Using STT-RAM to Enable Energy-Efficient Near-Threshold Chip Multiprocessors*

Xiang Pan
Computer Science and Engineering
The Ohio State University
panxi@cse.ohio-state.edu

Radu Teodorescu
Computer Science and Engineering
The Ohio State University
teodores@cse.ohio-state.edu

ABSTRACT

Near-threshold computing is gaining traction as an energy-efficient solution for power-constrained systems. This paper proposes a novel near-threshold chip multiprocessor design that uses non-volatile spin-transfer torque random access memory (STT-RAM) technology to implement all on-chip caches. This technology has several advantages over SRAM that are particularly useful in near-threshold designs. Primarily, STT-RAM has very low leakage, saving a substantial fraction of the power consumed by near-threshold chips. In addition, the STT-RAM components run at a higher supply voltage to speed up write operations. This has the effect of making cache reads very fast to the point where L1 caches can be shared by several cores, improving performance. Overall, the proposed design saves 11–33% energy compared to an SRAM-based near-threshold system.

1. INTRODUCTION

Power consumption is now a first-class constraint in microprocessor design. Agressive supply voltage (V_{dd}) reduction to close to (or even below) the transistor threshold voltage (V_{th}) has been investigated as an energy-efficient solution for driving future processors with hundreds of cores [2]. Near-threshold operation lowers power consumption by about 100× at a cost of 10× lower operating frequency, potentially improving energy efficiency by an order of magnitude. However, near-threshold operation faces several challenges including decreased reliability, increased sensitivity to process variation, and high relative leakage power.

While dynamic power scales quadratically with supply voltage and linearly with frequency, leakage power only scales linearly with V_{dd}. As a result, leakage power dominates power consumption at low voltages. Most of that leakage is dissipated by on-chip caches, which can account for 20–40% of the total power depending on size.

We present a novel near-threshold chip multiprocessor design that uses non-volatile spin-transfer torque memory (STT-RAM) technology to implement all on-chip caches. This technology has several advantages over SRAM that are particularly useful in near-threshold designs such as low leakage, high density, and high endurance. A drawback of STT-RAM is its higher write latency and energy compared to SRAM. The proposed design runs the STT-RAM components of the chip at a higher supply voltage to speed up write operations. This has the effect of making cache reads very fast to the point where L1 caches can be shared by several cores within a cluster. This eliminates the need for within-cluster cache coherence improving latency and energy. We leverage the shared cache design in our CMP to help mitigate the effects of variation in core frequency.

Evaluation using parallel benchmarks from SPLASH2 and PARSEC shows average energy savings of 11–33% (depending on cache size) and performance improvement of 17% relative to the baseline near-threshold system.

2. NT CMP WITH STT-RAM CACHES

We design an NT CMP processor that replaces SRAM caches with STT-RAM versions. The CMP is organized in clusters that share a last-level cache (L3). The CMP has two externally regulated voltage domains. One domain, which contains most of the cores' logic is set at low NT V_{dd}. The second domain, which includes the entire STT-RAM cache hierarchy and a few logic units, runs at high (nominal) V_{dd}. The main reason for running the STT-RAM domain at nominal V_{dd} is to improve write speed. Since cores will generally run at lower frequencies at NTV, the write latency is relatively smaller (about 3 cycles for a core running at 500MHz).

Shared Cache Hierarchy: The nominal voltage STT-RAM cache has very fast read speeds, on the order of 0.4ns for a 256K L1 cache. This is much faster than needed for single-cycle cache reads at the lower NT frequencies (250MHz-1.25GHz). We exploit this property by sharing a single L1 instruction and an L1 data cache, as well as an L2 cache between all the cores in each cluster. This is achieved by running the shared L1 caches at high frequency (2.5GHz in our experiments - to match the 0.4ns access time) and time-multiplexing requests from the cores in each cluster. This greatly reduces the latency cost of sharing data between threads executing on different cores. It also reduces implementation complexity and energy cost.

*This work was supported in part by the Defense Advanced Research Projects Agency under the PERFECT (DARPA-BAA-12-24) program and the National Science Foundation under grants CCF-1117799 and CCF-1253933.

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08.
http://dx.doi.org/10.1145/2628071.2628132 .

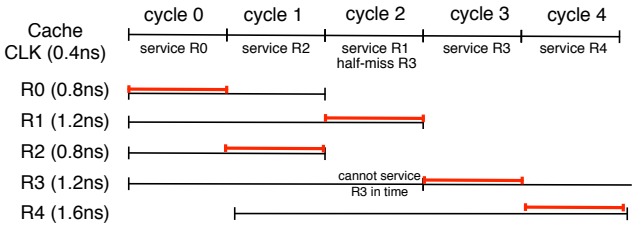

Figure 1: Example timeline of access requests from cores running at different frequencies to a fast shared cache.

Since our design allows cores in the same cluster to run at different frequencies, arbitrating access to the shared cache has to account for the different latency tolerances of each core. To simplify the implementation, the shared cache and all cores run at a frequencies corresponding to integer multipliers of a base clock period. As a result, all cache accesses align at cycle boundaries with the high frequency cache.

Figure 1 shows an example timeline illustrating how multiple access requests (R0-R4) from cores running with different clock periods (0.8ns-1.6ns) are served by the cache. In cycle 0 the cache receives requests R0-R3 from four different cores. For each request the figure includes a line segment that represents the cycle time of the core issuing that request. For instance, R1 comes from a core running at 833MHz (1.2ns clock cycle time). To ensure that, in the event of a hit, the cache responds within a single core clock cycle, the cache must send the data by the end of cache cycle 2.

In cycle 0 we have four new requests out of which the cache can only service one. In this example request R0 is the most urgent, so it will be serviced first, followed by request R2 in cycle 1. In cycle 2 both requests R1 and R3 are expiring but only one can be serviced. The controller chooses to serve R1; for R3 a "half-miss" event will be sent to the processor to indicate that the request could not be fulfilled in a single cycle, but this is not necessarily an L1 miss. Request R3 will be rescheduled to the following cycle.

Mitigating Frequency Variation: The shared cache design significantly reduces thread migration overhead. We leverage this benefit to reduce the effects of core-to-core frequency variation on application performance. In this design we allow cores to run at their best frequency, resulting in heterogeneous performance. To deal with this heterogeneity, we periodically migrate applications between fast and slow cores. The goal is to ensure multiple threads of parallel applications make roughly equal progress even though they are running on cores with different frequencies.

3. EVALUATION

Methodology: We modeled a 64-core CMP with a range of cluster sizes from 4 to 32 cores. Most experiments were conducted with a cluster size of 16 cores, which we found to be optimal. We also experimented with three cache sizes: small, medium, and large. Each core is a dual-issue out-of-order architecture. We used SESC to perform all our simulations and NVSim [1] to get STT-RAM latency and energy. We ran SPLASH-2 and PARSEC.

Power Analysis: Figure 2 shows the reduction in power consumption from the proposed architecture without thread migration (SH-STT-NoMig). We show results for three cache configurations: small, medium, and large. The medium size cache is the most typical one, with about 1MB/core of total cache capacity. We compare to a baseline that uses SRAM caches running at NT voltages in a traditional private cache hierarchy (PR-SRAM-NT).

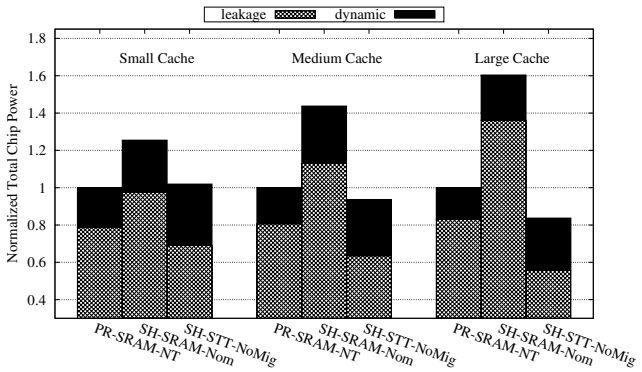

Figure 2: Power reduction of proposed design for three L2/L3 cache sizes: small, medium, and large.

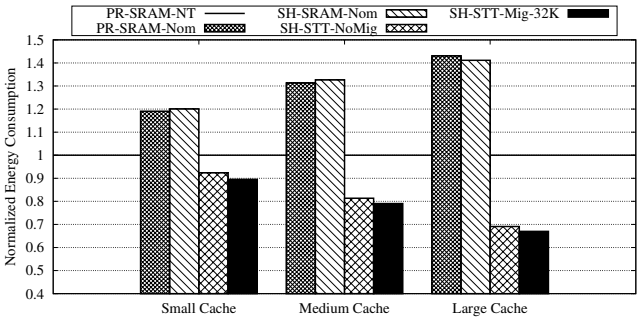

Figure 3: Energy consumption for small, medium, and large L2 and L3 cache configurations.

We can see that power is lower for SH-STT-NoMig compared to the baseline in all configurations except for the small cache one. The reduction in total power comes from lower leakage power at the cost of slightly increased dynamic power. For the medium and large configurations the power savings are significant, at 6.4% and 16.3% respectively.

For reference, we also compare to a SRAM design with shared cache running at nominal voltage (SH-SRAM-Nom). SH-SRAM-Nom uses between 20% and 100% more power than SH-STT-NoMig for the three cache sizes. This is due to the much higher leakage power consumed by SRAM.

Energy Analysis: Our proposed design lowers power consumption and reduces execution time. This results in important energy savings. Figure 3 shows that SH-STT-NoMig has 8% to 30% lower energy than the SRAM baseline depending on the cache size configuration. As expected we see much larger energy savings for the large cache configurations. We also show that the PR-SRAM-Nom configuration which uses SRAM caches at nominal V_{dd} uses 20-40% more energy than the NT SRAM baseline (PR-SRAM-NT).

4. REFERENCES

[1] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, July 2012.

[2] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, February 2010.