Optimizing Throughput/Power Trade-offs In Hardware Transactional Memory Using DVFS and Intelligent Scheduling

Clay Hughes
ECE Department
Florida State University
Panama City, FL 32405
chughes@pc.fsu.edu

Tao Li
ECE Department
University of Florida
Gainesville, FL 32611
taoli@ece.ufl.edu

ABSTRACT
Power has emerged as a first-order design constraint in modern processors and has energized microarchitecture researchers to produce a growing number of power optimization proposals. Almost in tandem with the move toward more energy-efficient designs, architects have been increasing the number of processing elements (PEs) on a single chip and promoting the concept of running multithreaded workloads. Nevertheless, software is still lagging behind and is often unable to exploit these additional resources – giving rise to transactional memory. Transactional memory is a promising programming abstraction that makes it easier for programmers to exploit the resources available in many-core processor systems by removing some of the complexity associated with traditional lock-based programming. This paper proposes new techniques to merge the power and transactional memory domains.

An analysis of the per-core and chip-wide power consumption of hardware transactional memory systems (HTMs) pinpoints two areas ripe for power management policies: transactional stalls and aborts. The first proposed policy uses dynamic voltage and frequency scaling (DVFS) during transactional stall periods. By frequency scaling PEs based on their transactional state, DVFS can increase the throughput and energy efficiency of HTMs. The second method uses a transaction’s conflict probability to reschedule transactions and clock gate aborted PEs to reduce overall contention and power consumption within the system. The proposed techniques are evaluated using three HTM configurations and are shown to reduce the energy delay squared product (ED2P) of the STAMP and SPLASH-2 benchmarks by an average of 18% when combined. Synthetic workloads are used to explore a wider range of program behaviors and the optimizations are shown to reduce the ED2P by an average of 29%. For a comparison, this work is shown reduce the ED2P by up to 30% relative to previous proposals for energy reduction in HTMs (e.g., transaction serialization).

Categories and Subject Descriptors
C.5.0 [Computer System Implementation]: General

General Terms
Performance, Design.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ABSTRACT
Power has emerged as a first-order design constraint in modern processors and has energized microarchitecture researchers to produce a growing number of power optimization proposals. Almost in tandem with the move toward more energy-efficient designs, architects have been increasing the number of processing elements (PEs) on a single chip and promoting the concept of running multithreaded workloads. Nevertheless, software is still lagging behind and is often unable to exploit these additional resources – giving rise to transactional memory. Transactional memory is a promising programming abstraction that makes it easier for programmers to exploit the resources available in many-core processor systems by removing some of the complexity associated with traditional lock-based programming. This paper proposes new techniques to merge the power and transactional memory domains.

An analysis of the per-core and chip-wide power consumption of hardware transactional memory systems (HTMs) pinpoints two areas ripe for power management policies: transactional stalls and aborts. The first proposed policy uses dynamic voltage and frequency scaling (DVFS) during transactional stall periods. By frequency scaling PEs based on their transactional state, DVFS can increase the throughput and energy efficiency of HTMs. The second method uses a transaction’s conflict probability to reschedule transactions and clock gate aborted PEs to reduce overall contention and power consumption within the system. The proposed techniques are evaluated using three HTM configurations and are shown to reduce the energy delay squared product (ED2P) of the STAMP and SPLASH-2 benchmarks by an average of 18% when combined. Synthetic workloads are used to explore a wider range of program behaviors and the optimizations are shown to reduce the ED2P by an average of 29%. For a comparison, this work is shown reduce the ED2P by up to 30% relative to previous proposals for energy reduction in HTMs (e.g., transaction serialization).

Categories and Subject Descriptors
C.5.0 [Computer System Implementation]: General

General Terms
Performance, Design.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2011 ACM 978-1-4503-0102-2/11/05...$10.00.

Keywords
Transactional memory, power.

1. INTRODUCTION
Power dissipation continues to be a first-order design constraint for modern computer designs from the chip level to data centers. At the chip level, power consumption can affect its reliability and performance and can increase packaging and manufacturing costs. And while chip multi-processors (CMPs) offer better energy efficiency than previous uniprocessors [14], they still suffer from the same heat removal problems as previous generations. However, unlike previous generations, CMPs provide more opportunities for balancing energy use. CMPs are designed for running multiple threads of execution, which often vary in performance and resource requirements, making them ideal candidates for runtime optimizations that can maximize program performance while minimizing chip power consumption.

Many of the threads executing in modern machines are actually disparate processes each running with a single thread. In order to exploit the types of resources offered by CMPs, programmers need to begin changing the way they write code, writing programs that consist of multiple threads that are able to take advantage of the ever-increasing number of processing elements (PEs). However, exploiting the available data and task parallelism in a program is often a challenging and time-consuming process, requiring significant time investments to extract performance and guarantee correctness. Transactional memory (TM) [7] has been proposed as a programming technique to replace locks, shifting some of the burden of synchronization from the programmer to the system architecture – in effect providing an abstraction of the implementation. While transactional memory was conceived of as a means to shift the programming burden, hardware implementations have ancillary benefits such as increased performance over locks and potential energy savings [21].

This work focuses on how transactional memory can be leveraged for energy and performance optimizations, making the following contributions:

- Dynamic frequency and scaling (DVFS) is introduced to reduce the power consumption of stalled processing elements and increase overall throughput by setting the clock frequency and supply voltage for each PE based on its current execution state and those of the collocated PEs. The optimization decreases the amount of time that a processor holds its read and write sets by increasing the clock frequency of NACK’d PEs and decreases the power consumption of NACK’d PEs by throttling the clock frequency. Using this DVFS policy improves the energy
delay product (ED2P), which is a joint measurement of the system power and performance, by up to 43%.

- A new preemptive transaction scheduler is introduced based on the system’s conflict density. This scheduling policy prevents potentially contentious transactions from issuing and clock gates the resident PE, reducing the system power. Furthermore, because there are fewer executing transactions those that remain running have a lower probability of experiencing an abort, which increases total throughput and reduces the system ED2P by as much as 76%.

- The new policies are then combined and compared against previously proposed power management techniques for transactional memory that use clock gating and transactional serialization. The new policies show improvements between 12% and 30% relative to the previous work.

The next section describes the motivation behind the proposed schemes. Section 3 contains the evaluation methodology and implementation details. The proposed power management policies and results are described in Section 4. The paper concludes with a discussion of related work in Section 5 and concludes in Section 6.

2. MOTIVATION

This section provides an overview of the power of the different hardware transactional memory systems using the SPLASH-2 [31] and STAMP [19] benchmarks. Benchmarks are referenced by the abbreviations in Table 3 and the transactional memory systems are described using their primary design points: conflict detection and version management. Conflict detection defines when conflicts are detected and version management defines where new and old values within a transaction are stored. Both use the same basic nomenclature and can be either eager or lazy. With eager conflict detection, addresses are checked for conflicts on each read and write within the transaction whereas lazy checks addresses when a transaction attempts to commit. Eager version management writes new values in place and copies old values elsewhere; lazy does the opposite, leaving old values in place and writing new values elsewhere. The system designs are referenced as LK – lock, EE – eager conflict/eager versioning, EL – eager conflict/lazy versioning, and LL – lazy conflict/lazy versioning.

Consider Figure 1, which shows a breakdown of the power consumption from 14 benchmarks using locks, eager-eager, eager-lazy, and lazy-lazy. There is no measurable difference in the average power consumption of the SPLASH-2 benchmarks for the lock and eager conflict detection schemes but the average power for lazy conflict detection is slightly higher due to increased log utilization. For the STAMP benchmarks, the average power varies from 32W to 192W. While any model would suffice for the SPLASH-2 benchmarks, eager-lazy minimizes the average power for the STAMP benchmarks. However, the ED2P for these benchmarks suggests that because their behavior is so diverse, there is no clear design choice that minimizes the power-performance. These experimental results show that when there is little or no contention, hardware transactional memory consumes approximately the same amount of power as a lock-based system. Under moderate or heavy contention, some of the transactional memory designs have much lower average power than their lock counterparts. What is important is that the slack power available in these benchmarks can be exploited to improve the performance while limiting the maximum chip power and temperature.

3. METHODOLOGY

This section describes the specific implementation details of the hardware transactional memory systems as well as the simulation environment, methodology, and benchmarks used to evaluate the power and energy characteristics of transactional memory workloads.

3.1 CMP DESIGN

Figure 2 shows the basic system architecture and Table 1 summarizes the design parameters. The CMP system consists of 4 processing elements based on 65nm technology; the base number of processors was chosen to reflect currently available configurations. The processors are 4-issue out-of-order with a split 64KB 4-way set associative write-back L1 cache. There is a 4MB 8-way set associative shared L2 cache split into 8 banks. The off chip memory is modeled as 4GB of DDR3. Cache coherence is maintained using an snooping-based MESI protocol. The power management structures are discussed in Section 3.2.

3.2 SIMULATOR DESIGN

The transactional memory simulator is a modified version of SuperTrans [22], a cycle-accurate detailed hardware transactional memory model that includes support for eager and lazy versioning and conflict detection modes. The conflict detection and version management schemes in SuperTrans are abstract, meaning while they were guided by previously proposed implementations [6]
SuperTrans was modified to mimic a generic signature-based transactional memory system similar to LogTM-SE [32] and BulkSC [4] and tracks read- and write-sets using per-processor Bloom filters [2]. Both versioning schemes implement a cacheable logging structure, which holds the virtual addresses and old (eager) or new (lazy) values of memory blocks modified during a transaction.

Table 1 lists both the core and transactional model parameters. Conflict detection is carried out per-cache line. The primary/secondary baseline latencies and primary variable latency quantify the latencies associated with a commit or an abort. The primary latency is associated with the long operation for the selected versioning scheme—abort for eager and commit for lazy. The secondary latency is the opposite; it sets the delays for a fast operation—commit for eager and abort for lazy. The baseline latency is the static overhead associated with a transaction (e.g., the sum of the bus arbitration and log overhead) and the variable latency is the additional time required for a transaction based on the transaction size.

The signature implementation uses the results from Sanchez et al. [25] and Yen et al. [33] for modeling the hardware implementation of signatures. Each 1024b signature is represented as 2 64B SRAMs along with the logic gates necessary to implement the Hn hashing functions. Each hash function consists of 5/2 2-input XORs for each bit of the hash and each XOR is assumed to consist of 6 transistors [30]. The dynamic power for each XOR was estimated using the following formula:

$$\Sigma_{i=1}^{N} \frac{1}{2} C_i V_{dd}^2 f_i$$

where $C_i$ is the output capacitance of the $i$th gate, $V_{dd}$ is the supply voltage, $f_i$ is the switching frequency, and $N$ is the total number of gates. The values were estimated using CACTI [28] and the switching frequency was assumed to be the clock frequency, which gives a worst-case estimation.

The power management system was modeled after Intel's Foxton Technology (FT) [18] and includes on-chip power and temperature sensors and a small microcontroller. Internally, the microcontroller was modeled as a single structure that consumes 0.5% of the total chip power. DVFS was added to SESC with the levels shown in Table 2. Watch [3] was integrated into the simulator to estimate the energy consumption for 64 individual structures per processor plus an additional 18 global structures based on values obtained from CACTI [28]. HotSpot [27] was used to estimate on-chip temperature, which is based on the current chip power and feeds into HotLeakage [35] to estimate the leakage power. Although recent work has explored the feasibility of on-chip regulators [12], this work assumes that voltage transitions require approximately 50k cycles at the base frequency or 20µs. When down-scaling the DVFS level, the frequency drop occurs over a two cycle period with the voltage lagging behind over the transition period. Up-scaling the DVFS level increases the frequency and voltage simultaneously over the transition period.

### 3.3 Workloads

For the evaluation, 14 benchmarks from two different benchmarking suites (SPLASH-2 and STAMP) along with 15 synthetic benchmarks were used. While SPLASH-2 provides a good comparison of design points for fine-grained transactions and highly optimized lock-behavior, it is believed that future transactional workloads will also be comprised of coarse granularity transactions that may not be well tuned. To capture this trend, workloads from the STAMP suite (ver. 0.9.6) of transactional benchmarks are used in the evaluation. Since the STAMP suite does not provide lock-based equivalents of the transactional benchmarks, lock versions were generated using the same level of granularity as the transactions. Table 3 gives the input set used for each benchmark. All benchmarks were run to completion.

TransPlant [23], a parameterized transactional memory benchmark creation tool, was used to generate the synthetic benchmarks. TransPlant takes a statistical descriptor file as an input and produces C-code that can be compiled and run on a simulator. Table 4 describes the first order design parameters that the user can specify. One of the goals of this work is to isolate those program characteristics that have the largest impact on the power. To accomplish this, the workloads are constructed so that the transactional work, in terms of instruction count and composition, is held constant. While task decomposition in real applications is not straightforward, keeping the total work

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Abbreviation</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>barnes</td>
<td>BN</td>
<td>16K particles</td>
</tr>
<tr>
<td>bayes</td>
<td>BY</td>
<td>1024 records</td>
</tr>
<tr>
<td>cholesky</td>
<td>CH</td>
<td>tk15.O</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>FA</td>
<td>35KMbps</td>
</tr>
<tr>
<td>fmm</td>
<td>FM</td>
<td>16K particles</td>
</tr>
<tr>
<td>genome</td>
<td>GN</td>
<td>g256 s16 n16384</td>
</tr>
<tr>
<td>kmeans</td>
<td>KM</td>
<td>Random1000_12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Abbreviation</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>labyrinth</td>
<td>LB</td>
<td>x32-y32-z3-n96</td>
</tr>
<tr>
<td>ocean-con</td>
<td>OC</td>
<td>258x258</td>
</tr>
<tr>
<td>ocean-non</td>
<td>ON</td>
<td>66x66</td>
</tr>
<tr>
<td>raytrace</td>
<td>RT</td>
<td>Teapot</td>
</tr>
<tr>
<td>vacation</td>
<td>VA</td>
<td>4096 tasks</td>
</tr>
<tr>
<td>water-ngs</td>
<td>WN</td>
<td>512 molecules</td>
</tr>
<tr>
<td>water-sp</td>
<td>WS</td>
<td>512 molecules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Freq (GHz)</th>
<th>2.93</th>
<th>2.67</th>
<th>2.40</th>
<th>2.27</th>
<th>2.20</th>
<th>2.13</th>
<th>2.00</th>
<th>1.87</th>
<th>1.73</th>
<th>1.60</th>
<th>1.47</th>
<th>1.33</th>
<th>1.20</th>
<th>1.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{dd}$ (V)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
<td>0.95</td>
<td>0.93</td>
<td>0.90</td>
<td>0.87</td>
<td>0.84</td>
<td>0.80</td>
<td>0.77</td>
<td>0.72</td>
<td>0.67</td>
<td>0.60</td>
</tr>
</tbody>
</table>
constant allows variables to be isolated. For example, if work was not held constant, transaction granularity could not be used as an independent variable in these workloads. Unless otherwise noted, transactions are evenly spaced throughout the program, allowing for a direct comparison across dimensions. Each transaction is responsible for at least one unique load and one unique store so that all transactions have at least some chance of conflicting; the probability of a conflict is random for each benchmark. In the granularity experiments, the work is broken down into successively smaller granularities so that as the granularity of the transactions becomes finer, transactions contain fewer instructions but the total number of transactions required to complete the work increases proportionately. While TransPlant provides two modes of conflict modeling, a high mode in which the distance between pairs of load/store operations to the same cache line is maximized and a random mode where this distance is randomized, only the random mode is used for the granularity experiments. Finally, it should be noted that since transactional work is calculated on a per-thread basis, trends can be compared across a varying number of processing elements, however the raw total cycle counts will differ based upon the number of threads. As such, all of the results for the synthetic benchmarks are reported as the mean of 50 trials.

4. SCHEDULING AND DVFS FOR IMPROVED POWER-PERFORMANCE

The discussion in Section 2 suggests that aborts and stalls have a large impact on the power and performance of many of the benchmarks. If true then there should be a net power-performance gain by avoiding time-intensive aborts and stalls. The first proposed policy leverages dynamic voltage and frequency scaling (DVFS) to decrease the amount of time processing elements are stalled during a NACK. A second policy, based on transaction scheduling, is proposed that utilizes a transaction's current conflict density [9] and its past performance to determine whether a transaction should be preemptively stalled, reducing a program's contention. A further extension, clock gating, is used to reduce the dynamic power of the stalled transaction.

4.1 DVFS FOR IMPROVED THROUGHPUT

Dynamic voltage and frequency scaling (DVFS) was introduced [16] as a means to reduce system power by dynamically controlling the voltage and frequency of PEs based on the system load. DVFS can be implemented at many levels within a system – in the microarchitecture [17], the operating system [11], or at the compiler level [8]. In this work, the power controller is modeled as Intel's FT controller and embedded in the microarchitecture.

On each 2μs probe interval, the conflict manager is queried. If a stall is detected, the DVFS manager is invoked and the stalled core's frequency is decreased by 266MHz while the stalling core frequency is increased by 133MHz until the upper and lower bounds are reached, at 2.93GHz and 1.07GHz, respectively. If there are multiple stalled transactions residing on multiple processing elements, then the processor frequency is increased an additional step for each stalled processing element. On a successful commit, the power manager is preempted and all processing elements are returned to their default operating frequency. In the event of an abort, the process is repeated. However, if the abort count exceeds some allowable threshold, the aborted processing element is put into an idle state. While in this state, the core's clocks are gated (phase locked loops are disabled) and its caches flushed. The aborting processing element is then assigned to the highest performance state. The processing element remains at this frequency unless the chip-wide power approaches its threshold or unless there is a thermal emergency. On a successful commit, it returns to its default operating frequency and sends a signal to wake the idle processing element. By relaxing the contention between the transactions and exploiting the newly available slack power, total throughput is increased while maintaining or reducing average chip-wide and per-processing element power beneath the package’s allowed electrical and thermal limits.

4.2 DVFS RESULTS

Figure 3 shows the ED2P (Et) of the transactional execution normalized to the baseline of each implementation when using the dynamic voltage and frequency scaling scheme described in Section 4.1. As can be seen from the figure, the proposed scheme improves the ED2P by 8% for eager-eager, 7% for eager-lazy, and 7% for lazy-lazy. Because only one processor is allowed to have an up-scaled frequency and multiple processors can be down-scaled, much of the improvement comes from a reduction in energy consumption. There is a greater improvement in the ED2P for benchmarks that spend long periods of time with multiple processors in a NACK'd state such as bayes, kmeans, and labyrinth, but the reasons for the improvements can be applied to the remaining benchmarks.

Bayes: This is the longest running benchmark and is comprised of very large critical sections, averaging 87k instructions. However, the parent thread has a long setup time and skewes cycle calculations, making it appear that there is very little contention. Once the program reaches the parallel regions, bayes is highly contentious. The read and write sets, while large, are small relative to the transaction size with very few circular conflicts. This composition allows the eager conflict detection scheme to resolve most conflicts through NACKs, which benefits from the DVFS policy. While the eager schemes can NACK while waiting for a potentially conflicting address, lazy schemes only check for conflicts when a transaction commits. The DVFS policy considers contention for the commit bus as a NACK, which along with the abort policy is why there is moderate improvement for the lazy conflict detection scheme but less so, for all benchmarks, than the eager conflict detection schemes.

Kmeans: This benchmark has the highest ratio of stall cycles to total cycles of all of the benchmarks. The DVFS policy reduces the average power of eager-eager by 13%, eager-lazy by 14%, and lazy-lazy by 16% while reducing the execution time by 2% in all cases. For kmeans, lazy-lazy obtains more benefit because, for the baseline case, the transactions suffer from contention for the commit bus as well as aborts, extending the execution time and increasing the energy consumption due to rollbacks. The DVFS

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>Total number of threads in the program</td>
</tr>
<tr>
<td>Homogeneity</td>
<td>All threads have the same characteristics</td>
</tr>
<tr>
<td>Tx Granularity</td>
<td>Number of instructions in a transaction</td>
</tr>
<tr>
<td>Tx Stride</td>
<td>Number of instructions between transactions</td>
</tr>
<tr>
<td>Read Set</td>
<td>Number of unique reads in a transaction</td>
</tr>
<tr>
<td>Write Set</td>
<td>Number of unique writes in a transaction</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>Number of global memory accesses</td>
</tr>
<tr>
<td>Conflict Dist.</td>
<td>Distribution of global memory accesses</td>
</tr>
<tr>
<td>Tx Inst. Mix</td>
<td>Instruction mix of transactional section(s)</td>
</tr>
<tr>
<td>Sq Inst. Mix</td>
<td>Instruction mix of sequential section(s)</td>
</tr>
</tbody>
</table>
policy helps by reducing the number of aborts from 412 to 278, decreasing the total energy and the execution time.

Labyrinth: This benchmark consists of very coarse-grain critical regions, averaging almost 400k instructions each, making it the coarsest of the benchmarks and giving it the highest ratio of contentious work – both in terms of aborts and stalls. All of the transactional implementations suffer from multiple rollbacks and stalls. This is the only benchmark where the average power increases with the proposed DVFS. From Figure 1, the average power for the eager conflict detection schemes is 30W. The power is low because rollbacks are expensive in terms of cycles but only require the L1D, L2, and data buses, resulting in lower power density. By scaling the frequency and allowing one thread to complete faster than others many of the aborts are avoided (36%), which has a twofold effect. First, the power density is higher because the pipeline is active more often. Second, because there is not as much time spent performing bookkeeping and rollbacks, the execution time is shortened, which increases the power density but decreases total execution time. However, despite the fact that the average power increases to 62.2W for both eager schemes, it remains low enough that there is never a thermal emergency. The average power for the lazy-lazy platform decreases by 26%. The reduction is primarily due to the decrease in aborts with a small decrease in the average power consumption.

While these three benchmarks show the most improvement, the causes of the reduction in ED2P can be extended to all of the benchmarks, to some degree. The average power of cholesky, fmm, ocean-contiguous, vacation, and water-spatial remains roughly the same for all of the transactional models when using DVFS but the execution time is reduced. For the remaining benchmarks, the average power is reduced along with the execution time (with the exception of labyrinth when using eager conflict detection).

4.3 CONFLICT PROBABILITY

While the DVFS policy discussed in Section 4.1 primarily targeted NACKing transactions, the preemptive stalling policy is targeted at aborting transactions and perceived contention within the transactional system. When a transaction aborts, the contention manager resolves the conflict using the prescribed resolution policy. In the systems discussed in Section 2, the contention manager invokes an exponential backoff policy that prevents a transaction from reissuing using an exponentially increasing interval, up to some maximum. The proposed addition to the contention manager is called when a transaction begins its execution and works in conjunction with the contention manager. A software manager is invoked within the power controller to compute the transaction’s conflict potential for the current iteration, \( C[n] \), given by 
\[
\]

Where the conflict probability, \( C_p \), is:
\[
C_p = \left( \frac{\text{CurrentAborts} + \beta \cdot \text{ActiveTransactions}}{\text{TotalAvailableProcessors}} \right)
\]

\( \alpha \) and \( \beta \) are scaling factors used to weight the effect of the previous conflict potential and to determine how responsive the system is to the number of aborts, respectively. If the conflict potential exceeds some threshold, \( \rho \), then the transaction is preempted and stalled for a brief interval before it attempts to reissue. If the potential is below the threshold, the transaction is allowed to issue normally. When a transaction begins, a software manager is invoked on the on-chip microcontroller to calculate the new conflict probability. In the simulator, this is modeled as seven floating-point instructions that must be completed before the transaction begins. The result is stored in a special register in the calling PE. Clock gating is instant while wake-up from clock gating takes two cycles.

Initial tests showed a minor improvement in the ED2P for lazy conflict detection but almost no change for eager. This was because eager is already adept at avoiding many of the aborts that affect the lazy implementation and, although there was measurable reduction energy for some of the more contentious benchmarks, much of the improvement in lazy came from reduced runtime. To improve the results, clock gating was introduced to work in tandem with the contention manager and the new scheduling policy. The new scheme works the same as above with two modifications. First, when a transaction is stalled, the processor's clocks are halted, effectively setting the dynamic power to zero for the processor on which the transaction is executing. Second, the processor does not wake up after a given interval, instead it waits for another transaction to commit before un-gating occurs.

4.4 CONFLICT PROBABILITY RESULTS

This evaluation is based on the same configurations from Section 4.2 with the addition of the conflict probability scheme. Figure 4 shows the new ED2P using the scheduling enhancement normalized to the base case for each design point. Improvement in the ED2P is seen for all but two benchmarks, genome and raytrace on the lazy-lazy platform. For most of the benchmarks, the reduction in ED2P is the same across all of the transactional implementations, which is due to the lack of contention in the benchmarks. However, the scheduling scheme does reduce both the static and dynamic power of the benchmarks. On average, the proposed scheduling policy produces in 
\[
\text{decrease in dynamic power because of the clock gating scheme. The reduction in energy use and execution times leads to an}
\]

**Figure 3. ED2P (Et) Using DVFS Normalized to Base Case**
reduction of 17% for eager-eager, 17% for eager-lazy, and 10% for lazy-lazy.

Of the benchmarks, *labyrinth* shows the largest improvement across all three implementations while the remainder of the applications show modest improvements. The reason is that *labyrinth* spends more than 98% of its cycles in a NACK or abort state while the other benchmarks typically spend less than 1% of their time in these states. This benchmark consists of very coarse-grain critical regions, averaging almost 400k instructions each, making it the coarsest of the benchmarks and giving it the highest ratio of contentious work. The transactional implementations suffer from multiple rollbacks and stalls and the lazy-lazy scheme suffers from twice as many aborts. All three schemes obtain more than a 2x ED2P improvement for *labyrinth* with the eager conflict detection schemes reaching a 5x improvement. For lazy-lazy, the total runtime remains roughly the same but there are 78% fewer aborts, which means that the dynamic power used for speculative execution of these transactions has been saved through preemption and clock gating. For the eager conflict detection schemes, preemptive stalling provides more than 50% reduction in runtime, which directly reduces ED2P.

The outliers on the lazy conflict detection scheme, *genome* and *raytrace*, are due to the restrictive scheduling algorithm. Although their total energy is lower than in the base cases, the execution time for these benchmarks is increased by several million cycles, which leads to the increased ED2P. For example, *genome* contains nearly 6k critical sections but less than 1% of them result in aborts for the transactional models. The critical sections in *genome* average 2.4k instructions over 4.9k cycles and comprise 70% of the dynamic execution. When the algorithm is applied to *genome*, the number of aborts is reduced from 106 to 87 and the number of NACKs is reduced from 3794 to 2988 but the average number of cycles consumed by each transaction increases to 5.7k. The algorithm does not consider individual transactions, meaning that it only knows that there are *t* active transactions and not the program counter of each transaction. If each available processor has an active transaction and if the abort count increases too quickly, the result is an overly pessimistic representation of the contention, stalling transactions longer than may be necessary. The end result of which is akin to the serialization scheme discussed in Section 4.6. The scaling factors, α and β, are fixed; a feedback mechanism that can shift these for each active process may provide a better prediction mechanism but the philosophy behind both of the proposed designs was to provide a simple implementation with very little runtime overhead.

### 4.5 COMBINING THE SCHEMES

Although both of the policies described in Sections 4.1 and 4.3 are linked with aborts, the DVFS policy relies on NACKs as the primary motivator while the preemptive scheduler relies on perceived contention, allowing the schemes to be used together. Figure 5 shows the ED2P when both DVFS and the probabilistic conflict scheduler are used together. The proposed policies effectively work together, providing a reduction in the ED2P of 19% for eager-eager, 20% for eager-lazy, and 15% for lazy-lazy. The trend is similar to that of Figure 4 because the contention management policy provides the majority of the energy reduction for most of the benchmarks. The exceptions are *barnes*, *bayes*, and *raytrace* (and *genome* for lazy-lazy), which benefit more from the DVFS policy.

### 4.6 MEASURING UP

In this section, the proposed DVFS and scheduling policies are compared with two previous studies. The first comparison is based on work done by Sanyal *et al.* [26]. When a transaction is aborted by a committing transaction, the clocks of the aborted processor are halted and remain so until a local timer expires. The timer value is derived from an equation that takes into account the abort count and how long the blocked processor has been gated.
For the experiments presented here, the model is ideal – meaning that the structures proposed in their work are not modeled at the microarchitecture level and the delay algorithm is able to complete instantly. It should be noted that the original paper used an analytical model to derive results based on memory traces, not integrated functional and timing models. The second comparison is from Moreshet et al. [21] who proposed a serialization algorithm for power-savings in hardware transactional memory. When a conflict is detected and a transaction is forced to abort, instead of reissuing the transaction it is placed in a queue until a successful commit is detected at which point it is reissued. When the queue is empty, the system returns to its default state. For their work, the authors only reported the power of the memory subsystem; the results reported here are for the entire processor and main memory.

Table 5 provides the results for both the gating (gating) and serialization (serial) schemes along with the proposed DVFS and scheduling (DVFS+CS) policies proposed in this paper. Clock gating alone does not noticeably reduce the ED2P for most of the benchmarks. Although the average power of the benchmarks is reduced by an average of 0.9% for eager-eager, 0.7% for eager-laZY, and 1.1% for lazy-lazy, the execution time is increased as well, offsetting any benefit. The exceptions are kmeans and labyrinth. The ED2P for kmeans is an improvement over DVFS+CS using clock gating and is explained in the discussion in Section 4.4. For labyrinth, the average power increases by 118% for eager-eager and eager-lazy but whose execution time is decreased by 37%, resulting in a net loss in the power-performance domain. While the gating algorithm can save some energy in a hardware transactional memory system, it has the drawback of limiting the performance. For the serialization algorithm, the results are much the same (note that kmeans and labyrinth on the lazy-lazy system would not complete). Although there is a slight reduction (less than 1% on average) in the average power for most of the benchmarks, as with the clock gating method most of the reduction is offset by increased execution time.

The combined policies proposed in this paper provide between 21-30% improvement in ED2P reduction relative to clock gating and serialization for eager conflict detection and 12-22% for lazy conflict detection. It is clear that for transactional programs with an abundance of contention, serialization and clock gating cannot improve the power and performance jointly and both the DVFS method and the contention prediction algorithm proposed in Sections 4.1 and 4.3 provide superior results. If the future of transactional memory is to improve the efficiency of parallel programming, then it can be expected that highly optimized programs like cholesky and ocean will not be the norm and programs are more likely to resemble some of the STAMP benchmarks. Regardless, to highlight the effect that the proposed methods have on a range of transactional memory program behavior, synthetic benchmarks are needed.

4.7 SYNTHETIC WORKLOADS

This section provides an analysis of the power and performance of the different hardware transactional memory systems using synthetic benchmarks. Synthetic benchmarks [1] are miniature programs for use in early design evaluations. The advantage of synthetic benchmarks is that they can be used when the simulation time of real benchmarks is prohibitively long or for design space evaluation where no suitable benchmark exists, as is the case for this research. The benchmarks for this analysis are derived by employing a parameterized form of workload synthesis using the TransPlant [23] tool.

For these experiments, the transactional granularity is scaled by powers of 2 beginning with 8 instructions and continuing to 128k instructions; the transaction stride, the distance between transactions, is equal to the transaction size so that the static number of transactional and sequential instructions remains equal. Memory accesses are modeled as circular arrays. On a per-thread basis, there is no reuse outside of the transaction that first references a specific location, ensuring that a single transaction in each thread can only interfere with a single transaction in another thread. For example in a program with n threads, TX1-A can interfere with TX2-A, TX3-A, …, and TXn-1-A but never with TXn-B, where n is the thread ID.

4.8 SYNTHETIC WORKLOAD RESULTS

Figure 6 shows the ED2P of the synthetic benchmarks normalized to the base case for each example as transaction granularity increases. Immediately apparent is the abrupt shift in the trend at the 4k granularity. The reason for this relates to the average power of the transactional models. On the base system, the eager-lazy model has the highest average power out of the three designs – peaking at 70W. At 4k, there is an abrupt drop in the average power for the two eager conflict detection schemes; eager-eager drops by 54% to 32W and eager-lazy drops by 60% to 41W while
the average power for the lazy versioning scheme increases by an average of 6% until the granularity reaches 32k at which point it begins slowly decreasing. A breakdown of the transactional cycles, shown in Figure 7, is needed to further explain these phenomena.

The top graph in Figure 7 shows the relative execution time for the eager-eager system. Referring back to Figure 6, the reduction in ED2P remains roughly flat until the transaction size reaches 4k and is a result of reduced power consumption from the conflict-aware scheduling policy; execution time remains mostly unchanged. At 4k, the benchmarks begin to spend more and more time in a NACKd state and the system is able to avoid aborting by stallng the processor, which itself reduces the average by 54% to 32W as pipelines become idle. DVFS and the conflict-aware scheduling policy are able to further reduce the power for all three schemes by an additional 60% and decrease the execution time by 54% to 16k occurrences. Beginning at 8k, the aborts become so persistent that the power manager essentially halts all but one processing element. Note that this is a different situation from the one in Section 4.4 where the scheduler was unnecessarily penalizing raytrace and genome but the result is the same – reduced power consumption relative to the baseline system but increased runtime, which increases the ED2P.

5. RELATED WORK

The new work discussed in this paper is related to prior work on power management techniques. In [15], the authors show how processes can be mapped onto a variable number of processing elements while sleeping unused ones and guaranteeing some minimum performance threshold. Isci et al. [10] proposed managing per-core voltage and frequency levels based on application behavior to manage total chip power. [29] proposed using linear programming to identify the optimal voltage and frequency levels for each core in a CMP to increase throughput and reduce ED2P. Rangan et al. [24] show how threads can migrate between different PEs to achieve nearly the same power reduction as per-core DVFS while [13] propose an algorithm to improve fairness between co-executing threads. The drawback with all of these approaches is that they require online profiling of the runtime environment and computationally-intensive algorithms to meet their desired goals. The proposals outlined in this paper are less intrusive and achieve excellent results with minimal overhead.

There has been some recent research into the energy use of transactional memory for embedded and multiprocessor systems. Ferri recently proposed unifying the L1 and transactional cache in an embedded system and showed that using a small victim cache to reduce the pressure from sharing improved the energy-delay product [5]. Moreshet et al. [21] showed that hardware transactional memory systems can be more energy efficient than locks in the absence of contention. They then proposed a serialization mechanism for HTMs and showed that it lowered energy consumption for their microbenchmarks. However, their work relied on four non-contentious SPLASH-2 benchmarks and one in-house microbenchmark, making it difficult to draw any meaningful conclusions. Using an analytical model to estimate the additional power for an Alpha 21264, Sanyal et al. [26] proposed a technique for clock gating on an abort using TCC. Neither of these proposals exploit the feedback inherently available in transactional memory like the scheduler proposed by Yoo and Lee [34] who proposed an adaptive scheduler using parallelism feedback and showed speedups of almost 2x for his selected benchmarks. While the energy reduction scheme proposed here has some similarities to previous work, it differs in two main regards. First, it abstracts the differences in the hardware, allowing for an almost direct comparison of power for different conflict detection and version management schemes. Secondly, the proposed method does not assume that contentious transactions should be serialized like [21] and is much less complicated than [26].
To reduce system power and increase throughput when transactions are in a NACK state, a dynamic frequency and scaling system is proposed. By increasing the clock frequency of NACKing PEs and throttling the clock frequency of NACK’d PEs, the number of stall and abort cycles is reduced, increasing throughput. The PEs in low-power states serve to reduce or maintain the average system power. Together these effects serve to reduce the system ED2P, or improve the power-performance of the system by 8% for eager-eager, 7% for eager-lazy, and 7% for lazy-lazy. To limit the number of aborts a program experiences and control power usage during these periods, a new transaction scheduling policy is proposed that utilizes a transaction's current and past conflict density to determine whether a transaction should be preemptively stalled and its clock disabled. This technique provides an average reduction in the ED2P of 17% for eager-eager, 17% for eager-lazy, and 10% for lazy-lazy. When applied together, the DVFS and scheduling policies provide a reduction in the ED2P of 19% for eager-eager, 20% for eager-lazy, and 15% for lazy-lazy. More importantly, the benchmarks with greater contention (labyrinth) obtained even greater reductions – up to 76%. These results show the potential for manipulating clock frequencies for transactional memory for improved throughput while maintaining or reducing local and chip-wide power budgets and lay the foundation for future work in aggressive power management strategies for multithreaded workloads in the many-core era.

7. ACKNOWLEDGEMENTS
This work was supported by NSF grants CNS-0834288 and CCF-0845721 (CAREER) and SRC grant 2008-HJ-1798 and by three IBM Faculty Awards. The authors would also like to thank the UF HPC Center for providing computational resources and the anonymous reviewers for their valuable suggestions.

8. REFERENCES


