Short Paper

Many-Core Token-Based Adaptive Power Gating

Andrew B. Kahng, Seokhyeong Kang, Tajana Simunic Rosing, and Richard Strong

***Abstract*—Among power dissipation components, leakage power has become more dominant with each successive technology node. Leakage energy waste can be reduced by power gating. In this paper, we extend token-based adaptive power gating (TAP), a technique to power gate an actively executing core during memory accesses, to many-core Chip Multi-Processors (CMPs). TAP works by tracking every system memory request and its estimated time of arrival so that a core may power gate itself without performance or energy loss. Previous work on TAP [11] shows several benefits compared to earlier state-of-the-art techniques [10], including zero performance hit and 2.58 times average energy savings for out-of-order cores. We show that TAP can adapt to increasing memory contention by increasing power-gated time by 3.69 times compared to a low memory-pressure case. We also scale TAP to many-core architectures with a distributed wake-up controller that is capable of supporting staggered wake-ups and able to power gate each core for 99.07% of the time, achieved by a non-scalable centralized scheme.**

***Index Terms*—Adaptive power gating, energy savings, low power design, many-core architecture.**

I. Introduction

During every cycle that a core is on, even when stalled, leakage power is consumed via gate leakage, gate-induced drain leakage, junction leakage, and subthreshold leakage. A core may stall quite often if it is intensely accessing the memory subsystem, as every time a thread makes a memory request that misses in the L1 cache, the core is subjected to a variable access latency. This variable latency often translates into a core stall during which no forward thread progress occurs and energy is wasted. For a 32-nm out-of-order EV6 core, stall energy can be up to 39.1% of total energy consumption for the Spec2006 benchmarks [17].

Power gating is a technique that drastically reduces leakage power by cutting off the current path from supply to ground through introduction of a transistor switch between them. At one end of the spectrum, functional unit power gating reduces power consumption of unused core functional units [25] with wake-up latencies of several nanoseconds. At the other end, entire cores may be power gated and woken up, with latencies of several tens of microseconds to account for saving and restoring all core state from memory [22]. An intermediate mechanism, memory access power gating [10], provides the ability to power gate an entire core, wake up a power-gated core in about 10 ns, and maintain the core's architectural and cache state.

In this paper, we extend token-based adaptive power gating (TAP) [11] to many-core architectures. TAP deterministically applies power gating during core stalls that are caused by the variable latency of requests to the memory subsystem. TAP achieves this by providing the capability to track every ongoing memory request and the expected response time for each memory access that misses in the L1 cache. An expected lower bound on latency is sent to each core's programmable power gating switch (PPGS) by modifying the cache controllers to send a token on any miss where the token includes an estimate of the access latency of a next-level memory hit. The result is that TAP can support power gating with no performance loss.

Previous work on TAP [11] assumes a centralized wake-up controller (WUC), which is a critical limiter to the scalability of the system. During each power-gating action, the core must communicate with the WUC to ensure that it can safely use its wake-up mode without violating voltage noise constraints. We replace the WUC with a distributed wake-up scheme that assigns cores to predetermined, recurring wake-up slots that enforce safe wake-up modes across the CMP. Such a scheme is shown to operate nearly as efficiently as an ideal WUC. Our key contributions are the following.

1) We introduce a distributed wake-up scheme to control core wake-up modes in many-core designs that sacrifices only 0.93% of power-gated time.
2) We demonstrate that TAP can adapt to an increase in memory contention by increasing power-gated time by 3.69 times as the number of threads increases from 1 to 32.
3) We design and implement a staggered wake-up scheme capable of reducing wake-up latency by up to 58.2%.

II. Related Work

Power gating has been studied at both circuit and architectural levels. Circuit-level papers typically analyze different circuit techniques aimed at reducing wake-up latency, efficiently retaining logic states, minimizing ground bounce, and achieving resilience to process variation. Microarchitectural works typically examine questions related to use of different power-gating modes, what to power gate, predicting when to power gate, and control algorithms to avoid energy penalties from poor power-gating decisions. The following briefly reviews representative works in these two areas.

In the realm of circuit innovation, the pioneering work in [8] has been followed by many works on fundamental circuit design issues related to power gating, including switch-cell sizing, data-retention methods, physical-implementation methodologies, and mode-transition noise analysis and reduction. The recent survey in [23] gives an excellent summary of the history and highlights of power-gating techniques.

Agarwal *et al.* [4] and Singh *et al.* [24] examine multiple sleep modes that feature different wake-up overheads and leakage power savings. To minimize ground bounce during mode transition, Kim *et al.* [12] control turn-on voltage (V_{GS}), which makes sleep transistors turn on in a non-uniform

Manuscript received June 25, 2012; revised October 2, 2012 and December 27, 2012; accepted January 15, 2013. Date of current version July 15, 2013. This work was supported in part by MARCO FCRP (GSRC and MuSyC centers), Qualcomm, Oracle, Huawei, and NSF under Grant SHF-0916127, Grant SHF-1218666, Grant SHF-1116667, and Grant CCF-1162085. This paper was recommended by Associate Editor Y. Shin. The authors would like to thank K. Jeong for his early work on this project [10].

The authors are with the University of California, San Diego, CA 92093 USA (e-mail: abk@ucsd.edu; shkang@vlsicad.ucsd.edu; tajana@ucsd.edu; rstrong@eng.ucsd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2257923

stepwise manner. Chowdhury *et al.* [6] propose a tri-mode power-gating technique using PMOS switches in parallel with NMOS footer switches. Finally, Zhang *et al.* [26] propose a multimode power-gating technique using three NMOS switches with different sizes and threshold voltages. Using various combinations of the three switches, they can provide multiple power-gating modes with different leakage savings.

In the microarchitectural arena, Hu *et al.* [9] propose use of power gating to reduce functional unit leakage power when applications underutilize their functional units. They [9] also develop equations to estimate the break-even points for power gating an out-of-order superscalar processor. Lungu *et al.* [18] show that in many cases, the predictor of [9] can lead to increased energy consumption. Madan *et al.* [19] extend the idea of [18] to the core level and propose a guard mechanism that reduces harmful use of power gating.

Power-gating technology is also readily visible in leading commercial products. The recent Nehalem architecture employs power gating at the core level to reduce leakage power on idle cores, but 100 ms is required to wake up a core [13], [16]. AMD [22] has improved this power-gating technique by optimizing the wake-up sequence to skip built-in self-tests (BIST) and restoration of cache state; this results in wake-up times as short as 75 μs. In today's systems, the operating system (OS) typically power gates the cores in the idle loop, missing out on power gating long memory accesses.

Li *et al.* [14] consider a scheme that directs core DVFS behavior based on signals from the L2-cache and estimates of instruction-level parallelism to reduce energy consumption on memory bound applications, but does not consider power gating. Our early work on memory access power gating [10], provides a mechanism to power gate an entire core, wake up a power-gated core in about 10 ns, and maintain the core's architectural and cache state. It uses a combination of a PPGS, state-retention cells, and source biasing to enable the core to efficiently enter and exit a power-gated state. The mechanism is shown to be capable of reducing leakage power during memory accesses. A similar work on memory access aware power gating for MPSoCs [15], examines the potential to power gate an in-order core while monitoring a single memory bus and estimating memory latencies. TAP [11] extends memory access power gating to out-of-order execution and considers using staggered wake-up for cores, but avoids questions about the scalability of TAP to many-core architectures. By contrast, our present work addresses the scalability of TAP to many-core designs.

III. System Design

A memory access power-gating controller must provide three functions. First, the controller should be able to predict the expected duration of core stalls. Second, the controller must assign a core's PPGS [10] a wake-up mode that does not violate supply voltage noise constraints of the system when waking up a core. Last, the controller must retain essential core architectural and performance-related state. Together, these functions allow for energy savings and minimal performance hit without violating voltage noise constraints. The rest of this section describes how our system provides these three functions.

A. Token-Based Latency Monitoring

TAP informs each PPGS about expected memory latency by modifying the cache controllers to send tokens on cache misses that include an estimate of the lower-bound access latency of a next-level memory hit derived from Table I and a time stamp of creation.[1] The controllers send the tokens to the PPGS of the core that requested the memory access. Once the PPGS receives the token, it looks at the lower-bound latency to satisfy the request and power gates the core if the core is both stalled and idle long enough to save energy. Should the core receive more than one token for simultaneous memory requests, it tracks each expected response separately and schedules the resumption of core execution to satisfy the earliest response. If a token is delayed in the memory subsystem by a controller or queue, the PPGS can compare its arrival time with its generation time stamp and previous tokens to determine whether the token should be ignored.

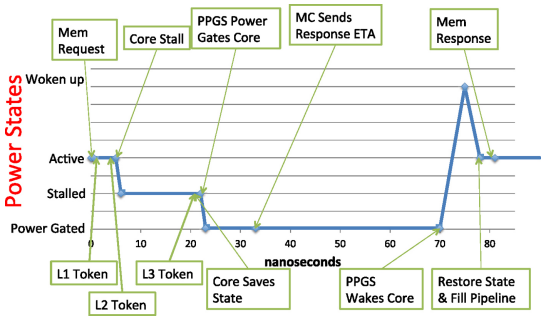

Fig. 1. Diagram depicting the core going through various power states (power gated, stalled, active, woken up) as the PPGS power gates the core on a memory access.

Whenever a memory request misses all the way to the memory controller, the response latency experiences a significant amount of variability. This variability is caused by the complexity of DRAM memory [27], which includes bank queues, availability of the data in the row-buffer, writing wrong address row-buffers, accessing the column in the row-buffer, and channel contention between banks. TAP adapts to memory variability by adding a special token. As soon as the last-level cache experiences a miss, a token is sent to the requesting core's PPGS with an estimated completion time of "unknown". This is a directive to the PPGS to start power gating its core immediately and to expect one additional token with the ETA of the memory response. Once the memory controller submits the memory access to one of the banks and determines whether the access is a row-buffer hit or miss, it sends the second *ETA* token to the core's PPGS with the ETA of the response assuming that there is no memory channel contention. The PPGS receives the second token before the response and schedules core wake-up for the appropriate time.

Fig. 1 shows a timing-accurate diagram of a PPGS power gating the core in response to messages from the TAP technique. At time 0 ns, a memory request occurs that will miss in the cache hierarchy and cause a memory access. The PPGS then receives tokens for the L1, L2, and L3 misses. Just after receiving the L2 token, the core stalls due to a dependency. After the L3 token is received, the PPGS decides to power gate the core and saves all core state. The core is then power gated and the memory controller (MC) sends a updated ETA for the memory response. At 70 ns, the PPGS begins waking up the core. At 78 ns, the core state is restored and the pipeline is restarted. The memory response comes back at 81 ns and the core resumes execution as if nothing happened.

[1]A cache controller sitting on the core side of a shared NUCA cache would require a per-bank lower-bound access latency.

0	1	2	3	4	...
3	4	5	6	7	...
6	7	8	0	1	...
0	1	2	3	4	...
...	...	...	...	...	...

(a)

0	1	2	3	4	...
2	3	4	0	1	...
4	0	1	2	3	...
1	2	3	4	0	...
...	...	...	...	...	...

(b)

0	1	2	0	1	...
2	1	0	2	1	...
1	2	0	1	2	...
0	2	1	0	2	...
...	...	...	...	...	...

(c)

Fig. 2. Wake-up slot assignments with different number of slots(η). (a) η = 9. (b) η = 5. (c) η = 3.

B. Distributed Staggered Wake Up

Previous work [10] designed its WUC for the worst case when all cores wake up simultaneously. However, wake-up latency is significantly reduced when we stagger the wake-up sequence so that two cores wake up at slightly different times (e.g., offset by 1 ns). In such a case, stagger reduces the worst-case peak current and voltage noise that a core may experience. We now consider how to integrate stagger into a many-core design; analysis of the benefit of stagger is given in Section IV-B.

To use staggered wake up, a controller must give wake-up modes and times to each core. For a large multicore system (e.g., 64 cores), a given core's PPGS may not tolerate the latency to communicate with a centralized WUC due to propagation and queuing delay across the chip. However, we observe that non-adjacent cores do not significantly affect core wake-up latency, and with proper guard-band, only adjacent cores (eight cores at most) need to be considered.

This observation motivates a distributed design that assigns each core to a recurring wake-up slot. In this scheme, each core is given a recurring slot at which it can start waking up. To avoid any performance hit, a core should select a slot before the deadline to start waking up. The average wake-up delay for a core is defined by two degrees of freedom: the number of unique wake-up slots, η, and the stagger between two adjacent wake-up slots, ψ. The worst-case reduction in power-gated time occurs when a core predicts that it would need to wake up ϵ seconds before its assigned slot such that $\epsilon < \psi$, causing the core to wake up $\eta * \psi - \epsilon$ seconds earlier. Given a core that wakes up at any time, uniformly at random, the average expected reduction in power-gated time is $\frac{\eta * \psi}{2}$.

Values of η and ψ should be chosen to maximize a core's power-gated time and minimize the safe wake-up latency of the core. Given η, wake-up slots should be assigned to cores to minimize the number of adjacent woken-up cores. Fig. 2 shows three ways of assigning wake-up modes to cores such that the number of adjacent woken-up cores is minimized with a preference given to cores waking up simultaneously in the diagonal position. An increase in η and ψ reduces the maximum number of simultaneous core wake-ups and the minimum safe wake-up latency. At the same time, increasing these two parameters increases average expected reduction in power-gated time. According to our simulations, system energy savings are maximized when η and ψ equal 5 (no simultaneous wake-ups) and 0.9 ns, respectively. This setting results in a 10.3 ns minimal wake-up latency and 2.25 ns $\pm$ 1.30 ns average reduction in power-gated time per core power-gating opportunity, compared to 9.6 ns with 0.9 ns stagger for an ideal centralized WUC, when simulated on GEM5 [5] with the Spec2006 benchmarks.

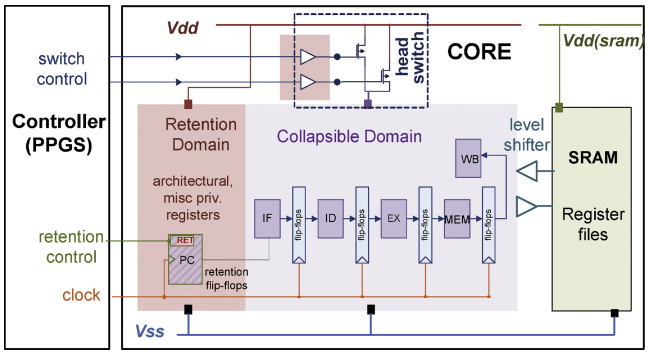

Fig. 3. Interface for power gating and data retention.

C. Core State Retention and Restoration

To avoid losing core state that is required for correct and efficient execution, essential sequential and SRAM cells must be retained. We use the technique from [10] that replaces a subset of sequential cells with live-slave retention flip-flops [7] that can be triggered to retain their logical values before a power-gating action at a cost of 20% increase in area and power versus a normal flip-flop. Only those sequential cells comprising the architectural registers necessary to refill the pipeline are selected, which results in a 3.4% area overhead for the processor. SRAM cells' state is retained through source biasing [21] in which the supply voltage is reduced to 50% of nominal so that SRAM leakage is reduced, but logical state is maintained. This technique enables saving contents of L1 caches, TLBs, branch predictor state, physical registers, etc. To provide supply power during power gating, a separate non-collapsible voltage domain provides power to the retention flip-flops and SRAM cells. Thus, as the power is gated from combinational logic and non-essential sequential cells, the separate voltage rail provides power to maintain core state. The overhead from multipower domains and separate voltage rails already exists for power gating cores today. Additional cycles are required for the power gating and wake-up sequence, and to disable/enable the clock, trigger data retention, refill the pipeline, and de-assert/assert the clamps. We model the power-down and wake-up sequence as in [10]. Fig. 3 shows an in-order core implementation for the power gating and restoration with retention flip-flops.

IV. Results

Table I summarizes all system parameters in our experiments. Unless otherwise stated, our results assume a 32-nm HP circuit technology, a 10.2 ns wake-up mode, and an out-of-order CMP. The following overheads were considered when calculating the reported results: core wake-up energy, core wake-up delay, core pipeline-refill latency, retention overhead of live-slave retention cells, SRAM leakage during source biasing mode of operation, and voltage noise safety.

We simulate the system with the GEM5 simulator [5]. GEM5 features cycle-level models of an out-of-order core, the cache hierarchy, and the interconnect. We integrate GEM5 with DRAMSim2 [1] to provide cycle-level modeling of the memory subsystem including the memory controller, DRAM modules, and shared channels used for communication.

We use McPAT [17] to model dynamic power, leakage power, peak power, and area. We update McPAT's technology.cc file to accurately reflect the ITRS 2010 update report [2].

TABLE I
SYSTEM CONFIGURATION VALUES

parameter	value	notes
EV6 core model	DEC-Alpha EV6	
EV6 core clock	3.3 GHz–1.9 GHz	
EV6 execution	6-way out-of-order	
EV6 functional units	6ALU,2IMULT 2FPALU	
ICache/DCache	32KB-8way 1 cycle	
L2 Cache	256KB-8way 4 ns	Private per core
L3 Cache	8MB-16way 13 ns	Shared
Memory	DDR3 2GB 50 ns	
Core-to-L1 token latency	0.5 ns	controller delays
Core-to-L2 token latency	4.5 ns	controller delays
Core-to-L3 token latency	17.5 ns	controller delays
Core-to-centralized-WUC latency	5 ns	controller delays
PPGS wake-up modes	4.5 ns–16.9 ns	SPICE
EV6 pipeline refill latency	2.12 ns	7 pipeline stages
EV6 core wake-up energy (EWE)	15,358pJ	Charge cells
EV6 leakage power (ELP)	0.916 Watts	McPAT [17]
EV6 PG leakage reduction (ELPR)	97.65%	[7]
EV6 PG break even point	$EWE/(ELPR * ELP)$	
EV6 DFLT core wake-up latency	10.2 ns	SPICE
EV6 FUPG wake-up energy	9641pJ	McPAT, ITRS [2]
EV6 FUPG wake-up latency	6.4 ns	SPICE

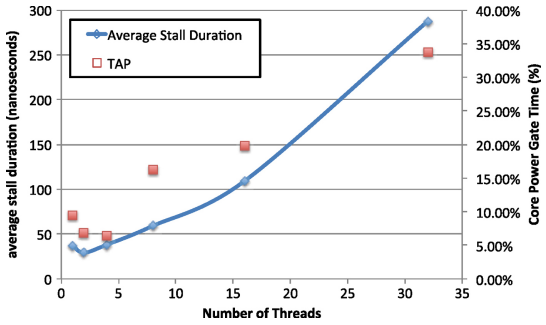

Fig. 4. TAP adapting to increasing memory contention. The left y-axis shows the duration of the stall in nanoseconds, while the right y-axis shows the percentage of time that TAP can power gate as a function to the number of STREAM threads.

A. Adapting to Memory Contention.

We show how TAP adapts to a system facing increased memory contention for the multithreaded memory benchmark STREAM running on a CMP with up to 32 cores. STREAM is a memory-intense benchmark used to measure sustained memory bandwidth and computation rates for simple vector kernels [3]. We modify STREAM to act as a parallel memory benchmark such that more threads cause more simultaneous requests to the memory subsystem. Increasing STREAM's thread count causes more queuing of memory requests, longer delay per request, and more frequent stalling of each core. A good power-gating technique should be able to power gate the core more often to reduce power consumption.

Fig. 4 depicts both average duration of core stalls and the percentage of total simulation time TAP power gates the core. First, we note that as the number of threads increases, the average duration of a core stall increases. From one to 32 threads, average core-stall duration increases[2] by 7.82× from 36.77 ns to 287.63 ns. The increase in the average core-stall duration is caused by more threads making parallel requests to the memory subsystem at once, fewer row-buffer hits, and memory channel contention. As cores experience increased memory latency, TAP power gates the core longer. From one to 32 threads, TAP power gates cores 3.69 times longer from 9.08% to 33.50% of simulation time. However, TAP power gates its core less for four threads than for two even though average core-stall time increases. This is because TAP uses a conservative lower-bound estimate of memory-response time and does not account for all memory scheduling possibilities. The benefit of this conservative policy is the absence of any application performance hit.

[2]From one to two threads, core-stall duration decreases. This happens because more threads increase the amount of available cache (more cores) while increasing the number of simultaneous memory requests.

B. Distributed, Staggered Wake Up

For a 16-core system with multiple cores waking up simultaneously, voltage noise on the power distribution network can cause unsafe voltage drops on neighboring active cores. By introducing a sub-nanosecond stagger between any two adjacent cores waking up, worst-case inrush current and resulting voltage noise are reduced. The result is a faster wake-up mode and increased energy savings on the chip. Hence, cores should wake up staggered by at least a fraction of a nanosecond.

Fig. 5 shows SPICE simulation of the effect of stagger on core wake-up latency for no-stagger, 0.3 ns-stagger, 0.6 ns-stagger, and 0.9 ns-stagger for CMPs composed of 16 EV6 cores as 0 to 14 cores are idle. Staggered wake up can reduce the variance between the min and max wake-up latency when most cores are actively executing or waking-up. For the 16-core CMP with no cores idle and no stagger, the max and min wake-up latency is 18.4 ns and 9.7 ns, whereas, a staggered wake up of 0.9 ns decreases the max and min values to 10.7 ns and 8.6 ns, respectively. As more cores go idle, staggered wake up has less impact on reducing the variance of wake-up latency because cores are less likely to interfere with each other, and the core's location becomes the dominant factor in wake-up latency. For example, when 14 cores are idle in a 16-core CMP, the min and max wake-up latencies are both 4.5 ns and 9.1 ns, respectively, in both the no-stagger and 0.9 ns-stagger cases. Lastly, stagger reduces the maximum wake-up latency as more cores are active. An example of this is for the 16-core case when no core is idle; maximum wake-up latency is 18.4 ns without stagger and 10.7 ns with 0.9 ns stagger, a reduction of 58.2%. Thus, stagger relaxes the guardband on wake-up latency.

Fig. 6 shows the energy impact of staggered wake up for a CMP with 16 EV6 cores. The largest improvement in energy savings is 3.14% for *mcf* as energy savings increase from 18.92% to 22.06% for staggered wake ups of 0.0 ns and 0.9 ns respectively. On average, energy savings go from 3.52% to 4.34% as stagger increases from 0 ns to 0.9 ns. Staggered wake up does not cause any decrease in energy savings. Although cores may have to wake up at slightly different times, the savings in latency with which they wake up is much greater than the stagger offset of 0.9 ns.

To have cores safely and reliably use staggered wake up, a controller or scheme is required to control the wake-up behavior of all cores. A centralized WUC would be able to use more aggressive wake-up modes and power gate the core for longer, but such a design does not scale to many cores. By contrast, the distributed wake-up control of Section III can scale to many-core designs, but sacrifices some power-gating time waiting for an assigned wake-up slot. Fig. 7 shows

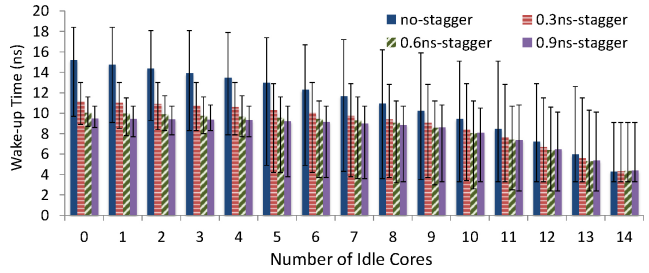

Fig. 5. Improvement in core wake-up latency with increased stagger for a 16 EV6 core CMP as 0 to 14 cores are idle. The average latency is shown with bars denoting the min and max safe wake-up modes.

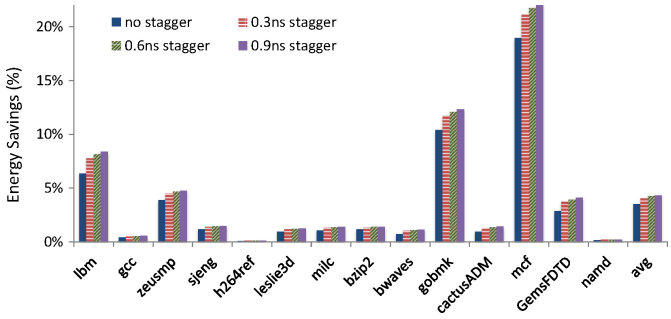

Fig. 6. Improvement in energy savings with staggered wake ups in an 16 EV6 core CMP. Benchmarks with less than 0.2% energy savings filtered.

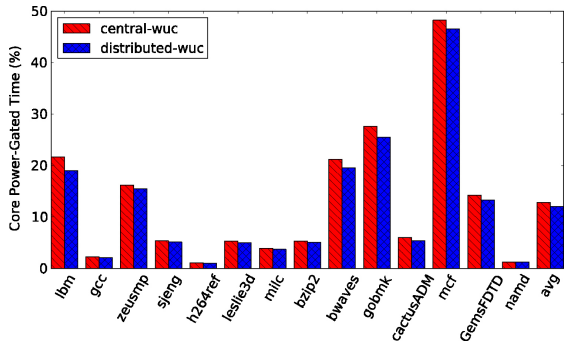

Fig. 7. Comparison of power-gated time using staggered wake up with a centralized WUC versus a distributed WUC.

the difference of application power-gated time for a subset of the SPEC2006 benchmarks[3] in a 16-core architecture. Our simulations show that the distributed wake-up control has a maximum decrease in power-gated time of 2.68% (0.93% on average) for the benchmark *lbm*. This indicates that the difference in energy savings between a distributed scheme and a centralized WUC would be negligible, but that a distributed scheme scales to many-core architectures.

V. Conclusion

With each new generation of microprocessors, leakage power becomes an increasingly dominant issue. In this paper, we extend TAP [11] to many-core designs through a distributed, staggered wake-up scheme. Our system can adapt to different levels of memory contention by increasing power-gated time by 3.69 times of total execution time as the number of threads increases from one to 32. We demonstrated that a wake-up stagger of 0.9 ns reduced core wake-up latency by up to 58.2% (7.7 ns). Finally, a distributed wake-up scheme that uses staggered wake-up can power gate cores for 99.07% of the time achieved by an ideal centralized scheme, while still being able to scale to many-core designs.

[3]We use the Simpoint methodology [20] with 100M-instruction representative regions for each application.

References

[1] *DRAMSim2*. (2011) [Online]. Available: http://www.ece.umd.edu/dramsim/

[2] *International Technology Roadmap for Semiconductors*. (2010) [Online]. Available: http://www.itrs.net/

[3] *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. (2011) [Online]. Available: http://www.cs.virginia.edu/stream/

[4] K. Agarwal, H. Deogun, D. Sylvester, and K. Nowka, "Power gating with multiple sleep modes," in *Proc. Int. Symp. Quality Electron. Des.*, 2006, pp. 633–637.

[5] 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.*, vol. 26, no. 4, pp. 52–60, Jul.–Aug.2006.

[6] M. H. Chowdhury, J. Gjanci, and P. Khaled, "Innovative power gating for leakage reduction," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 1568–1571.

[7] D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual*, Berlin, Germany: Springer, 2007.

[8] M. Horiguchi, T. Sakata, and K. Itoh, "Switched-source-impedance CMOS circuit for low standby subthreshold current giga-scale LSI's," *IEEE J. Solid-State Circuits*, vol. 28, no. 11, pp. 1131–1135, May 1993.

[9] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proc. Int. Symp. Low Power Electron. Des.*, 2004, pp. 32–37.

[10] K. Jeong, A. B. Kahng, S. Kang, T. S. Rosing, and R. Strong, "MAPG: Memory access power gating," in *Proc. Des., Autom. Test Eur.*, 2012, pp. 1054–1059.

[11] A. B. Kahng, S. Kang, T. S. Rosing, and R. Strong, "TAP: Token-based adaptive power gating," in *Proc. Int. Symp. Low Power Electron. Des.*, 2012, pp. 203–208.

[12] S. Kim, S. V. Kosonocky, and D. R. Knebel, "Understanding and minimizing ground bounce during mode transition of power gating structures," in *Proc. Int. Symp. Low Power Electron. Des.*, 2003, pp. 22–25.

[13] R. Kumar and G. Hinton, "A family of 45nm IA processors," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2009, pp. 58–59.

[14] H. Li, C. Cher, T. N. Vijaykumar, and K. Roy, "VSV: L2-miss-driven variable supply-voltage scaling for low power," in *Proc. IEEE Int. Symp. Microarch.*, Dec. 2003, pp. 19–28.

[15] Y. Lin, C. Yang, J. Huang, and N. Chang, "Memory access aware power gating for MPSoCs," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2012, pp. 121–126.

[16] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power management of datacenter workloads using per-core power gating," *IEEE Comput. Arch. Lett.*, vol. 8, no. 2, pp. 48–51, Feb. 2009.

[17] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. IEEE/ACM Int. Symp. Microarch.*, Dec. 2009, pp. 469–480.

[18] A. Lungu, P. Bose, A. Buyuktosunoglu, and D. J. Sorin, "Dynamic power gating with quality guarantees," in *Proc. Int. Symp. Low Power Electron. Des.*, 2009, pp. 377–382.

[19] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram, "A guarded power gating for multi-core processors," in *Proc. Int. Symp. High-Performance Comput. Arch.*, 2011, pp. 291–300.

[20] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," in *Proc. Int. Conf. Meas. Modeling Comput. Syst.*, 2003, pp. 318–319.

[21] H. Qin, Y. Cao, D. Markovic, A. Vladimirescue, and J. Rabaey, "SRAM leakage suppression by minimizing standby supply voltage," in *Proc. Int. Symp. Quality Electron. Des.*, 2004, pp. 55–60.

[22] A. Rogers, D. Kaplan, E. Quinnell, and B. Kwan, "The Core-C6 (CC6) sleep state of the AMD Bobcat x86 microprocessor," in *Proc. Int. Symp. Low Power Electron. Des.*, 2012, pp. 367–372.

[23] Y. Shin, J. Seomun, K.-M. Choi, and T. Sakurai, "Power gating: Circuits, design methodologies, and best practice for standard-cell VLSI designs," *ACM Trans. Design Autom. Electron. Syst.*, vol. 15, no. 4, pp. 1–37, 2010.

[24] H. Singh, K. Agarwal, D. Sylvester, and K. Nowka, "Enhanced leakage reduction techniques using intermediate strength power gating," *IEEE Trans. Very Large Scale Integr.*, vol. 15, no. 11, pp. 1215–1224, Nov. 2007.

[25] O. Wechsler, "Setting new standards for energy-efficient performance," *Technol. Intel Mag.*, 2006 [Online]. Available: http://www.intel.com/pressroom/kits/core2duo/pdf/ICM_whitepaper.pdf

[26] Z. Zhang, X. Kavousianos, K. Chakrabarty, and Y. Tsiatouhas, "A robust and reconfigurable multi-mode power gating architecture," in *Proc. Int. Conf. VLSI Des.*, 2011, pp. 280–285.

[27] H. Zheng and Z. Zhu, "Power and performance trade-offs in contemporary DRAM system designs for multicore processors," *IEEE Trans. Comput.*, vol. 59, no. 8, pp. 1033–1046, Aug. 2010.