The Implications of Shared Data Synchronization Techniques on Multi-Core Energy Efficiency

Ashok Gautham[1], Kunal Korgaonkar[1,2], Patanjali SLPSK[1], Shankar Balachandran[1], and Kamakoti Veezhinathan[1]

[1]Department of Computer Science and Engineering, IIT Madras
[2]IBM Research Lab, Bangalore, India

Abstract

Shared data synchronization is at the heart of the multi-core revolution since it is essential for writing concurrent programs. Ideally, a synchronization technique should be able to fully exploit the available cores, leading to improved performance. However, with the growing demand for energy-efficient systems, it also needs to work within the energy and power budget of the system. In this paper, we perform a detailed study of the performance as well as energy efficiency of popular shared-data synchronization techniques on a commodity multi-core processor. We show that Software Transactional Memory (STM) systems can perform better than locks for workloads where a significant portion of the running time is spent in the critical sections. We also show how power-conserving techniques available on modern processors like C-states and clock frequency scaling impact energy consumption and performance. Finally, we compare the performance of STMs and locks under similar power budgets.

1 Introduction

Prior to the multi-core era, every new processor generation speeded up sequential programs. With the introduction of mulct-core systems, it became necessary to make programs multi-threaded to take advantage of the cores. When multiple threads are in flight, one needs to use synchronization techniques to coordinate data access. Traditional synchronization techniques like spinlocks and mutexes introduce wasteful and idle cycles respectively while attempting to acquire the lock. If no other workload can fill up these idle cycles, then this may result in core under-utilization. Ideally, programmers can use *fine-grained locks* to reduce the number of wasted cycles and achieve highly concurrent systems but they have to be vigilant to ensure that they have not introduced any deadlocks. Furthermore, correctness of fine-grained lock based programs is hard to guarantee. *Coarse-grained locks* on the other hand are easy to program and reason about, but they form a bottleneck in exploiting the parallelism available.

While locks have been used for decades, research in synchronization has resulted in the development of Transactional Memory (TM). Software Transactional Memory [10] provides an easy-to-use abstraction for writing correct multi-threaded programs. Also, it is optimistic in nature and is thus able to exploit non-explicit parallelism available in programs. However, for low thread counts, the overhead associated with TM might lower the benefits obtained from using it [1]. For higher thread counts, the scalability and speedup obtained from STMs compensate for this overhead as demonstrated later in this paper.

2 Motivation

The race in the current mainstream processor market is no longer just for performance, but also for increased energy efficiency. Occurrence of a high percentage of wasteful or idle CPU cycles indicates under-utilization of the core. When a system's total energy/power budget is kept under perspective, wasted cycles can in turn lead to wastage of energy. Esmaeilzadeh et. al [2] indicates that though more cores will be available in newer processors, power constraints will limit the extent to which they can be used. Moreover, the two largest consumers of processors - data centers and battery-operated mobile devices - demand energy efficient systems. A prioritization of efforts towards achieving multi-core energy efficiency is thus the order of the day.

In the era of multi-cores, the synchronization technique should ideally: *(i) Exploit the hardware parallelism only if it leads to improved performance for the*

[1]Hardware transactional memory (HTM) can lower this overhead. We use STMs for our analysis since a full-fledged HTM isn't available on commodity hardware, though there are several research prototypes.

additional energy expenditure; (ii) Enable the workload to conserve energy whenever possible without severely impacting performance. The choice of synchronization technique is therefore crucial for multi-core systems.

Given that there is no "one-size-fits-all" solution for energy-efficient concurrent systems, a detailed study of energy characteristics of workloads is necessary to understand which technique would be better for a given application. In this paper, we explore the following synchronization techniques to find an ideal solution for synchronization intensive workloads: i) Spinlock ii) Mutex iii) Software Transactional Memory.

2.1 Contributions of this Paper

- To our best knowledge, this is the first paper to evaluate the energy-efficiency of STMs on a real-world commodity multi-core with a full-fledged OS.
- The results indicate that STMs can in fact be more time and energy efficient than locks for workloads that spend significant time inside critical sections. This is in contrast to previously reported results [4].
- It is also the first paper to explore the impact of power-saving techniques in modern multi-cores like C-states and CPU frequency scaling on synchronization techniques using commodity hardware.
- It also explores the possibility of trading a small fraction of performance for larger reduction in power consumption of STMs.

The choice of commodity hardware and software was to improve the immediate practical applicability of the results. Also, since the values obtained are on real hardware, the shortcomings of simulation-based approaches are avoided.

2.2 Evaluation Metrics

We evaluate the synchronization techniques based on the following metrics which have been chosen carefully to express energy, power and timing characteristics exhibited by the workloads.

Time or Delay: Time taken for the workload to complete.

Energy Delay Product: Product of the time spent by the application and the energy expended while running.

Performance per Watt: Number of transactions (critical sections) processed per second for 1 Watt of power. The values obtained for PPW cannot be directly compared across benchmarks since the length of critical sections in different benchmark applications differ.

3 Related Work

Moreshet et. al [7] were the first to compare the energy efficiency of transactional memory-based and lock-based programs. Experiments were carried out on a Simics-based model of a HTM. While all analysis was done on a rather simple shared-array micro-benchmark, it showed that transactional memory based systems have the capability to outperform locks by a factor of more than 5x in terms of energy. Similar conclusions were drawn in [8] where the SPLASH-2 benchmark suite was used.

Klein et. al [4] studied the performance of Software Transactional Memory systems against that of locks for the more recent STAMP [6] benchmark suite on an MPSoC based simulated system. In this setup, locks are implemented as spinlocks and compare-and-swap is implemented using semaphores. Furthermore, the system does not include an OS. To compensate for not being able to deschedule the lock, the authors deduct the energy consumed by the lock while spinning. Given these assumptions, the authors show that STMs can be up to 22x less energy-efficient than locks. However, the interactions between the OS and the application cannot be ignored as highlighted by Liu [5].

To our best knowledge, no previous work has compared energy efficiency of classical synchronization techniques (like locks) with STMs when run on real hardware.

4 Experimental Methodology

We ran the benchmark applications in STAMP on Intel's Core i7-2600 CPU. As STAMP ships only with STM and sequential modes of execution, we ported it to run using spinlocks with an exponential back-off and pthread mutex. Further, the benchmark applications were annotated to read from the RAPL [1] energy counters at the start and end of the region of interest. This allows exclusion of energy consumed by the setup and verification times of the algorithm from the reported values. All experiments were run for 8 threads (pinned to corresponding logical cores). Linux's default `CFS` process scheduler and `menu` cpuidle governor were used for all experiments.

RAPL: Running Average Power Limit (or RAPL) counters introduced in [1] are available in processors based on Intel's Sandy Bridge micro-architecture. These counters allow us to get accurate energy consumption statistics of the processor package from an on-board controller. Intel specifies that these counters are updated every 1ms. Hahnel [3] and Rotem [9] further demonstrate that these counters match external measurements. As DRAM energy isn't measured in the processor we used, we cross-validated our results using external power measurement.

Choice of STM: RSTM [11] (version 7) was chosen as

Table 1: Comparison of Sequential, Mutex, SpinLock and Swiss implementations

	bayes	genome	K-Means High	KMeans Low	Vacation High	Vacation Low	intruder	ssca2	yada
$\%T_{crit}$	83%	97%	7%	3%	86%	86%	33%	17%	100%
Time(s)									
Seq	22.08	6.46	1.68	10.35	16.31	12.4	14.61	11.99	6.76
Mutex	23.43	6.04	0.78	2.55	19.71	15.36	20.22	15.33	8.3
Spinlock	25.47	5.55	**0.47**	**2.53**	18.12	13.49	15.87	9.98	9.58
STM	**17.38**	**1.57**	0.91	3.97	**5.59**	**3.99**	**6.49**	**7.57**	**4.16**
EDP(Js)									
Seq	9452.23	697.42	53.47	1945.68	4377.48	2492.29	3428.81	2420.88	**881.93**
Mutex	10258.45	1285.23	35.6	**407.38**	12559.96	7906.66	14642.84	9555.71	2097.92
Spinlock	20285.58	1232.16	**13.31**	414.53	14051.22	7786.22	10882.99	2980.57	4010.67
STM	**6570.45**	**118.15**	51.74	1021.13	**1667.24**	**850.92**	**2146.31**	**1522.57**	975.26
PPW(Txns/s/W)									
Seq	4.64	23071.7	129264.36	60380.18	**15623.07**	**20865.19**	**99816.52**	110747.75	**20353.3**
Mutex	4.53	11692.83	90169.26	52525.35	6580.49	8146.44	32358.79	35878.25	10504.47
Spinlock	2.49	11216.91	**145951.63**	**61732.97**	5408.19	7264.56	34165.89	74848.77	6344.28
STM	**24.3**	**33180.97**	72069.23	38357.32	14064.76	19668.78	70801.15	**111216.09**	11325.49

the STM framework since it is very configurable and allows broader exploration on the STM algorithm front. The Swiss algorithm that uses eager locking, redo-logs and mixed invalidation is used as the algorithm of choice since it was experimentally determined to be competitive.

Choice of Benchmark Suite: STAMP [6] is a synchronization-intensive benchmark suite that has a mix of applications from various domains like machine learning, data mining and online transaction processing . It proves to be an excellent choice for our analysis since it consists of a wide variety of applications, in terms of the percentage of time spent by them in critical sections (T_{crit}) which ranges from less than 5%, to close to 100% [6](only indicative - may vary based on technique). For example, the `vacation` benchmark application models an online transaction processing system (similar to `specjbb2000`) spends close to 90% time in critical section while `K-Means` from data mining spends only 2% time in it. STAMP applications also use non-linear data structures like trees. Usage of fine-grained locks for such data structures is complicated [6]. Coarse grained locks and TM are therefore the only viable options for parallelization.

Choice of Hardware Platform: An Intel Core i7-2600 CPU was used for all experiments. This system is based on Sandy Bridge microarchitecture. It has 4 physical cores and uses Intel's HyperThreading technology - effectively providing 8 parallel hardware threads of execution. The processor has the capability of running at several frequency levels between 1.6 GHz and 3.4 GHz. The processors supports 3 low-power idle states (C1, C3 and C6). Each core can independently be at any of these states or the active C0 state. All experiments were run on a 64-bit system running Linux kernel 3.2.0.

5 Results

From Table 1, one can see that neither spinlocks nor mutexes perform well for most applications. For some benchmarks, the lock-based programs show a slowdown when compared to even the sequential implementation. Further, STMs perform well in all metrics. For the `vacation` benchmark under high-contention, there is a 224% speedup over spinlocks and a 653% reduction in EDP over mutex locks. Also, there is a 213% improvement in PPW over Mutex. Though STM and sequential implementations have comparable PPW, STM is almost 3 times faster than its sequential counterpart and also has a superior EDP.

5.1 Analysis

Performance of the sequential implementation : The EDP of the sequential implementation is competitive for several benchmarks though the performance is poor. Since sequential programs run only on one core, the other cores are in deep sleep (C6 state) and thus consume very little power. Therefore, even though the workload takes longer to run, the instantaneous PPW is quite high.
Performance of the lock-based implementations: We used both spin-locks and mutexes since they represent a performance-power tradeoff. Spinlocks consume more power because of the busy-wait, while mutexes experience a slowdown because of OS-initiated scheduling and descheduling of threads.

Locks outperform sequential programs only in 3 of the 7 applications presented. To understand why the spinlocks behave badly for most of the workloads, the spinlock based `genome` was profiled using the perf utility. As expected, there was no spinning in the 1-threaded case.

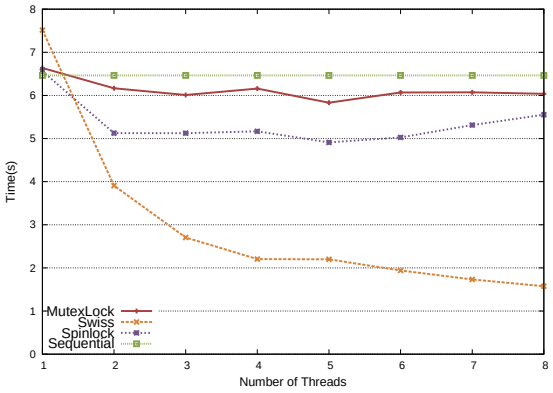

Figure 1: No. of Threads vs Time - Genome

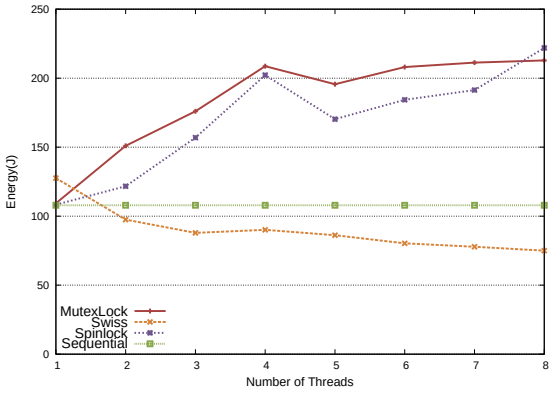

Figure 2: No. of Threads vs Energy - Genome

In the 2-threaded case, close to 40% of the time was spent in spinning. This further rose to 70% for 4-threads and 84% when run with 8 threads.

The behaviour of the mutex can be explained by the `user` and `system` times of the applications run. The `system` time indicates the time spent in the OS which we believe is a good indicator of the application-OS interactions. As the number of threads increases, the `system` time increases rapidly for mutex-based implementations. For example, in the 1-threaded case the `system` time for `genome` is 0.372 seconds. However, this time rises to 1.2 for the 2-threaded case, 1.8 for 4-threaded case and 4.02 seconds for the 8-threaded case. This is in spite of the overall wall clock time remaining fairly consistent across thread count.

The `K-Means` benchmark spends a small portion of its time in the critical section. This results in a contrary trend for `K-Means`. It is the only benchmark where lock-based programs perform better than STMs. Also, because of the large portion of code where all threads can run freely without needing synchronization, there is a significant speedup obtained by using any multi-threading technique.

Performance of STM-based implementations The table also indicates that for most benchmarks, STMs perform the best with respect to performance and overall energy consumed. This can be attributed to ability of STMs to exploit disjoint access parallelism that may not be explicitly annotated in the program. Further, since the number of threads is 8, the initial overhead of STMs is compensated for by the additional parallelism exploited. For `Yada`, STMs show better performance but still do worse than sequential with respect to energy. This can be attributed to the CPU idle states that can be exploited by sequential programs for the inactive cores. This shows that better performance alone is not indicative of better energy efficiency further motivating study of energy-efficiency in its own right.

6 Other Analysis

6.1 Scaling with number of threads

Figures 1 and 2 show how the performance of the `genome` benchmark scales with respect to the number of threads. Though the overhead of STM makes it slower for the single threaded case, it scales better as more threads are added. The spinlock and mutex are unable to utilize the extra parallelism available. Due to optimistic execution, STM is able to exploit disjoint access parallelism, which is unavailable in case of lock based systems. Further, even though the performance of mutex and spinlocks get better when compared to sequential, their energy efficiency becomes much worse than sequential as thread count increases because of the additional cores in operation.

6.2 Impact of CPU frequency scaling governors

Table 2: Impact of CPU frequency scaling governors on Mutex

Governor	E	EDP	PPW
Performance	142.8	699.1	17429.2
Ondemand	141.8	792.3	17550.8
Conservative	130.7	1210.1	19034.2
Powersave	127.9	1299.4	19457.9

CPU frequency governors enable lowering CPU frequencies when there are idle cycles, effectively reducing the application's energy wastage. `Performance` and `powersave` are non-adaptive governors that always run the processor at the highest and the lowest available frequencies respectively. Adaptive CPU Frequency governors like `ondemand` and `conservative` dynamically changes the clock frequency of the core based on the uti-

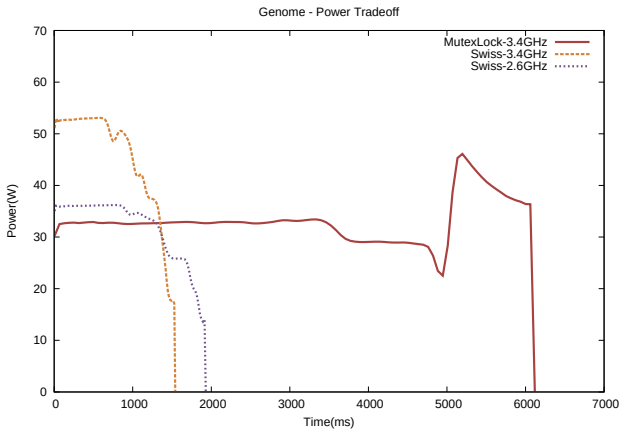

Figure 3: STM Performance-Energy trade-off

lization levels. Adaptive governors are more beneficial to mutex locks than to spinlocks or STMs. This is because mutex lock have a lot of idle cycles unlike spinlocks or STM. The impact of governors on mutex lock's performance is shown in Table 2. STMs hardly have idle cycles since the execution is optimistic. For STMs, both adaptive algorithms considered converge at the highest possible frequency and thereby have very little impact on the energy consumption of the application.

6.3 Trading off Performance for Energy

Though STMs have a better PPW than lock-based implementations in the benchmarks considered, high CPU utilization also translate into higher average and peak power consumption. Considering this disparity between STM and mutex lock, we need to compare STM and mutex on a common ground. For this, we use equi-watt comparison where we compare performance of STM with the performance of mutex lock at the same wattage. We achieve this by lowering the CPU clock frequency of the cores for STM runs such that the respective average power consumption values matches that of the mutex-based implementation. Table 3 lists the average and peak power numbers for STM and mutex for different clock frequencies for the `genome` benchmark application. Figure 3 shows the smoothed run time power consumption for STM and mutex running at full clock frequency and STM running at a lower clock frequency where its power consumption is roughly equal to that of the mutex-based implementation. We notice that even after reducing the clock frequencies the performance of STM is still over 3 times faster than the mutex implementation, without any increase in power for `genome`.

Frequency (GHz)	Mutex Peak(W)	Mutex Avg(W)	Swiss Peak(W)	Swiss Avg(W)
1.6	26.7833	13.7310	24.2977	18.4723
1.8	29.7852	15.7801	27.5080	20.8763
2.0	32.6613	17.1100	30.8812	22.9525
2.2	32.8765	18.9044	34.6230	26.8045
2.4	36.8203	20.6330	38.4482	28.3442
2.6	39.8757	23.2967	42.7315	31.6619
2.8	44.5601	24.9497	49.4253	36.1220
3.0	58.6239	28.2032	52.2302	40.3889
3.4	57.1547	33.0223	62.4075	45.1334

Table 3: Clock frequency vs Peak and Average Power

7 Future Directions

This paper carries out a preliminary analysis of the energy efficiency of synchronization techniques on the STAMP benchmark applications. However, STAMP does not contain transactions that have irrevocable operations like I/O which may have different performance characteristics. Other benchmarks like RMSTM need to be explored for this. Techniques like speculative lock elision that remove the bottlenecks associated with coarse grained locks too need to be accounted for.

References

[1] David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R., and Le, C. RAPL: Memory power estimation and capping. In *Proceedings ofISLPED '10* (2010), ACM, pp. 189–194.

[2] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., and Burger, D. Dark silicon and the end of multicore scaling. *SIGARCH Computer Architecture News 39*, 3 (June 2011), 365–376.

[3] Hahnel, M., Dobel, B., Volp, M., and Hartig, H. Measuring Energy Consumption for Short Code Paths Using RAPL. In *Proceedings of GREENMETRICS'12* (2012).

[4] Klein, F., Baldassin, A., Moreira, J., Centoducatte, P., Rigo, S., and Azevedo, R. STM versus Lock-Based Systems: An Energy Consumption Perspective. In *Proceedings of ISLPED '10* (2010), ACM, pp. 431–436.

[5] Liu, Y. Energy-efficient synchronization through program patterns. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on* (june 2012), pp. 35 –40.

[6] Minh, C. C., Chung, J., Kozyrakis, C., and Olukotun, K. STAMP: Stanford Transactional Applications for Multi-Processing. In *Proceedings of IISWC'08* (2008), pp. 35–46.

[7] Moreshet, T., Bahar, R. I., and Herlihy, M. Energy reduction in multiprocessor systems using transactional memory. In *Proceedings of ISLPED '05* (2005), ACM, pp. 331–334.

[8] Moreshet, T., Bahar, R. I., and Herlihy, M. Energy-aware microprocessor synchronization: Transactional memory vs. locks, 2006.

[9] Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., and Rajwan, D. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro 32*, 2 (2012), 20–27.

[10] Shavit, N., and Touitou, D. Software transactional memory. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing* (1995), ACM, pp. 204–213.

[11] Spear, M. F., Dalessandro, L., Marathe, V. J., and Scott, M. L. A comprehensive strategy for contention management in software transactional memory. In *Proceedings of the PPoPP'09* (2009), ACM, pp. 141–150.