Green-CM: Energy efficient contention management for Transactional Memory

Shady Issa, Paolo Romano
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
{shadi.issa,paolo.romano}@tecnico.ulisboa.pt

Mats Brorsson
KTH, Royal Institute of Technology
Stockholm, Sweden
matsbror@kth.se

***Abstract*—Transactional memory (TM) is emerging as an attractive synchronization mechanism for concurrent computing. In this work we aim at filling a relevant gap in the TM literature, by investigating the issue of energy efficiency for one crucial building block of TM systems: contention management.**

Green-CM, the solution proposed in this paper, is the first contention management scheme explicitly designed to jointly optimize both performance and energy consumption. To this end Green-TM combines three key mechanisms: i) it leverages on a novel asymmetric design, which combines different back-off policies in order to take advantage of dynamic frequency and voltage scaling; ii) it introduces an energy efficient design of the back-off mechanism, which combines spin-based and sleep-based implementations; iii) it makes extensive use of self-tuning mechanisms to pursue optimal efficiency across highly heterogeneous workloads.

We evaluate Green-CM from both the energy and performance perspectives, and show that it can achieve enhanced efficiency by up to 2.35 times with respect to state of the art contention managers, with an average gain of more than 60% when using 64 threads.

I. Introduction

Transactional Memory (TM) is an emerging paradigm for parallel programming that represents an attractive alternative to traditional lock-based synchronization techniques. The concept was introduced 20 years ago [1] and was object of a thorough research during the past decade. Nowadays, two of the main processor manufacturers, Intel and IBM, provide hardware support for transactional memory in their latest generation CPUs [2], [3].

The basic idea behind transactional memory is to arrange the code into blocks, called transactions, that are to be executed atomically. The transactional memory is responsible for management of data races between different transactions in a transparent way, ensuring isolation and atomicity.

Most TM implementations take a speculative approach and run transactions in a lock-free, optimistic fashion. This makes them prone to incur high abort rates in presence of high contention workloads, which can lead to severe degradation of both performance and energy efficiency. It is the responsibility of the *Contention Manager (CM)* module to reduce the detrimental effects of contention, by deciding which transactions should be aborted in case of a conflict, and when to restart an aborted transaction.

This work was supported by national funds through Fundação para a Ciência e Tecnologia (FCT), via the projects UID/CEC/50021/2013 and EXPL/EEI-ESS/0361/2013

The literature in the area of CM is quite prolific [4]–[7], yet most existing CM solutions are designed to optimize solely performance. However, up to date very few works have investigated CM designs aimed at maximizing energy efficiency [8], [9], an aspect that is increasingly relevant for a wide range of systems, from sensors or mobile nodes powered by batteries, to data centers, whose scalability is nowadays constrained by their energy costs [10].

This work aims at filling this relevant gap in the CM literature by proposing Green-CM, a contention manager that optimizes energy-efficiency of TM applications via three key mechanisms:

1) Green-CM introduces an energy efficient implementation of one fundamental building block at the basis of most existing CMs and that can have a strong impact on energy consumption: the `back-off` primitive that is used whenever the CM decides to block a conflicting transaction for some period of time. The proposed solution uses a hybrid approach that alternates between two implementations, based, respectively, on spinning and timer-interrupts. The latter allows for effectively reducing energy consumption, but incurs long latencies due to the need for invoking a system call; spinning has opposite advantages and drawbacks: it is accurate also for very short backing off periods, but suffers of high energy costs. By leveraging on both implementations in synergy, Green-CM aims to achieve the best of both worlds, namely low energy consumption and high accuracy.
2) Green-CM introduces an innovative, *Asymmetric* CM (ACM) policy that aims to take advantage of DVFS (Dynamic Voltage and Frequency Scaling) [11], an architectural feature that is widely employed in modern processors [12], [13] in order to enhance their energy efficiency. DVFS allows various cores of a processor (and/or various processors in a multi-socket system) to adjust dynamically the voltages and frequencies at which they operate: on the one hand, this allows for reducing the energy consumed by idle cores; on the other hand, it allows for increasing the frequency of active cores, as long as the number of idle cores is large enough to ensure that the global thermal envelope remains within acceptable margins. ACM is based on the key idea of promoting the exploitation of DVFS capabilities via

the usage of asymmetric back-off policies. More in detail, ACM combines aggressive and conservative (i.e., linearly vs exponentially increasing) back-off policies, in order to promote the dynamic creation, at medium/high contention scenarios, of two sets of threads: 1) threads that are likely to be backing off, allowing the corresponding processor to enter deep sleep states, and 2) threads that spend most of their time executing transactions, and which can run at higher frequencies thanks to DVFS.
3) Green-CM makes extensive use of lightweight reinforcement learning techniques to dynamically adapt its internal parameters and specifically: a) determining automatically in which scenarios spin vs timer-interrupts based implementations should be used, and b) what degree of asymmetry should be used when determining the Contention Management back-off policies. We propose and evaluate different variants of gradient descent based controllers to tackle each of these two problems, both individually and in conjunction.

We conduct an extensive experimental study, based on standard TM benchmarks (STAMP [14] and STM-Bench7 [15]) and recent TM-based implementations of real-life applications (Memcached [16]), and considering 6 alternative CM implementations, which we evaluate from the twofold perspective of performance and energy consumption. The experimental data shows that Green-CM achieves average gains of 25% when considering joint energy-performance metrics, i.e., energy-delay product (EDP), with peak gains that extend up to 2.35x lower EDP. We also assess the effectiveness of the proposed self-tuning mechanisms, which, we show, achieve performance that are on average within 15% from, and sometimes even superior to, the best, manually identified, static solutions.

The remainder of this paper is structured as follows. Section II discusses related work. Section III details the design of Green-CM which is then evaluated in Section IV. Finally, Section V concludes the paper.

II. Related work

Most of the literature of TM is concerned with optimizing TM performance [17]–[20], but the issue of energy efficiency is much less explored. Indeed, the few existing works on TM that tried to optimize both energy and performance were mainly revolving around energy efficient hardware implementations of TM [8], [21]–[23].

Sanyal et al. [21] pursue energy efficiency in hardware transactional memory by clock gating processors upon abort of a transaction. Baldassin et al. [9] adopted a similar idea, although implemented at the software level and integrated with the CM module: using DVFS to lower the frequency of cores upon abort and during the (exponential) back-off phase. Their study was limited to 8 threads only using a simulator that has a very low cost for entering a lower frequency mode, which make this solution largely sub-optimal in practice.

Two studies were performed on energy consumption of TM. Rughetti [24], [25] studied the performance and energy trade-offs of various algorithms; the study showed the necessity of adaptability within TM to minimize data contention as it is the main source of energy consumption. Diegues et al. [26] evaluated the performance and energy efficiency of different TM implementations, including hardware-based (HTM), software-based (STM) and hybrid (HyTM) using a large number of popular benchmarks. The results of this study highlight that the choice of the right TM implementation is strongly workload dependant. The Green-CM algorithm has been designed to operate both with STM and HTM, as it does requires no information on the set of items accessed by aborting transactions — an information not available when using HTM [27].

Wamhoff et al. [28] performed an extensive study of DVFS in Intel and AMD processors. This work characterized the behavior of frequency scaling on both architectures describing how it can be utilized either automatically (hardware triggered) or manually (software enabled). This work showed also how to exploit DVFS in order to enhance the performance of an STM called FastLane [29]. This STM has a master thread that runs at a boosted frequency and whose transactions never abort. The downside is that it forces all other threads to run at lower frequencies, which yields performance gains only at low thread counts

A large body of research has been devoted to investigate CM algorithms [4]–[7] ranging from very simple policies, such as aggressive CM in which the victim transaction is always aborted, to more complex algorithms that use different heuristics for determining the back-off time upon aborts (e.g., linear vs exponential), or that take into account the amount of work done by the contending transactions. None of these CM policies were evaluated from the perspective of energy efficiency, and we fill this gap in Section IV-B, where we compare Green-CM with 6 state of the art CM algorithms (and report significant gains).

Finally, our work is related to the studies that have analyzed the energy efficiency of alternative implementations of locking schemes (whereas we focus on TM). For instance, it was shown [30], that a hybrid combination of busy-waiting and sleeping is the optimal solution in terms of energy-delay product to implement a mutex semaphore. This work, however, leaves unsolved the issue of determining when to use spinning or sleeping. In Green-CM we use a similar concept in the implementation of the primitive used by threads to back-off for the completion of their back-off phase: we opt for either spinning or sleeping depending on the duration of the requested backing off phase. There are however at least two fundamental differences. Our technique is employed for a different primitive, i.e., a back-off and not a mutex, for which the duration of the backing off time period is a-priori known: we can hence exploit this information to make an informed decision on whether to use spin-based or

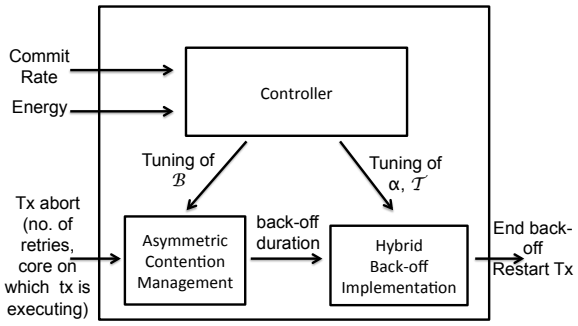


Figure 1: Architecture of Green-CM

sleep-based implementations. Further, Green-CM solves the problem of automating the decision of when to use spinning or sleeping via an on-line self-tuning technique.

III. GREEN-CM

This section is devoted to describing Green-CM. We start by illustrating its high level architecture in Figure 1, in which we can identify three main components: the Asymmetric Contention Manager (ACM), a hybrid implementation of the back-off primitive, and the Controller.

Upon the abort event of a transaction, the first component to be triggered is the Asymmetric Contention Manager. This module is in charge of determining the duration of the back-off phase for the transaction and, as we will discuss more in detail in Section III-B, it determines whether to use an aggressive (i.e., linear) or a conservative (i.e., exponential) back-off depending on two factors: i) the CPU core on which the corresponding thread is running, and ii) the boosting degree (noted $\mathcal{B}$ in the following), i.e., a tunable parameter (dynamically configured by the Controller) that allows for controlling how many threads should use each of the two available back-off policies.

The duration of the back-off phase is provided as input to a hybrid implementation of the back-off primitive, which is described in Section III-A. The hybrid back-off primitive uses either a spin-based or a sleep-based implementation depending on two factors: i) the duration of the the back-off phase, and ii) the value of two parameters, noted α and $\mathcal{T}$, which represent, respectively, the number of spin cycles executed in a time unit, and the minimum back-off duration for using a sleep-based approach. Analogously to the case of $\mathcal{B}$, the tuning of α and $\mathcal{T}$ is delegated to the Controller.

The Controller module gathers periodic measurements on the commit rate and energy consumption over the last time window, and uses this information to implement a lightweight, on-line self-tuning scheme. We used a sampling period of 1 msec, this sampling period ensures that energy measurements on our target architecture are reliable. We shall discuss the Controller module in Section III-C.

A. Hybrid Back-off Implementation

The goal of contention managers is to reduce the detrimental effects of contention on the efficiency of Transactional Memory. This objective is pursued by reducing the likelihood that threads executing conflicting transactions execute at the same time. One of the most common techniques to perform this is to force threads to back-off for a certain period of time when they encounter contention, before restarting the aborted transaction. The duration of the back-off period is determined by the back-off policy employed by the CM (e.g., exponential back-off) and can be expressed either in terms of processor cycles (e.g., number of iterations during which to spin) or in time units (e.g., nanoseconds).

In principle there exist two ways of implementing a back-off mechanism: i) spinning, i.e., busy-waiting, in an empty loop, or possibly by invoking at each iteration "pipeline-friendly" assembly instructions, such as `pause` in x86 architectures; ii) sleeping by invoking the `sleep` system call. The two methods exhibit clear trade-offs for what concerns performance and energy consumption. Busy waiting has very fine granularity, but, from the perspective of energy consumption, it is strongly inefficient. Sleeping, on the other hand, achieves low energy consumption but provides coarse granularity (various tens of microseconds in recent architectures/OSs [28])), which can have a detrimental impact on the effectiveness of the CM policy.

Since existing CM have focused on optimizing performance, basically neglecting the issue of energy efficiency, existing implementations rely solely on spin-based approaches — which, do not suffer of the accuracy issues of sleep-based approaches for short back-off periods. The drawback is that they miss the opportunity of reducing energy consumption when back-off times are sufficiently large to be effectively supported using sleep-based implementations.

The hybrid approach that we propose in this paper, and whose pseudo-code is reported in Algorithm 1, is based on a simple, yet effective idea: using a sleep-based or a spin-based implementation depending on the duration of the back-off period. The intuition is that spin-based implementations are ideal for "sufficiently short" back-off periods, whereas sleep-based ones work best for "sufficiently long" back-off period.

Despite the idea may at first glance appear relatively straightforward, it does hide two non-trivial, and closely intertwined, issues:

1) Spin-based and sleep-based implementations operate using different time scales: the latter expresses the back-off duration in real-time units (e.g., nanoseconds), whereas the latter uses spin cycles, or, equivalently, processor cycles. In order to hide both implementations under the same interface, and use the two transparently, it is necessary to reconcile their time scales, by identifying a conversion factor, which we note α, in order to map processor cycles to real-time (or

```
1 function void back-off( int waitCycles)
2 if waitCycles/α ≤ T then
3 |   while waitCycles ≠ 0 do
4 |   |   waitCycles--;
5 |   end
6 else
7 |   sleep(waitCycles/α - min_sleep );
8 end
```

Algorithm 1: Pseudo-code for the hybrid back-off mechanism.

vice versa).

The issue here is that, in modern processors, the number of spin (or processor) cycles executed within a time unit can vary significantly depending on the impact that the workload's characteristics have on architectural aspects like DVFS, pipeline and caching.

2) What is the minimum value of the back-off duration, which we denote by $\mathcal{T}$, for which sleep-based implementations are more efficient (from a joint energy-performance perspective) than spin based ones? Ideally, the value of $\mathcal{T}$ should be set to the minimum back-off duration for which the gains in terms of energy consumption achieved by sleeping outweigh the performance losses due to its lower accuracy.

One additional noteworthy aspect is that, in order to enhance the accuracy of the sleep-based implementation, in line 7, we adjust the requested sleep duration before invoking the sleep system call. More in detail, we subtract from the target back-off duration (i.e., $\frac{\text{waitCycles}}{\alpha}$) the minimum latency for executing a sleep system call (i.e., for executing `sleep(0)`), which we noted min_sleep in the pseudo-code. In fact, whenever the sleep system call is called with x as input parameter, the actual latency for the execution of sleep is equal to min_sleep+x+err, where err is an error factor depending on the actual sleep accuracy (e.g., hardware timer resolution). This latency can be easily measured experimentally, and by taking it into account, one can significantly enhance accuracy when the back-off duration is of the same order of min_sleep.

As we will discuss more in detail in Section III-C, the identification of the correct value of the parameters α and $\mathcal{T}$ plays a crucial role in determining the efficiency of the CM scheme. We address this problem via a light-weight, on-line self-tuning mechanism, which we also detail in Section III-C.

B. Asymmetric Contention Management

Ideally, a CM may take advantage of the DVFS capabilities provided by modern CPUs by scaling down the frequency of a core whenever a transaction has to be aborted and backed off, and scaling up the frequency of that as soon as the back-off period completes. If a sufficient number of transactions are in the back-off state, the CM could explicitly request to boost the frequency of some cores, provided that the thermal envelope of the corresponding CPU is within the safety margin. Unfortunately, controlling the dynamic frequency scaling mechanism requires issuing system calls, which induce prohibitive costs [28] and would largely outweigh the gains achievable thanks to DVFS.

The idea at the basis of ACM is to approximate such an ideal, yet impractical CM policy, by using a lightweight design that aims at favouring the spontaneous activation of hardware-controlled DVFS mechanisms. Modern CPUs, in fact, identify in an automatic fashion the opportunity to boost the frequency of subset of cores, whenever a sufficient number of cores in the same CPU have entered a sleep state and are executing below the nominal frequency.

In order to make the contention management scheme DVFS-aware we exploit a simple idea, which is, to the best of our knowledge, still unexplored in the literature, i.e., we adopt an asymmetric approach that divides threads into two categories: threads active on cores to be boosted, which we call *boosted* threads, and threads executing on cores to be pushed towards lower operating frequencies. This can be achieved by letting the contention management treat these two categories in an asymmetric fashion: the threads to be boosted will be backed off for linearly increasing periods, while the other category will be backed off for exponentially increasing periods.

Under such arrangement, and considering a hybrid implementation of the back-off mechanism, like the one described in the previous section, boosted threads are likely to be either executing transactions or spinning, as they will most likely back-off for short periods. The other threads, on the other hand, tend to back-off for longer periods. Hence, they are more likely to use the sleep-based back-off implementation and to have their cores enter deeper sleep states.

Note that, in order to favour the activation of the hardware-controlled DVFS mechanism, the selection of which threads should be boosted has to be made in an architecture-aware fashion. In fact, in order to create the preconditions for DVFS to accelerate the frequency of the core on which a boosted thread is executing, the number of boosted threads active in each CPU should not exceed the maximum number of cores $\mathcal{M}$ that can simultaneously execute at frequencies higher than the nominal ones. For instance, in AMD Opteron CPUs such as the ones that we use in our experimental evaluation, at most two cores out of the 8 cores available in each processor can enter the boosted state, when the other 6 are sleeping. This architecture dependant parameter is taken into account by the ACM, which scatters the $\mathcal{B}$ boosted threads across the available CPUs, assigning at most $\mathcal{M}$ boosted threads per CPU.

As we will further discuss in Section III-C, the decision of how many boosted threads to use, which we call boosting

degree ($\mathcal{B}$) is non-trivial, as the optimal tuning is in general workload-dependant. As already mentioned, we delegate the task of automating the tuning of $\mathcal{B}$ to the Controller module, which we describe next.

C. Controller

As already mentioned, the Controller relies on on-line self-tuning techniques in order to identify the values of the parameters $\alpha, \mathcal{T}$ and $\mathcal{B}$ that yield maximum energy-efficiency. Before discussing the design of the proposed self-tuning mechanism, though, we present experimental data aimed at highlighting the relevance of tuning each of these three parameters. The analysis of this data will also allow us to obtain some important insights that have driven the design of the self-tuning mechanisms employed by the Controller.

- **The need for self-tuning.** Let us start by performing a sensitivity study to the tuning of α and $\mathcal{T}$. To this end, we consider two benchmarks of the STAMP benchmark suite [14], which generate workloads with distinct characteristics: Intruder generates relatively long transactions that have a high contention probability; transactions in Kmeans, conversely, are relatively short and less prone to aborts. The results reported in this section, and in the remainder of the paper, were obtained running with 64 threads on a machine equipped with an AMD Opteron 6272 CPU running linux 3.13 and equipped with 32 GB of RAM.

In Figure 2 we set the number of active threads to 64, and report the EDP obtained when varying α from 100 to 10^7 with $\mathcal{T} = min\_sleep$ (which we recall is the minimum sleep granularity), normalized with respect to the EDP obtained when using the optimal values for α and $\mathcal{T}$, identified via an exhaustive off-line search. The rationale for setting the threshold $\mathcal{T} = min\_sleep$ is that sleep is expected to pay off only if the sleep time is larger than the time it takes to execute the sleep system call. In fact, for lower sleep times, the CPU is going anyway to be occupied for min_sleep time units and, hence, consume more CPU cycles than a spin-based implementation.

The plot allows us to draw two interesting conclusions. On the one hand, we observe that the optimal value of α for the two benchmarks is significantly different, being equal to 5K for Intruder and to 250K for Kmeans — a difference of two orders of magnitude. Also, if one uses the optimal setting of α for Kmeans, resp. Intruder, with Intruder, resp. Kmeans, the EDP is more than $2\times$, resp. $5\times$ higher. These data clearly highlight the relevance of appropriately tuning this parameter.

On the other hand, by setting statically $\mathcal{T} = min\_sleep$ (and properly tuning α) we obtain an EDP that is very close to (i.e., at most 5% higher than) the EDP obtained by using any alternative value of $\mathcal{T}$. This is true despite the fact that the two considered benchmarks have radically different workload characteristics. In fact, we have experimentally verified across the entire set of benchmarks considered in this paper (whose full list is provided in Section IV-B) that the quality of the solution obtained by using this simple heuristic is always very close to the optimal one. In the light of these considerations, and in order to maximize the convergence speed of the self-tuning mechanisms employed by the Controller, we opt for setting setting $\mathcal{T} = min\_sleep$ and consequently reducing the dimensionality of the optimization problem.

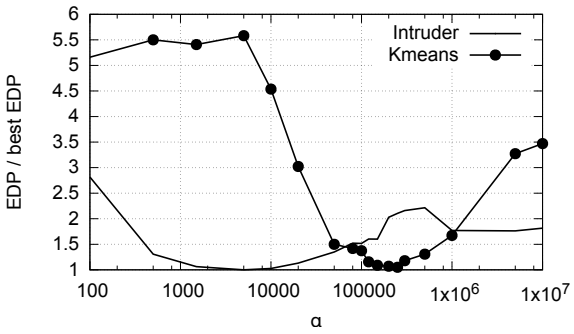


Figure 2: EDP of different static configurations of α with $\mathcal{T} = min\_sleep$, normalized with respect to the EDP of the best, off-line identified, configuration for α and $\mathcal{T}$.

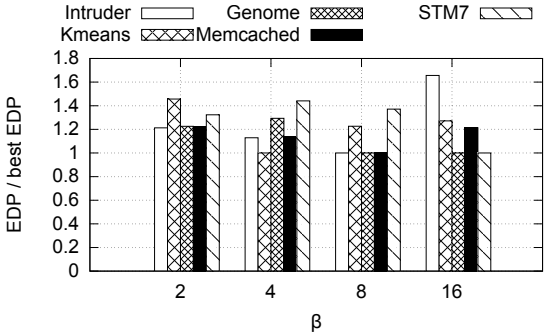


Figure 3: EDP for different static values of $\mathcal{B}$ normalized to the EDP obtained using the best setting of $\mathcal{B}$.

Next, we analyze the effects of using different values for the $\mathcal{B}$ parameter. To this end we consider 5 popular TM benchmarks, namely Intruder, Kmeans, Genome, Memcached and STMBench7 [1], which we run using 64 threads in total. We treat $\mathcal{B}$ as the independent parameter, and set α and $\mathcal{T}$ to their optimal, off-line determined, values. In the machine used in our study, at most 16 cores can operate above the nominal frequencies (when the remaining 48 are in sleep state). Hence, we accordingly set the maximum value of $\mathcal{B}$ to 16. In Figure 3 we report the EDP obtained using different static values of $\mathcal{B}$ normalized to the EDP obtained using the best setting of $\mathcal{B}$. By the plot we get that the optimal setting of $\mathcal{B}$ varies significantly with the considered benchmark: for Intruder, for instance, EDP degrades by around 70% if $\mathcal{B} = 16$, since the contention level generated

[1]Since the key performance indicator for Memcached and STMBench7 is the commit rate, for these benchmarks we compute EDP as the ratio between energy consumed and commit rate.

by having so many threads using an aggressive, linear back-off policy grows unacceptably large; the opposite is true for STMBench7, for which the optimal $\mathcal{B}$'s value is 16, and using lower values can yield up to 40% increase of EDP.

- **Design of the self-tuning scheme.** As already discussed, by setting $\mathcal{T} = min\_sleep$ we reduce the dimensionality of the on-line optimization problem that the controller has to tackle, which is limited to identify the optimal tuning of α and $\mathcal{B}$. The Controller tackles this problem by employing a lightweight, model-free on-line search approach [31], which identifies the optimal values of the target parameters by exploring alternative points in the $\alpha \times \mathcal{B}$ space. There are two main design decisions that are the basis of any model-free on-line optimization algorithm:

– How to explore the search space.
– The exploration vs exploitation dilemma, i.e., when to stop exploring and start exploiting the available knowledge.

In the design of the Controller we have considered alternative policies for tackling each of these two problems. We describe each of them in the following, and postpone their evaluation to Section IV-A.

How to explore the search space. The exploration policies that we consider represent variants of the classic hill-climbing algorithm, which, we recall, operates as follows: at each iteration the neighbours of the current configuration are tested and the one that maximizes the target metric is set as the new configuration for the next iteration. The basic hill-climbing algorithm suffers of three main problems, which can be addressed by considering several additional mechanisms, described in the following:

∗ *local minima*: due to the localized nature of its search policy, hill-climbing is well known to be prone to get stuck in local minima. A commonly employed solution to this problem is to force random jumps with a fixed, small probability. This variant is noted *jmpX*, where X is the jump probability.

∗ *curse of dimensionality*: the number of neighbours for a configuration grows exponentially with the dimensionality of the search space [31]. In order to circumvent this issue we consider two alternative exploration policies: i) We treat the two dimensions α and $\mathcal{B}$ as tunable in a completely independent fashion, and run two hill-climbing based optimizers, each targeting a different dimension, in parallel and without any synchronization. We note this policy as *independent*. ii) We subdivide the exploration in phases, and during each phase we optimize along exclusively one dimension, changing the target dimension whenever a phase ends. We note this policy as *alternate*.

∗ *slow convergence in large domains*: the hill-climbing can converge after an unacceptably high number of explorations if the domain that is being explored spans a broad range of values and the granularity used to identify the neighbours of the current configuration (also called, exploration step) is too small. On the other hand, using overly large exploration steps increases convergence speed, but can have a detrimental effect on the quality of the identified solution. In the problem at hand, as already noted, α spans a very broad domain (from a few hundreds to about one million). To cope with this issue, when moving along the α dimension the controller uses an adaptive exploration step: it starts by adopting a large (125K) exploration step, which it halves whenever the direction of exploration along the α dimension is inverted (because a suboptimal value is found) till a minimum value for the exploration step (1K) is reached. We use instead a fixed exploration step equal to one when moving along the $\mathcal{T}$ dimension.

Exploring vs exploiting. The hill-climbing approach never stops exploring, i.e., when it identifies a minimum, it keeps on oscillating around it for ever. This has the advantage of making it prone to react to changes in the function being optimized (e.g., imputable for instance to shifts of the application's workload). On the down side, if the function is stable, moving away from the (local) optimum, and re-exploring a configuration that is known to be suboptimal, means incurring a certain penalty. A simple heuristic that can be used to tackle this problem is to detect subsequent oscillations around the current local minimum, and stop explorations. We call such a variant `stabilizing`.

IV. Evaluation

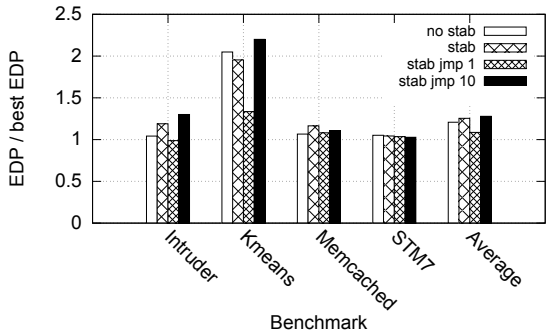
This section aims to quantitatively evaluate Green-CM from a twofold perspective. We start by assessing the effectiveness of the various self-tuning considered in Section III-C. Then, we evaluate the performance, energy consumption, and EDP of Green-CM with respect to state of art CM solutions.

As already mentioned, Green-CM was designed to work with both hardware and software based TM implementations. In this study, we select as reference TM implementation TinySTM [18], a software-based TM that has been shown to excel in a wide range of workloads [26]. TinySTM comes with different contention managers including exponential back-off with busy waiting. We consider a set of 4 well-known TM benchmarks: Intruder-high and Kmeans-high, from the STAMP suite [14], and Memcached [16] and STMBench7 [15], both generating 50% read and 50% write transactions. The results are the average of at least 5 runs.

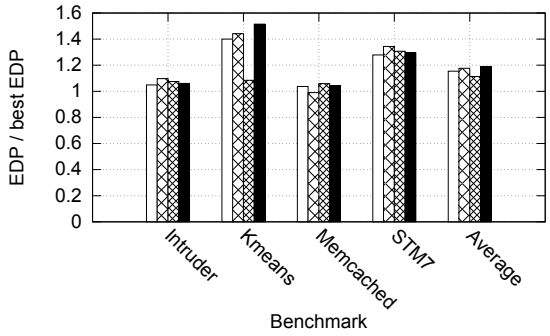
A. Tuning Strategies

We start by evaluating the effectiveness of the self-tuning algorithms when operating on each dimension of the search space $\alpha \times \mathcal{B}$ individually. This preliminary analysis will allow to circumscribe the combinations of self-tuning algorithms that will be evaluated to optimize α and $\mathcal{B}$ in conjunction.

Individual tuning of α and $\mathcal{B}$. For the individual optimization of α and $\mathcal{B}$ we fix the number of active threads at 64



(a) Tuning α



(b) Tuning $\mathcal{B}$

Figure 4: Normalized EDP across different benchmarks using various strategies to self-tune individually α and $\mathcal{B}$.

and consider four alternative self-tuning strategies: *nostab*, a non-stabilizing policy that does not perform probabilistic jumps; *stab*, a stabilizing policy that does not perform probabilistic jumps; *stab jmp1* and *stab jmp10*, wo stabilizing policies that perform random jumps with probability, resp., 1% and 10%. When evaluating the self-tuning of α, we set $\mathcal{B} = 0$. When self-tuning $\mathcal{B}$, we set α to the corresponding optimal, off-line found value for $\mathcal{B}$.

Figure 4(a) shows the EDP across different benchmarks when using the different tuning strategies for α, normalized to the EDP obtained when using the (per-benchmark) optimal, off-line found value of α. By the plot, it can be deduced that the stabilizing tuner that performs random jumps with 1% probability outperforms all others, achieving an EDP that is only 8% larger than the optimal static solution (identified via an exhaustive off-line exploration).The reason behind this is the fact that stabilization minimizes the cost paid oscillating around a minimum. Also, a small jump probability is sufficient to allow the tuner to escape from local minima, without excessively hindering performance with overly frequent random explorations.

Analogous considerations can be drawn by analyzing Figure 4(b), which shows the results for the same study conducted on the tuner of $\mathcal{B}$. The considered strategies behave similarly to the previous case, although the relative differences between them are smaller. We argue that this is a consequence of the fact that identifying the optimal tuning of $\mathcal{B}$ is a relatively easier problem, given that the corresponding domain is much smaller than the one of α.

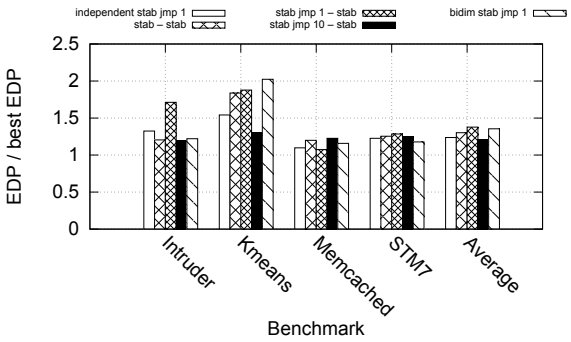


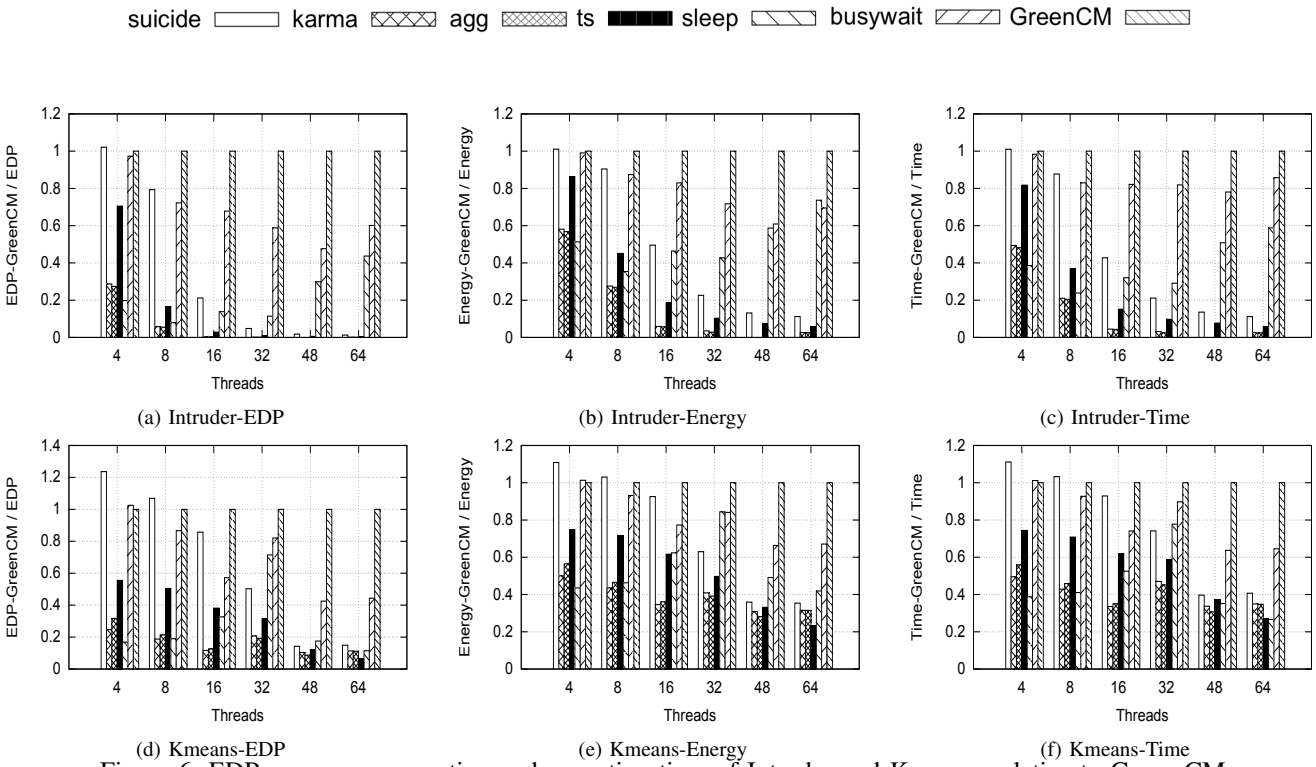
Figure 5: Normalized EDP across different benchmarks using different strategies for coupling the two tuners.

Joint tuning of α and $\mathcal{B}$. Next, we consider different strategies for tuning α and $\mathcal{B}$ in conjunction. We consider the following tuning policies: i) *independent stab jmp1*, two independent tuners, using stabilization and random jumps with 1% probability. ii) *bidim stab jmp1*, same as above, except that a single learner is used that explores all the current neighbours in the bi-dimensional $\alpha \times \mathcal{B}$ space; iii) `stab jmp X - stab`, an alternate policy, which starts by exploring the α space until it stabilizes. In phase 2, an exploration in the $\mathcal{B}$ space is performed till stabilization. In phase 3, and in the subsequent odd phases, it optimizes α performing random jumps with probability X% until it stabilizes on a new optimum configuration. In phase 4, and in the subsequent even phases, it explores the $\mathcal{B}$ dimension until it stabilizes.

Figure 5 shows the EDP normalized w.r.t. the best static configuration which was found by testing offline different combinations of $\mathcal{T}$ and $\mathcal{B}$. An interesting fact that can be deduced from these results is that using higher probability of random jumps after performing a stabilization yields better results as compared to using a single learner. We argue that this can depend on the fact that, in order to escape from a local minimum, a larger number of attempts is required, on average, in a bi-dimensional space. Hence, using a larger jump probability is more beneficial in this scenario. The 2nd best option, with an only marginally higher EDP value with respect to `stab jmp 10 - stab` is represented by the independent tuners. This suggests that, despite the lack of synchronization between the two tuners, they can still successfully crawl the search space and quickly identify high quality solutions.

B. Evaluating Green-CM

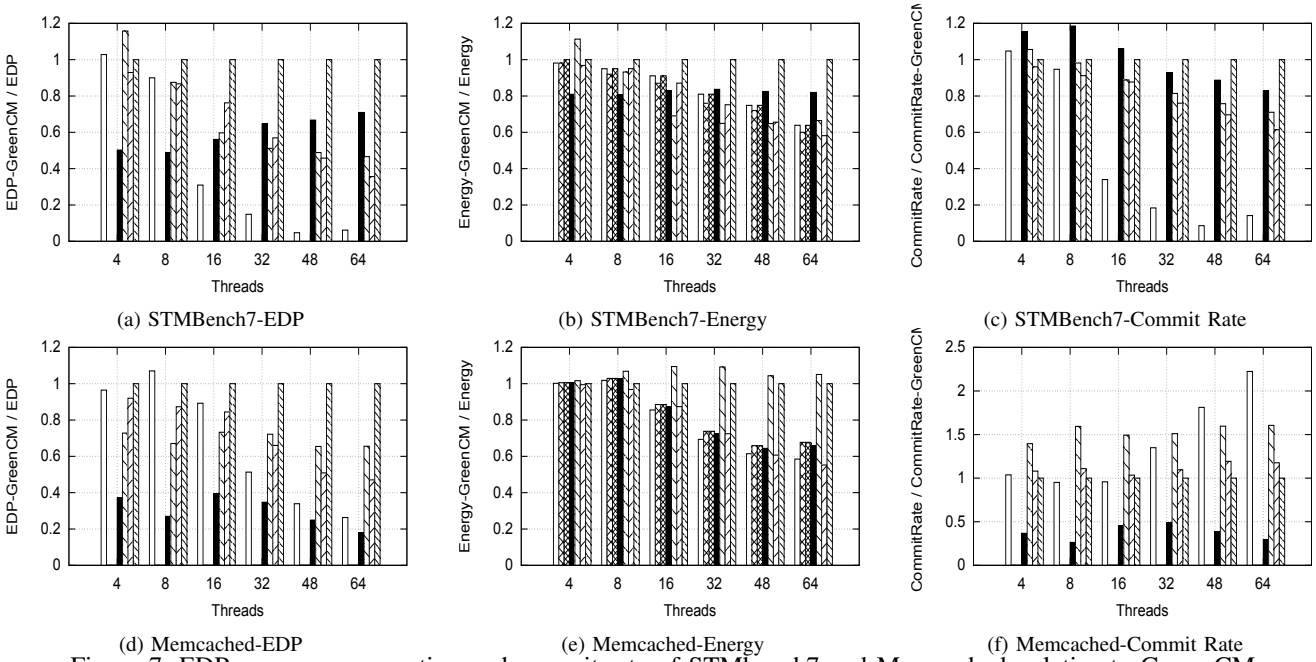
In this section we compare Green-CM, using the `stab jmp 10 - stab` tuner, with respect to the following state of the art CMs: suicide, karma, timestamp (ts), aggressive (agg), exponential back-off with sleep for back-off imple-



(a) Intruder-EDP (b) Intruder-Energy (c) Intruder-Time

(d) Kmeans-EDP (e) Kmeans-Energy (f) Kmeans-Time

Figure 6: EDP, energy consumption and execution time of Intruder and Kmeans, relative to Green-CM



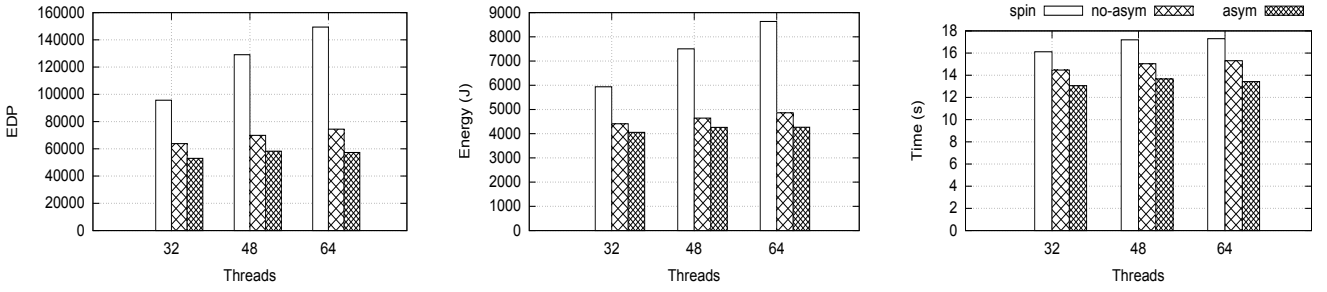
(a) STMBench7-EDP (b) STMBench7-Energy (c) STMBench7-Commit Rate

(d) Memcached-EDP (e) Memcached-Energy (f) Memcached-Commit Rate

Figure 7: EDP, energy consumption and commit rate of STMbench7 and Memcached, relative to Green-CM

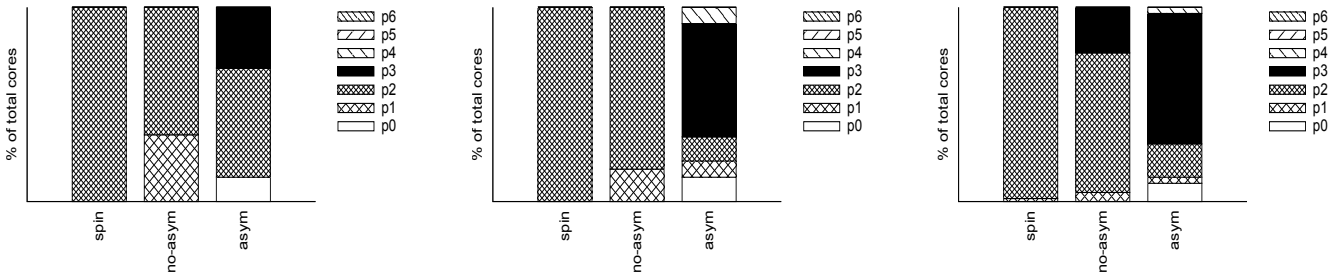
mentation (sleep) and exponential back-off with spin for back-off implementation (spin).

Figure 6, shows the EDP, energy consumption and running time for Intruder and Kmeans. Figure 7 shows the EDP, energy consumption and commit rate for STMbench7 [15] and Memcached [16]. We normalized the performances of the considered CMs with respect to the ones of Green-CM, defining the normalization in such a way to guarantee that values higher (resp. lower) than one mean worse (resp. better) performance than Green-CM, independently of the considered metric. This was performed to enhance data visualization, as in some cases Green-CM outperforms



(a) EDP (b) Energy (c) Time

Figure 8: EDP, energy consumption and time for Intruder with and without ACM enabled.



(a) Intruder, 32 threads (b) Intruder, 48 threads (c) Intruder, 64 threads

Figure 9: Frequency distribution of cores for different thread configurations for Intruder.

existing CMs by various high orders of magnitude.

Overall, Green-CM can achieve up to 2.35 times lower EDP than the best other contention manager with an average of 65% improvement across all benchmarks at 64 threads and 83% improvements at 48 threads with an overall average gain of 25% across all benchmark and thread configurations. Green-CM also achieves better efficiency in terms of EDP in most thread configurations higher than 4 threads for all benchmarks except KMeans and Memcached where it is as good as the best competitor at 8 threads..

We can note also that the gains from using Green-CM are more prevalent at higher number of threads. This is expected, since the higher the thread counts, the higher the contention level, the higher the relevance of contention management.

Finally to demonstrate the individual impact of using an asymmetric contention management strategy we evaluate our system with and without asymmetry enabled. Figure 8 shows the EDP, energy consumed and execution time for running Intruder with three different configurations: exponential back-off using a spin-based implementation (spin), tuning only α with $\mathcal{B} = 0$ (no-asym), and Green-CM with both tuners enabled (asym).

From the results we can see that using an asymmetric policy for contention management yielded extra gains in terms of both energy and performance reaching around 25% at 64 threads. These gains can be explained by correlating the results with the average core frequency operating distribution charts shown in Figure 9. These charts show the distribution of cores according to their average operating frequency throughout the running time of the benchmarks. Note that the considered AMD processor support 7 different frequency levels, P0, .., P6, where P0 is the highest frequency (3.0GhZ), P6 is the lowest (1.4 GhZ) and P2 is the nominal frequency (2.1GhZ).

It can be seen that between 10 to 25% of the cores reach the maximum boosted state (P0) when asymmetry is enabled, providing evidence on the effectiveness of ACM to favour the spontaneous activation of hardware-controlled DVFS mechanisms. Another aspect that can be noted is that, as the number of threads increase, more cores get to operate at lower frequencies: this is a consequence of the increase of contention, which leads threads to back-off for longer periods. This explains the gains in terms in of energy efficiency compared to exponential back-off using spin for the back-off implementation.

V. Conclusion

In this work we investigated the design of Green-CM, an energy efficient contention manager for transactional memory systems. We evaluated, using realistic workloads and actual TM system, the energy efficiency of alternative implementations of the back-off primitive, i.e., one fundamental building block of several state of the art contention management techniques. In the light of this study, we proposed a hybrid implementation that determines the most efficient back-off implementation to use, on the basis of the specified back-off period.

On top of this building block, we designed an asymmetric policy for contention management that aims to favour the activation of the DVFS mechanisms that are ubiquitously present in modern CPU architectures. Our experimental study shows that the proposed solutions shows improved EDP for various workloads compared to state of the art.

REFERENCES

[1] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *ISCA '93*.

[2] R. M. Yoo, C. J. Hughes, K. Lai, and R. Rajwar, "Performance evaluation of Intel transactional synchronization extensions for high-performance computing," in *SC '13*.

[3] C. Jacobi, T. Slegel, and D. Greiner, "Transactional memory architecture and implementation for IBM System Z," in *MICRO-45*.

[4] R. Guerraoui, M. Herlihy, and B. Pochon, "Toward a theory of transactional contention managers," in *PODC '05*.

[5] W. N. Scherer, III and M. L. Scott, "Advanced contention management for dynamic software transactional memory," in *PODC '05*.

[6] R. Guerraoui, M. Herlihy, and B. Pochon, "Polymorphic contention management," in *DISC'05*. Springer-Verlag.

[7] R. Guerraoui, M. Herlihy, M. Kapalka, and B. Pochon, "Robust contention management in software transactional memory," in *SCOOL*.

[8] T. Moreshet, R. I. Bahar, and M. Herlihy, "Energy reduction in multiprocessor systems using transactional memory," in *ISLPED '05*.

[9] A. Baldassin, J. P. L. de Carvalho, L. A. G. Garcia, and R. Azevedo, "Energy-performance tradeoffs in software transactional memory," in *SBAC-PAD '12*.

[10] U. Hoelzle and L. A. Barroso, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers.

[11] B. et al, "A dynamic voltage scaled microprocessor system," *IEEE Journal of Solid-State Circuits*, vol. 35.

[12] A. Branover, D. Foley, and M. Steinman, "AMD fusion APU: Llano," *IEEE Micro*, vol. 32, no. 2, Mar.

[13] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the Intel microarchitecture code-named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, Mar.

[14] C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford transactional applications for multiprocessing," in *IISWC '08*.

[15] R. Guerraoui, M. Kapalka, and J. Vitek, "STMBench7: A benchmark for software transactional memory," in *EuroSys '07*.

[16] W. Ruan, T. Vyas, Y. Liu, and M. Spear, "Transactionalizing legacy code: An experience report using GCC and Memcached," in *ASPLOS '14*.

[17] L. Dalessandro, M. F. Spear, and M. L. Scott, "NOrec: Streamlining STM by abolishing ownership records," in *PPoPP '10*. ACM.

[18] P. Felber, C. Fetzer, and T. Riegel, "Dynamic performance tuning of word-based software transactional memory," in *PPoPP '08*.

[19] A. Dragojević, R. Guerraoui, and M. Kapalka, "Stretching transactional memory," in *PLDI '09*. ACM.

[20] N. Diegues and P. Romano, "Time-Warp: Lightweight abort minimization in transactional memory," in *PPoPP '14*.

[21] S. Sanyal, S. Roy, A. Cristal, O. S. Unsal, and M. Valero, "Clock gate on abort: Towards energy-efficient hardware transactional memory," in *IPDPS '09*.

[22] C. Ferri, S. Wood, T. Moreshet, R. Iris Bahar, and M. Herlihy, "Embedded-TM: Energy and complexity-effective hardware transactional memory for embedded multicore systems," *Journal of Parallel and Distributed Computing*, vol. 70, no. 10, 2010.

[23] E. Gaona-Ramrez, J. R. T. Gil, J. Fernndez, and M. E. Acacio, "Characterizing energy consumption in hardware transactional memory systems," in *SBAC-BAD '10*.

[24] D. Rughetti, P. Di Sanzo, and A. Pellegrini, "Adaptive transactional memories: Performance and energy consumption tradeoffs," in *NCCA '14*.

[25] D. Rughetti, P. Romano, F. Quaglia, and B. Ciciani, "Automatic tuning of the parallelism degree in hardware transactional memory," in *Euro-Par'14*, 2014.

[26] N. Diegues, P. Romano, and L. Rodrigues, "Virtues and limitations of commodity hardware transactional memory," in *PACT '14*.

[27] N. Diegues and P. Romano, "Seer: Probabilistic scheduling for hardware transactional memory," in *SPAA '15*.

[28] J.-T. Wamhoff, S. Diestelhorst, C. Fetzer, P. Marlier, P. Felber, and D. Dice, "The TURBO Diaries: Application-controlled frequency scaling explained," in *USENIX ATC'14*.

[29] J.-T. Wamhoff, C. Fetzer, P. Felber, E. Rivière, and G. Muller, "FastLane: Improving performance of software transactional memory for low thread counts," in *PPoPP '13*.

[30] C. Ferri, R. I. Bahar, M. Loghi, and M. Poncino, "Energy-optimal synchronization primitives for single-chip multiprocessors," in *GLSVLSI '09*.

[31] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.