Accelerate Context Switch by Racetrack-SRAM Hybrid Cells

Weiqi Zhang
Peking University
zhangweiqi@pku.edu.cn

Chao Zhang
Peking University
zhang.chao@pku.edu.cn

Guangyu Sun
Peking University
gsun@pku.edu.cn

ABSTRACT

Context switch is an essential feature of modern operating systems. The purpose of context switch is to provide concurrency processing of multiple programs. However, the backup and reload procedures due to context switch are time consuming. In this work, we propose a Racetrack-memory SRAM hybrid (RSH) cell design, which can be used to replace current SRAM cells in caches, to reduce the overhead of context switch. Using RSH reduces the overhead of context switch by 65.2% on average with 34% area overhead, compared with traditional SRAM designs.

Keywords

Racetrack Memory, Context Switch, Cache

1. INTRODUCTION

A single processor core can only load and run one process at a time. Other processes have to wait until the running process gives up controlling the processor. To fully utilize computing resources, run multiple programs roughly at the same time and avoid process hungriness, context switch (CS) was proposed and became an essential feature of modern Operating Systems (OS). With CS, OS gives each process a certain time slice to run. The OS will switch the process off whenever its time slice is over. However a huge time overhead is induced by doing so. When the CS happens, it will backup CPU registers and flush virtual address indexed caches and Translation Lookahead Buffers (TLBs). Once the context is switched back to previous states, the values in registers will be reloaded. But the data in caches and TLBs cannot be reloaded immediately due to its large quantity. As shown in Figure 1, CS incurs 25.9% time overhead of a program on average. The Run-alone time is the on-CPU time of a benchmark running alone and the With-CS time is the on-CPU time of the same benchmark running with a reference process competing the core. We can see the competition cause a lot of overhead due to cache and TLB miss.

The cost of CS has been fully studied by previous work [4]. The cost of context switch contains two parts: direct cost and indirect cost. The direct cost is caused by LOAD/STORE instructions of backing up and restoring registers, flushing caches, and flushing TLBs. On the other hand, the indirect cost makes up the majority of the CS overhead. Due to the flush of caches and TLB, quite a lot misses are unavoidable when a program is switched back. An instruction TLB miss leads to a long pipeline stall. The system need to look up the page table to find out the physical address of this instruction before its execution. Extra cache misses will be provoked because hot data cached is flushed in previous CS. These misses will eventually degrade the performance of programs. Thus, a method to reduce the overhead of context switch is required.

In this paper, we propose a new memory cell design called racetrack-SRAM-hybrid (RSH) cell. RSH cell is an racetrack memory (RM) augmented SRAM cell which can store multiple bits and be accessed by one bit at a time. Besides normal read and write operations, RSH cells can write data back to the augmented RM and reload them later. To prevent the data from being flushed to lower level storage, RSH cells are used to replace conventional SRAM cells in caches and TLBs. This method can significantly reduce the overhead of CS.

2. RACETRACK SRAM HYBRID CELL

To reduce the overhead of CS, we introduce a new cell structure, which can be used by both TLBs and caches. It is called Racetrack-SRAM Hybrid cell, which is a combination of an RM cell and a SRAM cell. Racetrack Memory(RM) is a type of emerging non-volatile memory based on spintronic technology. A lot of works focus on RM [7][3] because of its ultra-high storage density. Racetrack memory provides higher storage density by integrating multiple bits in a tape-like nanowire compared to STT-RAM [8]. As shown in the right part of Figure 2, the bits of RM are stored in separate domains (white blocks) separated by domain walls (dark bricks). The magnetization direction (arrows) of a domain can be considered as either bit 1 or bit 0. There are several access ports distributed along the nanowire. Each port is consisted of a transistor and a reference domain attached by a magnetic tunneling junction (MTJ). Only the domain aligned to the access port can be read or written. In order to access unaligned domains, the domains on the nanowire have to be shifted to the port. This operation is called shift operation.

An RSH cell has an SRAM cell and an RM cell, connected through the bit line of the SRAM cell. The data used in current context is stored in SRAM cells, and the backup data are stored in RM cells. Figure 2 shows the structure of RSH cells. The bit read from SRAM can be written into RM and bit read from RM can also be written into SRAM. Previous work [6] has demonstrated that STT-RAM cell can work with SRAM cell this way. The RM cell design shares the same access port structure with STT-RAM cell, thus a similar method can be used by the combination of RM and SRAM cells. To control the RSH cell, three transistors are added to a cell: two for data transportation between SRAM and RM, one for RM shift control. Even though the nanowire of a RSH cell is fabricated in a different layer from transistors, the extra transistors still occupy extra area. The area overhead can be up to 50%.

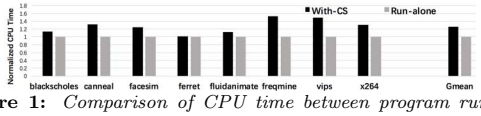

Figure 1: *Comparison of CPU time between program running alone and running with context switch.*

Since all the data in SRAM are required to be backed up or reloaded during the CS, different cells share the same behavior. When the backup operation is activated, all cells write data from SRAM to RM. And the data will be stored with the same offset within the RM cells. Thus

Nanoarch '16, July 18-20, 2016, Beijing, China
© 2016 ACM. ISBN 978-1-4503-4330-5/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2950067.2950070

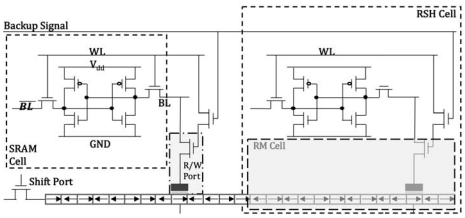

Figure 2: *An example of two Racetrack-SRAM hybrid (RSH) cellsl*

the shift distance of all RM cells are same. And this offers an opportunity for the combination of multiple RM cells with one nanowire. An example is shown by Figure 2, where two SRAM cells attached by two RM cells with a nanowire and a shift port. If we only keep the data for 4 processes, each RSH cell need 4 domains to backup the data. Thus a nanowire which can hold 64 bits can be shared by 16 SRAM cells, to reduce the area overhead. The area overhead can be reduced to 34.3%, if 16 SRAM cells share one RM nanowire.

3. RSH-BASED CS MECHANISM

In order to make full use of the caches and TLB built by RSH cell, we propose RSH architecture. The layout is demonstrated by Figure 3. The basic idea is adding a CS controller to memory management unit (MMU). The CS controller has three components: Crx, Cr3 and a mapping table. The Crx is used to temporarily store the Cr3 of incoming process. Note that the Cr3 is used by OS to save the page directory address of the current running process. Thus it can be regarded as a process's ID. The mapping table records the relationship between Cr3 value and RM bits positions, which means the mapping table's entry number is same as the bits number in an RM cell. Also, our mechanism keeps that RM bits aligned to the port is always allocated for the data used in current process.

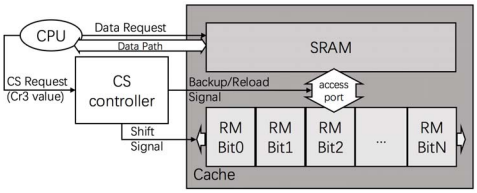

Figure 3: *The architecture of the RSH-based context switch mechanism*

Between two CSs. The RSH caches or TLBs are served as traditional SRAM-based caches or TLBs. When a CS comes, RSH-based backup and restore mechanism is triggered. Since the method used in TLBs is similar with that in caches, we take the caches' method as an example. The steps taken by the caches is listed as follows:

1. CPU writes the incoming process's page directory address into Crx and sends the incoming process's priority level to CS controller.
2. The CS controller senses that the value of Crx is different form Cr3. Then it looks up the mapping table to find out the RM bits position P1 mapped to Cr3 value and P2 mapped to Crx value. There might be three special situations:
   - The mapping table is full and incoming process has higher priority than current process. In this situation, the CS controller makes P2 = P1 and makes P1 invalid.
   - The mapping table is full and incoming process has lower or equal priority than current process. In this situation, the CS controller makes P2 invalid.
   - There is no matching entry in the mapping table of the incoming process but the mapping table is not yet full. In this situation, the CS controller mapping the first free RM bit to the incoming process. And fill the corresponding mapping table entry with the value in Crx.
3. The CS controller sends backup signal to cache.
4. If P1 equals to current RM position, data can be backed up into RM. If P1 is invalid, the data can only be flushed to lower level storage. Both situations, the SRAM cells are available for the incoming process.
5. If P2 is not invalid, the CS controller will shift the RM position to P2 and load the bits from RM into SRAM. Then CS controller send a success signal to CPU.
6. CS controller copy the page directory address from Crx into Cr3.

Note that if a process finishes, its data will be written back into lower level storage and its mapping table entry will be marked as free.

This technique reduces indirect overhead significantly, since it reduces the miss caused by flush and the congestion in lower level storage. However, the capability of this technique is limited by the RM domains allocated to each SRAM cell. If only 2 domains are allocated by 1 SRAM cell, only 2 process can be backed up efficiently by this technique.

4. EVALUATION

We implement the RSH architecture in MARSS x86 simulator [5]. The simulation configuration is as follow: CPU is 4 cores at 2GHz; L1 cache is 32K at 2 cycles access latency; L2 cache is 256K with 6 cycles access latency; L3 cache is 2M with 27 cycles access latency and the memory latency is 100 cycles.

We chose PARSEC-3 [1] as benchmark set. And the simulation result is shown by Figure 4. The With-CS time shows the on-CPU time of a process running with some reference processes in normal architecture. The Run-alone time shows the CPU time of each benchmark running alone. And the CS-RSH time shows the CPU time of each benchmark running with the same reference processes but on RSH architecture. As we can see the CS-RSH time is still slow than ideal(Run-alone). This is because some of the direct overhead like extra kernel code running during context switch cannot be avoided, and RSH switching also has an overhead time causing by RM shift operation and mapping table calculation. On average the RSH architecture can reduce the overhead of CS by 62.5% and gain a 14% speed up in performance.

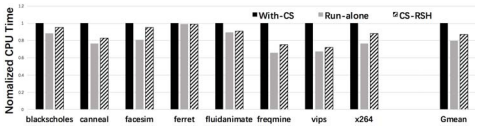

Figure 4: *Normalized CPU time comparison among three situations*

5. CONCLUSION

This paper proposes a racetrack-SRAM hybrid (RSH) cell design. And we also propose a RSH-based method to accelerate the context switch of modern operating systems. The evaluation result shows that this mechanism can effectively reduce the overhead of context switch by reducing the cache miss and TLB miss. On average, the overhead of context switch is reduced by 62.5%, and the performance is improved by 14%.

6. REFERENCES

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, 2008.

[2] W. Kang, Y. Huang, X. Zhang, Y. Zhou, W. Lv, and W. Zhao. Skyrmions as compact, robust and energy-efficient interconnects for domain wall (dw)-based systems. *CoRR*, abs/1601.03085, 2016.

[3] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. In *The Workshop on Experimental Computer Science*, page 2, 2007.

[4] A. Patel, F. Afram, S. Chen, and K. Ghose. Marss: a full system simulator for multicore x86 cpus. In *DAC*, pages 1050 – 1055, 2011.

[5] H. J. Tsai, C. C. Chen, K. H. Yang, and T. C. Yang. Leveraging data lifetime for energy-aware last level non-volatile sram caches using redundant store elimination. In *DAC*, pages 1–6, 2014.

[6] R. Venkatesan, S. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan. Stag: Spintronic-tape architecture for gpgpu cache hierarchies. In *ISCA*, pages 253–264, 2014.

[7] W. Zhao, Y. Zhang, H.-P. Trinh, J.-O. Klein, C. Chappert, R. Mantovan, A. Lamperti, R. Cowburn, T. Trypiniotis, M. Klaui, J. Heinen, B. Ocker, and D. Ravelosona. Magnetic domain-wall racetrack memory for high density and fast data storage. In *ICSICT*, pages 1–4, Oct 2012.