Clock Gate on Abort: Towards Energy-Efficient Hardware Transactional Memory

Sutirtha Sanyal[1], Sourav Roy[2], Adrian Cristal[1], Osman S. Unsal[1], Mateo Valero[1]
[1]Barcelona Supercomputing Center, Barcelona, Spain; [2]Freescale Semiconductors, India
[1]{sutirtha.sanyal,adrian.cristal,osman.unsal,mateo.valero}@bsc.es;[2]sourav.roy@freescale.com

***Abstract*—Transactional Memory (TM) is an emerging technology which promises to make parallel programming easier compared to earlier lock based approaches. However, as with any form of speculation, Transactional Memory too wastes a considerable amount of energy when the speculation goes wrong and transaction aborts. For Transactional Memory this wastage will typically be quite high because programmer will often mark a large portion of the code to be executed transactionally[4].**

We are proposing to turn-off a processor dynamically by gating all its clocks, whenever any transaction running in it is aborted. We have described a novel protocol which can be used in the Scalable-TCC like Hardware Transactional Memory systems. Also in the protocol we are proposing a gating-aware contention management policy to set the duration of the clock gating period precisely so that both performance and energy can be improved.

With our proposal we got an average 19% savings in the total consumed energy and even an average speed-up of 4%.

***Keywords*: Hardware Transactional Memory, Clock Gating, Distributed Shared Memory architecture, Transaction Abort, Low-power architecture.**

I. INTRODUCTION

Recent emergence of multi-core processors has generated interest in writing concurrent programs for mainstream applications. However, writing parallel programs for shared memory multiprocessors is difficult. The classical lock-based way of handling this issue is error-prone and non-composable.

The Transactional Memory[10], [5], [14], [12], [7] is a technology where programmers just need to wrap a portion of the code in what is known as a transaction. It is then the responsibility of the TM framework to enforce Atomicity and Isolation properties to guarantee correctness. Scalability issue is also handled by the runtime. Applications can scale too if the system has sufficient number of processors to run all the threads simultaneously and conflict rate among threads is low.

However Transactional Memory relies on a bulk amount of speculation to deliver optimistic concurrency control. If that speculation goes wrong, a processor needs to roll-back to its check-pointed architectural state and start execution afresh. This is known as "Abort" in the Transactional Memory.

If a thread undergoes abort for n number of times before committing its transaction, evidently all the work done in the previous n executions but the last one are completely wasted and so does the energy spent in doing that. We name those aborts as *Futile Abort*. Theoretically a parallel program with transaction support can still produce correct output if all those *Futile Abort*s are eliminated. However, it will not be possible to predict whether an abort is going to be futile or fruitful. Therefore a scheme comprising of both contention management and energy saving techniques is required where some of the wasted execution time and energy can be saved.

Ferri et al. has proposed a basic scheme to reduce the energy consumption for transactional memory[8]. However it assumes a simple hardware transactional memory with explicit transactional cache which is suitable only for embedded systems. Li and Martinez proposed "Thrifty Barrier"[13], which applies the energy-saving techniques at the synchronization points.

Our major contributions are:

- We have used a well-known energy saving technique, *clock gating*[1], [15] in the context of Hardware Transactional Memory (HTM) to save energy. Processor can be pushed into this state immediately by zeroing out all its clocks (reducing dynamic power consumption to zero). Processor can also be pulled out from this state instantaneously. We have proposed a novel protocol which gates processors dynamically on each abort and un-gates it depending on the number of aborts it has suffered and the state of its conflicting transaction.
- We have sketched out a simple contention management scheme which determines the period for clock gating a processor. This scheme has been shown to produce significant savings in the energy consumption. Execution time reduces as well because of the efficient conflict management among threads.
- A comprehensive power model has been developed for Alpha 21264 in 65nm. This model enables us to quantify the benefit of our proposed scheme.

II. SCALABLE TCC PROTOCOL

Our base line system is modeled after the Scalable TCC[5]. The Scalable TCC protocol relies on the distributed shared memory structure. In the system, multiple directories are present which maps different segments of the physical memory. Processors load data from directories and then update them speculatively in their own private L1 data cache. A centralized token vendor generates a token id when a processor reaches the commit stage. This token id (TID) acts as a timestamp for the transaction commit. When two conflicting transactions attempt to commit in the same directory, they get serialized based on their timestamp value. The older transaction will possess low TID and will be able to commit first. After a successful validation phase where no invalidation message is received, a processor can start committing. All speculative stores are committed in the directories and sharer of those cache-lines are sent invalidations.

III. PROPOSED ARCHITECTURAL CHANGES IN THE DIRECTORY

As discussed in the previous section, a transaction gets aborted only when a cache line that it has read in its local L1 speculatively, gets committed in a directory by some other thread. The abort message can come when the abortee is still in the middle of a transaction or spinning in its own commit instruction to get access to directories. Whenever this message comes, the transaction is aborted immediately and retried.

With our proposal we are gating all the clocks of the processor whenever it is aborted by any committing thread. After a processor is gated, it is to be ungated again by applying an algorithm which is described in the section V. However, to use this algorithm, some relevant information need to be stored in the directory before gating any processor.

Proc Id	Aborter Proc Id	Aborter Tx Id	Abort Counter	Renew Counter	Gate Timer	Off
P_0						
P_1						
P_2						
P_3						
⋮						
P_N						

Fig. 1. Proposed Additional table in each Directory

Figure 1 shows the additional fields we are proposing in the directory structure. A new table is added in each directory to hold **a)** Aborter Processor information **b)** The id of the Transaction which is responsible for the abort **c)** Abort Counter **d)** Renew Counter **e)** A Timer which keeps the count of the clock cycles the corresponding processor was off and **f)** Current state of the processor (OFF). Aborter processor information stores the processor id which aborted the victim processor in this directory. This field is set whenever the directory sends an invalidation to any processor. Along with the aborter information, the transaction which is causing the abort is also tracked and stored. A transaction is identified by the program counter value of the instruction which started the transaction. Therefore 64 bits for the Transaction id field is sufficient. The directory in which the abort takes place is responsible for querying and storing this information from the committing processor.

Abort count is an up-counter which stores the number of aborts the current running transaction has suffered till now. Whenever the processor commits, this field is reset to 0. A 8 bit counter may be sufficient for all practical applications. However, if a transaction aborts more than 255 times, abort count field will be saturated. Renew count field stores the number of times a processor has been gated at the current abort level. Use of this field is detailed in the section V.

The counter which counts the number of clock cycles a processor was gated for is stored in the next field. This counter is preset to a positive value every time either the Abort count field or the Renew count field is changed. After that, it is decremented in every directory-local clock tick. Once it hits 0, the gating period for that processor expires. The duration for which the processor will be turned off is discussed in the section VI. Finally one bit is kept corresponding with every processor entry to store its current state. If this bit is 1, the processor is gated. By default this bit will be set to 0.

IV. MODELING ENERGY REDUCTION AND EFFECT ON EXECUTION TIME

Let us suppose that with clock-gating the execution of the parallel section of an application takes N_2 units of time. Here we have measured the parallel execution time as the difference between the end time of the last transaction to the start time of the first transaction. Then the total energy consumed during the gated parallel execution E_g is given by

$$\begin{aligned} E_g \quad &= [N_2 * p - \textstyle\sum_{i=1}^{p} X_i * i] * P_{run} + \\ &\textstyle\sum_{i=1}^{p} X_i * i * \alpha_i * P_{miss} + \\ &\textstyle\sum_{i=1}^{p} X_i * i * \beta_i * P_{commit} + \\ &\textstyle\sum_{i=1}^{p} X_i * i * (1 - \alpha_i - \beta_i) * P_{gate} \qquad (1) \end{aligned}$$

Where X_i denotes the total amount of time during the whole execution when exactly i number of processors were "gated or waiting for a cache miss or performing commit". p is the total number of processors. α_i denotes

the proportion of processors within i number of processors which are serving cache miss in the interval X_i. β_i denotes the proportion of processors performing commit in the same interval.

P_{run} is the dynamic run mode power. P_{miss} is the power consumed while serving a L1 miss. P_{commit} is the power a processor dissipates during commit. In all these power numbers contribution due to leakage is already included. P_{gate} denotes the leakage power when the processor clock is gated.

In the (1), the first term corresponds to the energy consumed during the parallel execution when some or all of the processors are consuming full run mode power. To obtain this, we subtract all X_i intervals multiplied by the number of processors consuming low power from the total run time multiplied by the total number of processors. Next two terms add up the energy contribution from processors which are stalled because of cache miss or because of commit, respectively. Finally, the last term accounts for the leakage power which is present even if we gate the clock.

X_i, α_i and β_i are derived in the following way:

$$X_i = \sum_{k=1}^{l} \Delta_k^i \quad (2)$$

$$\alpha_i = \frac{\sum_{k=1}^{l} n_{m_k}^i \Delta_k^i}{i * X_i} \quad (3)$$

$$\beta_i = \frac{\sum_{k=1}^{l} n_{c_k}^i \Delta_k^i}{i * X_i} \quad (4)$$

Here l is the total number of intervals during the gated execution when exactly i number of processors were "gated or waiting for a cache miss or performing commit". So, X_i is the sum of length of all such intervals, denoted by Δ_k^i. α_i is the ratio of contribution from processors stalled because of cache misses in X_i to the number of all processors accounted in X_i. To find it, we calculate a weighted sum on processors serving cache misses. Therefore, $n_{m_k}^i$ denotes the number of processors stalled because of cache misses in the kth instance of an interval which contributed to X_i. Similarly $n_{c_k}^i$ denotes the number of processors doing commit.

For the ungated parallel execution one can obtain an equation like (1) for the total consumed energy, E_{ug}:

$$\begin{aligned} E_{ug} &= [N_1 * p - \textstyle\sum_{i=1}^{p} Y_i * i] * P_{run} + \\ &\textstyle\sum_{i=1}^{p} Y_i * i * \delta_i * P_{miss} + \\ &\textstyle\sum_{i=1}^{p} Y_i * i * (1 - \delta_i) * P_{commit} \end{aligned} \quad (5)$$

N_1 denotes the execution time of the parallel section without gating any processor. Y_i denotes the total amount of time in the complete execution when exactly i number of processors are "waiting for a cache miss or performing commit". Here δ_i is analogous to α_i and obtained in the same manner. Since no processors are gated, therefore if a processor is not stalled for a cache miss and still accounted within Y_i, then it must be because of commit-stall. The factor $(1-\delta_i)$ corresponds to that. An equivalent way to compute the total energy consumption is to track and sum up the individual contribution of each processor in each state.

Therefore the energy savings can be expressed as:

$$EnergyReduction = \frac{E_{ug}}{E_g} \quad (6)$$

Savings in the average power dissipation is given by:

$$AveragePowerReduction = \frac{E_{ug}}{E_g}\frac{N_2}{N_1} \quad (7)$$

In (1) and (5) we have assumed that leakage is present since power is not gated. However it is possible to gate power too in a fine-grained manner along with the clock gating using technologies like "State Retention Power Gating"[11].

Total parallel execution time decreases too along with the decrement in the consumed energy. This happens as a direct consequence of the gating-aware contention management scheme. In normal case, when no clock gating is applied, a processor can waste cycles by spinning at the commit instruction while some other processor is committing in the contended directory. The spinning processor ultimately gets aborted by the committing processor. However, that happens only when the committing processor finally commits the conflicting cache line in the directory. Therefore, till that moment all processors spinning at their commit instruction are wasting execution time by doing a futile spin, because they get aborted eventually. However, as we apply contention management along with the clock gating, it happens that a thread is delayed by a small amount of time which results in the two commits being skewed. In that scenario the time wasted in the futile spin is eliminated. Because the committing thread will abort the other conflicting thread while it is still in the middle of a transaction before reaching to the commit point. However sometime a processor will be gated for more than the duration which is required to maintain the same performance level. In that case it will result in a slowdown.

V. Proposed protocol for Gating/Ungating processors

In this section we will now present the complete algorithm used for gating and ungating a processor. The following algorithm assumes the baseline directory structure as per the scalable TCC protocol and our proposed changes in addition as described in the section III.

As shown in the Figure 2(a), there are four processors and corresponding four directories. An arrow between a processor and a directory denotes that the processor has read data from that directory and modified it during the transaction.

Now let us suppose that P0 has reached its commit instruction and started commit. Also let us assume that processor P1 and P2 are in the middle of a transaction at that instance and P3 has reached its commit point.

Now P0 has started the commit process earlier. As a result, it possesses a TID which is lower than P3. Therefore P0 can start committing into the directories D0, D1 and D2. Meanwhile P3 has to spin till it can get an access into the directory D0 which is currently servicing P0.

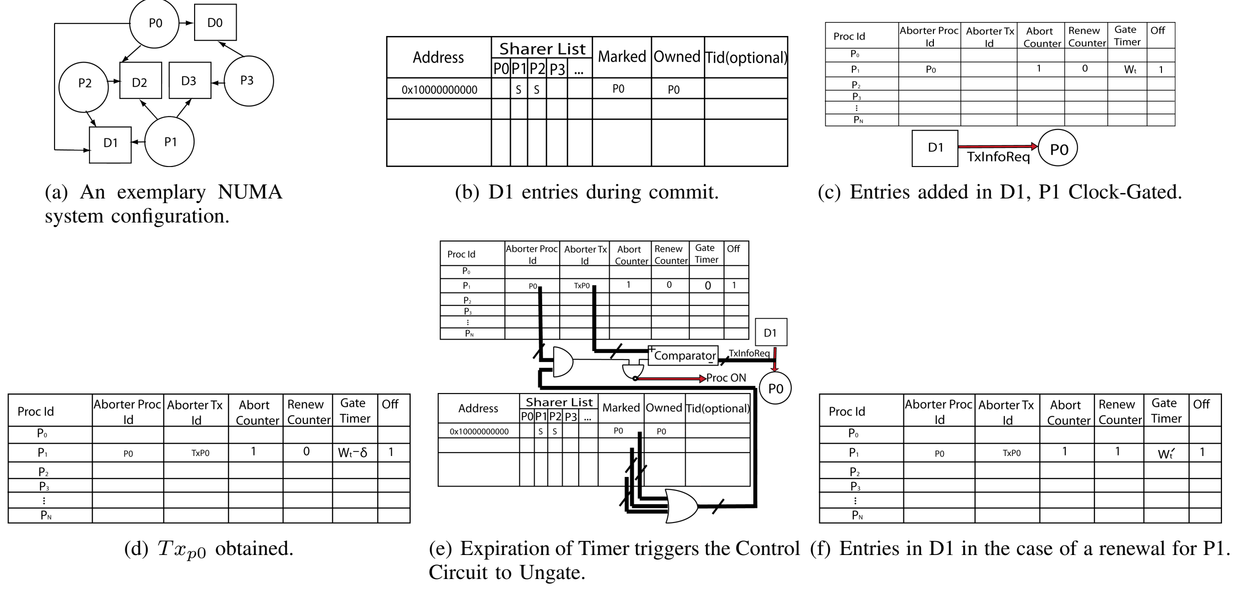
When P0 flushes its writeset into the directories, each directory updates the owner information of the cache line which gets committed. Earlier P0 was marked as the Sharer of the line and its new coherence state is the "Owner" (Figure 2(b)). Other processors marked as sharer (P1 and P2) will be sent invalidations.

At this point, we apply our protocol to *clock gate* all the processors which are getting aborted because of the commit from P0. The directory logs the aborter thread id in the additional table. As shown in the Figure 2(c), since P0 is aborting P1, the aborter proc id contains P0. The Abort count is set to 1 and Renew count is set to 0. Similar information are logged into the entries for P2 (in D1) and P3 (in D0). Immediately after entering these values, the directory starts the timer W_t and sends the "Stop Clock" signal to the victim processor P1. The value of the timer W_t is determined by the gating-aware contention management scheme which we present in the next section. When the "Stop Clock" signal arrives, a processor stops fetching any further instruction and goes to standby mode after finishing the execution of the current in-flight instruction. However the directory still does not have the information about the transaction which is responsible for the abort. This information is required to be stored in the "Aborter Tx Id" field. To obtain this information, directory sends out a message designated as "TxInfoReq" to the current committing processor. On receiving this message the processor replies with the id of the transaction that it is executing. The id is a program counter value which started the transaction. After getting the reply from P0, directory D1 stores it in the table as shown in the Figure 2(d).

After the timer expires, the gated processor will get a chance to turn on its clock. However, in the protocol we are proposing that if **a)** The aborter thread is still present in that directory and **b)** If the aborter thread is executing the same transaction which earlier killed the abortee transaction, then instead of sending the "on" signal, directory simply renews its gating time and loads a new timer value in the "gating timer" field. To check a and b, we are proposing the circuit shown in the Figure 2(e). In this circuit diagram, a bitwise "OR" is performed between all the processor ids which have expressed their intention to commit by marking the "Marked" bit. If none of these processor ids match with the id which was responsible for abort, then it will be prudent to turn on the victim processor. However, if the enemy processor is present in the directory, we further check to find out the transaction it is currently executing. This is done again using a "TxInfoReq" message. After receiving the reply, the answer is compared with the "Aborter TX" field stored earlier. If this further check turns out to be negative, then also the gated processor is sent an "on" command. In the case the processor P0 has itself been turned off by some other processor, the reply to the "TxInfoReq" message will be null and therefore the comparator output will be zero, turning the victim processor on. However, if this further check turns out to be positive because the enemy processor is executing the same transaction which earlier invalidated the victim, the gating period is extended. A fresh value for the timer, W_t' is loaded in the table. After a "renewal" the directory keeps track of the number of times a stopped processor renewed its gating period (Figure 2(f)). As the number of renewals goes up so does the initial value of the timer. Renew count field is reset to 0 whenever Abort count field is incremented. Abort count field is reset to 0 whenever a thread commits. The circuit will take multiple cycles to generate the "on" command because of the high fan-in bitwise "OR". That will extend the clock gating period further by a small amount of time.

The "on" command is delivered to the output of the main pll of the processor which is assumed to be always running. When the "on" signal goes high, the output from the main pll becomes available to all other plls like core pll, io pll and cache plls. After this wake-up, the processor needs to do a "Self Abort" of the transaction it was executing at the time of freeze. This is required to maintain the correctness of the program. However, this "Self Abort" event is not tracked by any of the directories.

It should be noted that a directory turns off or turns on a processor based on its local knowledge about the abort behavior of the processor. Therefore it may happen that a processor which is marked as "off" in one directory is marked as "on" in some other directory. However that will pose no problem since once turned off, a processor will not issue any load/store. On the other hand, it may happen that a processor has been turned on by a directory while it is still marked as "off" in some other directory. In this scenario, if any load/store request comes from a processor which is marked as off, directory assumes that it has been turned on by some other directory. Then



(a) An exemplary NUMA system configuration.

(b) D1 entries during commit.

(c) Entries added in D1, P1 Clock-Gated.

(d) Tx_{p0} obtained.

(e) Expiration of Timer triggers the Control Circuit to Ungate.

(f) Entries in D1 in the case of a renewal for P1.

Fig. 2. Protocol States for Gating/Ungating a Processor

it resets the "OFF" bit as well in its local table. This protocol is deadlock-free. A cycle of dependencies can not form because, once a processor is gated it can not cause abort to any other processor.

VI. Gating-Aware Contention Management Scheme

Several contention management policies exist for the Transactional Memory[17]. All of them have their respective pros and cons. In this section we are presenting a simple contention management policy which resembles polite back-off and is also aware that the processor is going through a clock gating phase because of an abort. We are proposing that if a processor executing a transaction is aborted by another transaction, then the victim processor should back off and as a consequence should be clock gated for an interval W_t where:

$$W_t = W_0(2^{\lceil \lg N_a \rceil} + 2^{\lceil \lg N_r \rceil}) \tag{8}$$

In (8) N_a denotes the Abort count and N_r denotes the Renew count. The ceiled logarithms ensure that the gating period models a "Staircase Function" where the gating timer value increases only after the abort count (or the renew count when the abort count remains same) exceeds a specific range. However, unlike conventional staircase function, this has discontinuities at exponentially spaced intervals. This results in a situation where the gating period is moderately high for highly-conflicting applications to get a reasonable savings in the energy. On the other hand the protocol described in the previous section biases slightly more on "turning on" the processor. That bias effectively balances out the loss of performance due to gating. However, if both the abort count and the renew count are low for any application, a processor will not be gated substantially. For those applications, performance will remain close to the base-line performance and the amount of energy savings will also be limited. Please note that a basic contention management scheme like exponential polite back-off does incur significant performance penalty for highly contentious applications and hence not applicable in such cases.

The constant factor W_0 has a first order significance. For large number of processors, this constant should be small since the number of aborts will be high. For small-scale systems this constant should be preset to a high value. The firmware which calculates W_t, can also preset W_0. Other contention management schemes based on the momentum of the transaction at the time of abort are possible. We have left them as future works.

VII. Alpha 21264 power model in 65nm

In this section we build an analytical power model for the Alpha 21264 in 65nm technology based on practical assumptions and simulations. The Alpha 21264 consists of 64KB L1 instruction and data caches, but does not have any L2 cache. It was last fabricated in 0.35 micron technology. We assume that the Alpha 21264 is scaled in technology without any change in functionality, consistent with the M5 simulation model.

The power consumption of the processor changes based on its operations. During cache miss and commits the processor will consume lesser power, since the core is idle during such operations and only the data cache

and I/O interfaces are active. It is to be noted that though power reduces, energy consumption increases due to the long waiting time for cache miss or commit to complete. The power distribution for the original Alpha 21264, that are of interest to us, is as follows[9] :

- Caches 15%
- Clock 32%
- I/O 5%
- Leakage 2.8%

In standard 65nm technology, without any optimization, the leakage component in a typical superscalar microprocessor is 30-40% of the total power. However with the advent of leakage saving techniques like usage of high-Vt cells in non-critical timing paths, and stacked transistors the leakage can be controlled significantly. In standby mode, leakage can be further reduced with the help of sleep transistors and body biasing techniques. However in this work we are only concerned with leakage in active mode. In active mode, with the use of above mentioned techniques, the leakage power can be safely reduced to less than 20%. Recent product datasheets and publications[16], [3] also substantiate that leakage ratio can be assumed to be 20% in 65nm technology. When the processor is clock gated, apart from leakage, only the PLL is active. The PLL consumes few milliWatts, whereas the leakage power in 65nm technology is several Watts. Hence its contribution is negligible. Therefore we conclude that in clock gated state, the processor consumes 20% of the total power.

The data cache that supports TCC consumes more power than a normal data cache. We used CACTI[18] to find out the power increase due to additional RW bits. Figure 3 shows the normalized power of a TCC data cache, assuming the normal data cache consumes 100 units of power. The RW bit resolution is varied from the cache-line size of 64B to 1B for various cache sizes in 65nm technology. For a 64KB cache with word level(2B) state tracking the power increase is limited to 5%. However the power increases considerably due to the addition of store address FIFO, commit controller and other control circuitry. We used power estimation tools (PowerTheater) to estimate power on representative RTL of write buffers (implemented with standard flip-flops) of various depths and also the commit controller. For a 64KB cache with 64B line, we require a store address FIFO of 1024 words each of 10 bits. Adding up the individual contributions, the power of the entire data cache that supports TCC is, conservatively, 1.5 times that of the normal data cache. In Alpha, 15% of the power is contributed by caches, among which data cache contributes 10% of the total power. Hence the TCC data cache consumes $1.5 * 10 = 15\%$ of the total power.

In 0.35 micron technology, the leakage power contribution is negligible. Hence the power distribution of individual modules in 0.35 micron technology can be directly applied to the dynamic power in 65nm technology. During commit the core is idle. But the data cache including the store address FIFO and commit controller, I/O interfaces and their clocks are active. As discussed before, the contribution of the TCC data cache to the dynamic power is 0.15. From the original Alpha 21264 published results the I/O interfaces and the clocks to the data cache and interfaces contribute 0.05 and 0.10 respectively. The fraction of total power consumed during commit is then given by,

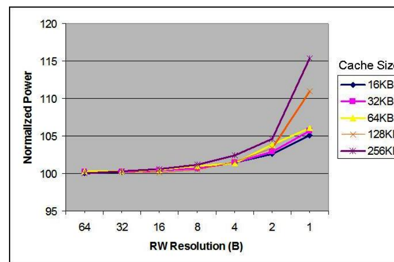


Fig. 3. Power Consumption of data cache supporting TCC

$$\begin{aligned}\text{Commit Power} &= 0.2 + 0.8 * (0.15 + 0.05 + 0.1) \\ &= 0.44\end{aligned}$$

During cache miss the core is stalled. After miss is detected, the cache (instruction or data) writes the new tag corresponding to the miss address in the tag memory, waits for the miss data to arrive and then writes it in the data memory. Further the data cache frequently performs a dirty line replacement on cache miss, by writing it to the next level of memory. The I/O interfaces are typically active during cache miss. In [6], the typical cache miss dynamic power is measured to be approximately 50% of the cache dynamic power during hits or high activity. This is a conservative average estimate and large miss penalties can further reduce the cache miss power.

$$\begin{aligned}\text{Cache-Miss Power} &= 0.2 + 0.8 * 0.5 * (0.15 + 0.05 + 0.1) \\ &= 0.32\end{aligned}$$

Finally we assume that at synchronization points the processor consumes full run mode power while executing spin-locks. Run mode power is the power consumed while executing normal code or transactions. Based on the above discussion, the power factors consumed during various operations are shown in table I. We are not interested in the absolute power values in this work. However to get a quick estimate of the absolute power values, a simple way is to scale the original Alpha 21264 area (by 0.5 every technology node) and then apply a typical average power density number corresponding to

TABLE I
POWER MODEL OF ALPHA 21264

Operation	Power Factor
Run	1.0
Cache Miss	0.32
Transaction Commit	0.44
Clock Gated	0.20

a superscalar out-of-order microprocessor.

VIII. SIMULATION SETUP AND RESULTS

Simulations are done in a substantially modified version of the M5 full-system simulator[2] simulating Alpha 21264 architecture with added support for a Scalable-TCC system. Table II lists the parameters used in the simulation.

TABLE II
PARAMETERS USED IN THE SIMULATION

Feature	Description
CPU	1-16 single issue in-order cores
L1D	64KB 64 byte line size 2-way associative 1 cycle latency
Interconnect	Common Split-Transaction Bus
Directory	Full-bit vector sharer, 10 cycle latency
Main Memory	1GB, 100 cycle latency, Single Read/Write Port

For preliminary evaluation purpose we have used 3 applications from the STAMP benchmark suite[4]. They are: genome, yada and intruder. Figure 4 reports the total execution time spent inside the parallel section. We have measured it for 4,8 and 16 processors configurations. Data are represented as 3 pairs for these 3 configurations. Each pair contains values corresponding to "without clock-gating" and "with clock-gating" settings. Speed-up number with respect to the ungated version is denoted on top of the bar representing the value with clock-gating. A number n denotes a speed-up of nx times.

As mentioned in the section IV, in most of the cases we have observed speed-ups. However in one case we have observed slowdown. As shown in the Figure 5, moderate to significant energy reductions are noted in all cases. The factor by which the energy consumption reduces is mentioned on top of the bar representing the gated version. For highly-conflicting application like "intrduer", abort rate is high and as a result savings in the energy is also reasonable. On the other-hand for moderately conflicting applications like "yada" and "genome", there are conflicting transactions which are either long or repeated several times inside loops. In those cases the renew counter tends to become high with the abort counter still remaining low. That too results in a large gating window and significant savings in the energy. In Figure 6, we have shown the savings in the average power dissipation. Figure 7 presents a sensitivity

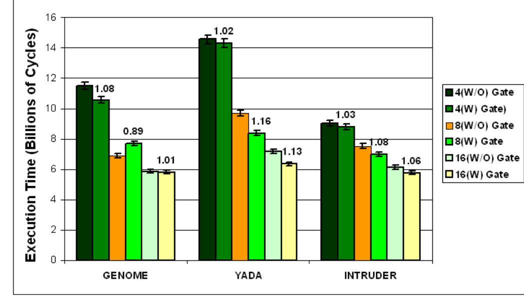


Fig. 4. Total Parallel Execution Time for 3 applications

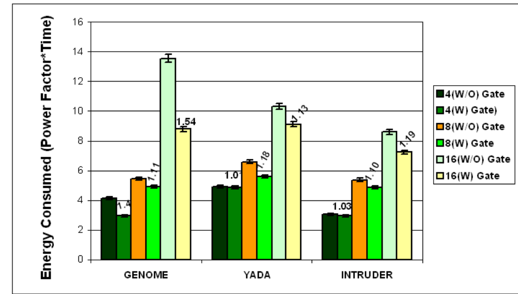


Fig. 5. Energy Consumption W and W/O Clock Gating

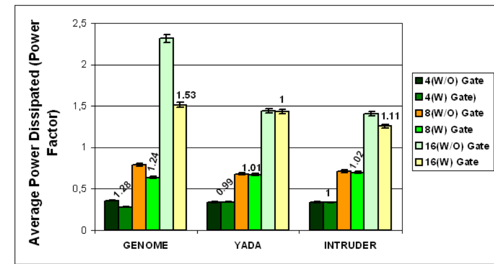


Fig. 6. Average Power Dissipation W and W/O Clock Gating

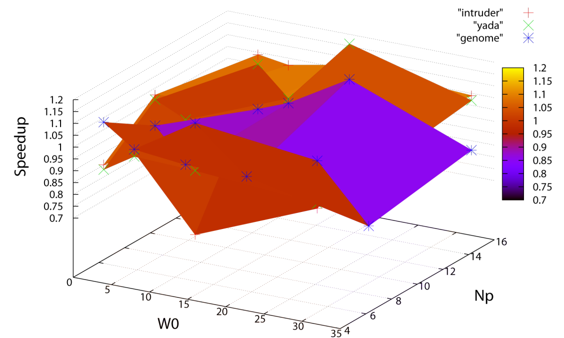


Fig. 7. Speed-Up as a function of W_0 and N_p

analysis of the speed-up as a function of W_0 and the number of processors (N_p). For our experiments, we have used W_0=8 and speed-up is obtained for all the cases (except for genome with 8 threads). As processor number changes, W_0 can further be adjusted to extract more performance. Across these 3 applications and 4,8 and 16 processors cases, we got average speed-up of 4%. Average reduction in the energy consumption is 19%. Reduction in the average power dissipation is 13%.

IX. Conclusion

In this paper, we have sketched out a mechanism which can be used to save energy in the Hardware Transactional Memory. We achieve this by clock gating to halt processors which are wasting energy in transaction abort. In the course of doing so, we have also recovered some wasted execution time by efficient contention management. This resulted in an overall speed-up. As per our knowledge, this is the first work that targets energy efficient HTM in large-scale multi-processor systems. The initial results are very promising and in future we plan to explore several other schemes on a larger suite of applications.

Acknowledgments

This work is supported by the Barcelona Supercomputing Center (Centro Nacional de Supercomputación) and Microsoft Research Lab, Cambridge vide the collaboration contract no TIN2007-60625; by the European Network of Excellence on High-Performance Embedded Architecture and Compilation (HiPEAC) and by the European Commission FP7 project VELOX (216852). Sutirtha Sanyal is also supported by a scholarship from the Government of Catalunya.

References

[1] Clock Gating Recommendations http://www.amd.com/epd/processors/6.32bitproc/x19195/19195.pdf.

[2] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE MICRO*, pages 52–60, 2006.

[3] Ke Cao, Sorin Dobre, and Jiang Hu. Standard cell characterization considering lithography induced variations. In *Proceedings of the Design Automation Conference*, San Francisco, California, 2006.

[4] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. Stamp: Stanford transactional applications for multi-processing. In *IISWC '08: Proceedings of The IEEE International Symposium on Workload Characterization*, September 2008.

[5] H. Chafi, J. Casper, B.D. Carlstrom, A. McDonald, C.C. Minh, W. Baek, C. Kozyrakis, and K. Olukotun. A Scalable, Non-blocking Approach to Transactional Memory. *Proc. of the 13th Intl. Symp. on High Performance Computer Architecture, Phoenix, AZ, Feb*, 2007.

[6] Lokesh Chandra and Sourav Roy. Estimation of energy consumed by software in processor caches. In *Proceedings of the International Symposium on VLSI-DAT*, Hsinchu, Taiwan, 2008.

[7] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid transactional memory. *Proceedings of the 2006 ASPLOS Conference*, 41(11):336–346, 2006.

[8] Cesare Ferri, Amber Viescas, Tali Moreshet, R. Iris Bahar, and Maurice Herlihy. Energy efficient synchronization techniques for embedded architectures. In *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 435–440, New York, NY, USA, 2008. ACM.

[9] Michael K. Gowan, Larry L. Biro, and Daniel B. Jackson. Power considerations in the design of the alpha 21264 microprocessor. In *Proceedings of the Design Automation Conference*, San Francisco, California, 1998.

[10] L. Hammond, V. Wong, M. Chen, B.D. Carlstrom, J.D. Davis, B. Hertzberg, M.K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional Memory Coherence and Consistency. *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)*, 1063(6897/04):20–00.

[11] S. Henzler, G. Georgakos, M. Eireiner, T. Nirschl, C. Pacha, J. Berthold, D. Schmitt-Landsiedel, I.T. AG, and G. Munich-Neubiberg. Dynamic state-retention flip-flop for fine-grained power gating with small design and power overhead. *Solid-State Circuits, IEEE Journal of*, 41(7):1654–1661, 2006.

[12] M. Herlihy and J.E.B. Moss. *Transactional memory: architectural support for lock-free data structures*. ACM New York, NY, USA, 1993.

[13] J. Li, JF Martinez, and MC Huang. The thrifty barrier: energy-aware synchronization in shared-memory multiprocessors. In *High Performance Computer Architecture, 2004. HPCA-10. Proceedings. 10th International Symposium on*, pages 14–23, 2004.

[14] K.E. Moore, J. Bobba, M.J. Moravan, M.D. Hill, and D.A. Wood. LogTM: Log-based transactional memory. *Proc. 12th Annual International Symposium on High Performance Computer Architecture*, 2006.

[15] Alon Naveh, Efraim Rotem, Avi Mendelson, Simcha Gochman, Rajshree Chabukswar, Karthik Krishnan, and Arun Kumar. Power and Thermal Management in the Intel CoreTM Duo Processor.

[16] Samuel Rodriguez and Bruce Jacob. Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm). In *Proceedings of International Symposium on Low Power Electronic Design*, Tegernsee, Germany, 2006.

[17] W.N. Scherer III and M.L. Scott. Advanced contention management for dynamic software transactional memory. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 240–248. ACM New York, NY, USA, 2005.

[18] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman Jouppi. Cacti 5.3, 2008.