been investigated also, but they did not provide any benefit at all in our experiments.

In order to investigate the reasons that prevented BACKOFF and DELAY from being effective, we conducted a detailed examination of their behavior by profiling their execution stages. Of particular interest is the time/energy spent while a transaction waits to be restarted after being aborted (transactional stalls). The following phases are considered in our profiling:

- APP – time or energy spent in the non-transactional part of the application;
- TX – this phase characterizes the time or energy spent while executing successfully committed transactions;
- ABORT – when a transaction aborts we subsume all the time or energy spent since it started under this category;
- STALL – for the BACKOFF and DELAY policies, a transaction is only allowed to restart when a specific event happens. The time or energy spent during this stall is represented by this category.

The breakdown results for the BACKOFF, DELAY and SUICIDE policies using the ETL_l algorithm in a 8-core configuration is showed in Figure 4. For this figure in particular we use the energy-delay product (EDP), a useful metric proposed by Gonzalez and Horowitz [12] with the goal of characterizing both low energy and fast runtimes. Figure 4 therefore presents the percentage of total EDP spent in each of the execution stages, for each STAMP application.

First of all, notice that the fraction of EDP resulted from non-transactional code (APP) varies widely from application to application. While it is practically absent in Labyrinth and Yada, it accounts for nearly 94% of the total EDP of Kmeans. This is expected, since the time spent in transaction is low for Kmeans.

The most important characteristic uncovered by Figure 4 is the fraction of EDP due to aborted transactions (ABORT) and transactional stalls (STALL), particularly for Intruder and Yada. Note that the policies BACKOFF and DELAY were indeed able to reduce the ABORT percentage, but then the STALL fraction prevented them from

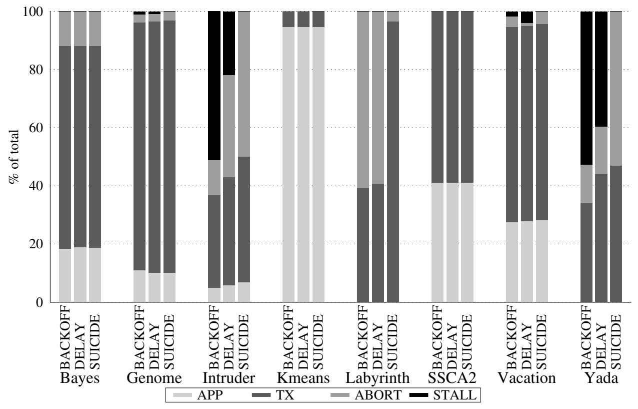

Figure 4. EDP breakdown for the STAMP applications with the three studied resolution policies in a 8-core configuration.

achieving practical use. In Intruder, for instance, BACKOFF was able to reduce the ABORT fraction from 50% (SUICIDE) to 12%, but the STALL fraction consumed 51% of the total EDP. Likewise, a reduction in the ABORT fraction with regard to SUICIDE can be observed in DELAY (from 51% to 35%), but its STALL ratio reached 22%. Similar considerations are also valid for Yada.

If we were able to somehow reduce the STALL fraction for BACKOFF and DELAY, they could as well outperform SUICIDE, as evidenced by Intruder and Yada. Motivated by this observation, we devised a DVFS-based strategy to improve the EDP that works as follows. Right after a transaction is aborted, the processor enters a low-power mode which is accomplished by reducing its frequency and voltage (from 200MHz to 1.56MHz). The processor then waits for the backoff period to end (BACKOFF) or for the contended lock to be released (DELAY) in an energy-efficient state. When the stall time ends, the processor previous frequency and voltage are restored and the transaction is restarted.

The rationale behind our technique is that the energy spent during transactional stall can be reduced without degrading performance significantly, considering that such period is considered idle time. The DVFS technique, however, imposes an extra overhead to enter and exit the low-power mode, which is approximately 50 cycles in our simulation platform.

Figure 5 shows the results we have obtained with the DVFS-based technique for the BACKOFF and DELAY policies. Due to space limitations, we only show the results for the 8-core configuration. As usual, the values are normalized wrt SUICIDE, since it was the best policy according to Section 4.2. Notice that the bars are representing EDP now (smaller values are better).

As can be seen in Figure 5, the proposed DVFS-based

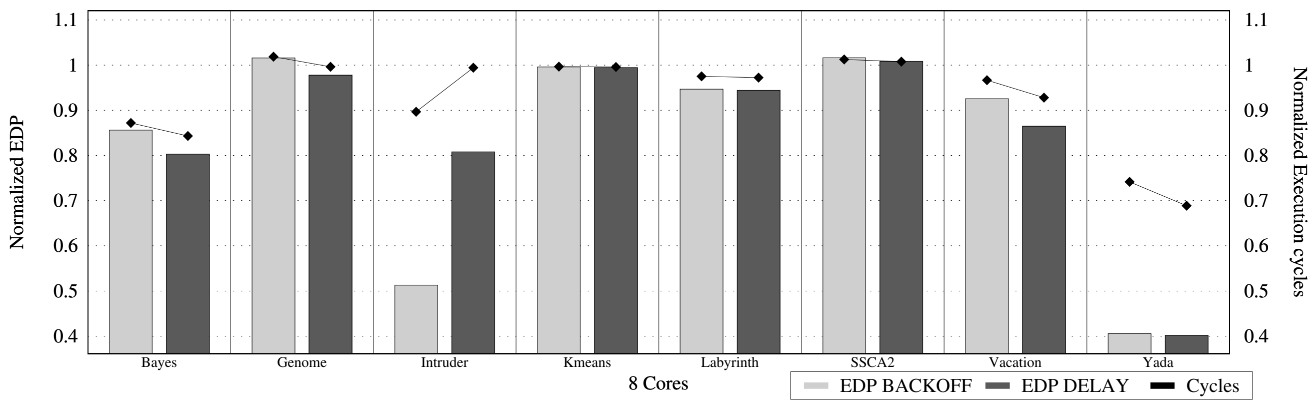

Figure 5. EDP and execution time for `BACKOFF` **and** `DELAY` **using** ETL_l**. The values are normalized wrt** ETL_l `SUICIDE` **(8 cores).**

mechanism provided significant enhancements. Virtually all applications took advantage of the reduction in operation frequency during transactional stall, especially `Intruder` and `Yada`, as predicted. In general, the average EDP improvement across the applications is 16.6% for `BACKOFF`, and 15% for `DELAY`. An average performance increase of 6.5% (`BACKOFF`) and 7.2% (`DELAY`) is also achieved with our technique. In particular, `Yada` exhibited an excellent improvement and benefited the most from the DVFS strategy, reaching a reduction of approximately 59% in EDP.

6. Related Work

Although power-performance tradeoffs have been investigated in the context of parallel applications before (e.g. [13, 19]), the literature dealing with energy-aware synchronization methods is more scarce. Li et al. [20] propose the thrifty barrier, a hardware-software mechanism aimed at reducing energy consumption in applications suffering from barrier synchronization imbalance. When a thread arrives at a barrier, a stall time is estimated and the processor is appropriately forced into a low-power mode, effectively reducing the energy waste in barrier spin-loops. Our integration of DVFS with contention managers resembles the thrifty barrier in the sense that the operation frequency of the processor is reduced after a transaction is aborted, thereby saving energy while it waits to be restarted.

There has been an increasing interest in the energy efficiency of transactional memory recently, even though the majority of the publications focus on hardware transactional memory. The first work we are aware of that investigates energy consumption in HTM systems is due to Moreshet et al. [24]. Although their results suggest that HTM has an advantage over locks in the absence of contention, the experimental evaluation consists of only microbenchmarks. Ferri et al. [9, 10] investigate energy-efficiency and performance of different HTM configurations tailored to high-end embedded systems. Since HTM systems are susceptible to cache overflows, the authors consider alternative cache architectures and analyze their impact on energy consumption.

A detailed characterization of performance and energy is presented by Gaona-Ramirez et al. [11] for two representative HTM systems: LogTM-SE [29] and TCC [4]. The simulator used by the authors estimates the energy consumed by the on-chip caches and the interconnection network. The characterization is conducted with the STAMP benchmark, and no attempt is made to optimize energy or performance. Sanyal et al. [26] offer a mechanism based on clock gating to save energy in a TCC-based HTM [4]. When a transaction is aborted, the corresponding processor is dynamically turned off. Experimental results reveal an average 19% reduction in energy consumption and a 4% speedup. Likewise, Hughes and Li [15] propose two policies to increase energy efficiency in HTM systems: a DVFS technique to save energy during transactional stalls, and a heuristic based on conflict probability to reschedule transactions and clock gate aborted processors. Our work is in tandem with the latter two papers in as much as we also capitalize on reducing the energy spent on aborted transactions and transactional stalls.

We are not aware of any other study on energy characterization of STM other than our previous works [2, 17, 18]. In this paper we cover a much broader spectrum of the STM design space, considering a new algorithm variation (encounter-time locking, lazy versioning) and two additional contention management policies (suicide and delay). We have also replaced our implementation infrastructure, which was based on the TL2 version provided with the STAMP benchmark [23], with TinySTM 1.0.3 [8].

7. Conclusions

In this paper we performed a thorough evaluation of energy and performance of the most important lock-based STM variations. Using the STAMP benchmark suite, our evaluation indicated that the ETL_l algorithm, along with the `SUICIDE` resolution policy, provided the best overall energy-performance tradeoff. By exploiting the slack available in applications showing high contention, our DVFS-enhanced resolution policies achieved an EDP improvement of 16% across the STAMP workloads when compared to the best previously analyzed policy (`SUICIDE`).

8 Acknowledgment

This work is supported by FAPESP and CNPq.

References

[1] A. Agarwal and M. Cherian. Adaptive backoff synchronization techniques. In *Proc. of the 16th ISCA*, pages 396–406, 1989.

[2] A. Baldassin, F. Klein, G. Araujo, R. Azevedo, and P. Centoducatte. Characterizing the energy consumption of software transactional memory. *IEEE Computer Architecture Letters*, 8(2):56–59, 2009.

[3] C. Cascaval, C. Blundell, M. Michael, H. W. Cain, P. Wu, S. Chiras, and S. Chatterjee. Software transactional memory: Why is it only a research toy? *Communications of the ACM*, 51(11):40–46, Nov. 2008.

[4] H. Chafi, J. Casper, B. D. Carlstrom, A. McDonald, C. C. Minh, W. Baek, C. Kozyrakis, and K. Olukotun. A scalable, non-blocking approach to transactional memory. In *Proc. of the 13th HPCA*, pages 97–108, Feb. 2007.

[5] D. Dice, O. Shalev, and N. Shavit. Transactional Locking II. In *20th International Symposium on Distributed Computing*, pages 194–208, Sept. 2006.

[6] D. Dice and N. Shavit. Understanding tradeoffs in software transactional memory. In *Proc. of the CGO*, pages 21–33, Mar. 2007.

[7] A. Dragojevic, P. Felber, V. Gramoli, and R. Guerraoui. Why STM can be more than a research toy. *Communications of the ACM*, 54(4):70–77, Apr. 2011.

[8] P. Felber, C. Fetzer, and T. Riegel. Dynamic performance tuning of word-based software transactional memory. In *Proc. of the 13th PPoPP*, pages 237–246, Feb. 2008.

[9] 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*, pages 435–440, May 2008.

[10] C. Ferri, S. Wood, T. Moreshet, R. I. Bahar, and M. Herlihy. Embedded-TM: Energy and complexity-effective hardware transactional memory for embedded multicore systems. *Journal of Parallel and Distributed Computing*, 70(10):1042–1052, Oct. 2010.

[11] E. Gaona-Ramirez, R. Titos-Gil, J. Fernandez, and M. E. Acacio. Characterizing energy consumption in hardware transactional memory systems. In *Proc. of the 22nd SBAC-PAD*, pages 9–16, Oct. 2010.

[12] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sept. 1996.

[13] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of both latency and throughput. In *Proc. of the ICCD*, pages 236–243, Oct. 2004.

[14] T. Harris, J. Larus, and R. Rajwar. *Transactional Memory*. Morgan & Claypool Publishers, 2 edition, June 2010.

[15] C. Hughes and T. Li. Optimizing throughput/power tradeoffs in hardware transactional memory using DVFS and intelligent scheduling. In *Proc. of the 25th ICS*, pages 141–150, 2011.

[16] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan & Claypool Publishers, 2008.

[17] F. Klein, A. Baldassin, G. Araujo, P. Centoducatte, and R. Azevedo. On the energy-efficiency of software transactional memory. In *Proc. of the 22nd SBCCI*, pages 1–6, Sept. 2009.

[18] F. Klein, A. Baldassin, J. Moreira, P. Centoducatte, S. Rigo, and R. Azevedo. STM versus lock-based systems: An energy consumption perspective. In *Proc. of the 2010 ISLPED*, pages 431–436, Oct. 2010.

[19] J. Li and J. F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *Proc. of the 12th HPCA*, pages 77–87, Feb. 2006.

[20] J. Li, J. F. Martinez, and M. C. Huang. The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors. In *Proc. of the 10th HPCA*, pages 14 – 23, Feb. 2004.

[21] M. Loghi, L. Benini, and M. Poncino. Power macromodeling of MPSoC message passing primitives. *ACM Transactions on Embedded Computing Systems*, 6(4):31, Sept. 2007.

[22] M. Loghi, M. Poncino, and L. Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *Proc. of the 14th GLSVLSI*, pages 410–406, Apr. 2004.

[23] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. In *Proc. of the IISWC*, pages 35–46, Sept. 2008.

[24] T. Moreshet, R. I. Bahar, and M. Herlihy. Energy reduction in multiprocessor systems using transactional memory. In *Proc. of the 2005 ISLPED*, pages 331–334, Aug. 2005.

[25] B. Saha, A.-R. Adl-Tabatabai, R. L. Hudson, C. C. Minh, and B. Hertzberg. McRT-STM: A high performance software transactional memory system for a multi-core runtime. In *Proc. of the 11th PPoPP*, pages 187–197, Mar. 2006.

[26] S. Sanyal, S. Roy, A. Cristal, O. S. Unsal, and M. Valero. Clock gate on abort: Towards energy-efficient hardware transactional memory. In *Proc. of the IPDPS*, pages 1–8, May 2009.

[27] N. Shavit and D. Touitou. Software transactional memory. In *Proc. of the 14th PODC*, pages 204–213, Aug. 1995.

[28] H. Sutter and J. Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, Sept. 2005.

[29] L. Yen, J. Bobba, M. R. Marty, K. E. Moore, H. Volos, M. D. Hill, M. M. Swift, and D. A. Wood. LogTM-SE: Decoupling hardware transactional memory from caches. In *Proc. of the 13th HPCA*, pages 261–272, Feb. 2007.