Proceedings of 11th ACM SIGOPS European Workshop, Leuven, Belgium, Sept. 2004

When Physical Is Not Real Enough

Frank Bellosa
University of Erlangen
Department of Computer Science 4 (Operating Systems)
bellosa@cs.fau.de

ABSTRACT

This position paper argues that policies for physical memory management and for memory power mode control should be relocated to the system software of a programmable memory management controller (MMC). Similarly to the mapping of virtual to physical addresses done by an MMU of a processor, this controller offers another level of mapping from physical addresses to real addresses in a multi-bank multi-technology (DRAM, MRAM, FLASH) memory system. Furthermore, the programmable memory controller is responsible for the allocation and migration of memory according to power and performance demands.

Our approach dissociates the aspects of memory protection and sharing from the aspect of energy-aware management of real memory. In this way, legacy operating systems do not have to be extended to reduce memory power dissipation, and power-aware memory is no longer limited to CPUs with an MMU.

1. Introduction

Memory is becoming more and more a target for power management. With systems providing multiple memory technologies (e.g., DRAM, MRAM and FLASH) with individual power levels, it is an appealing idea to employ the optimal memory technology at the best power level for each region of memory according to reference patterns and power/performance demands.

According to precise energy estimation models for DRAM memory, several OS policies have been proposed that save energy by exploiting power states:

- Memory controller policies [1]:
  By monitoring the time-gap between DRAM accesses, the threshold for a transition to a low power state can be determined. Because efficient hardware for an on-line finding of the optimal threshold is not available, the authors refrain from sophisticated policies and recommend to immediately transition to a low power state.
- Page allocation [2, 3]:
  In a multibank memory system, pages of a process are aggregated in a minimal set of memory banks, to keep as many banks as possible in a low-power state while only a few banks are busy. To hide latency the scheduler can give hints to wake up a set of banks before they are used by a process. Special care has to be taken for shared memory regions like dynamically-loaded libraries. DLL aggregation is proposed to avoid scattering effects. With reported energy savings of 42-90% of total memory power for a system with 16 RDRAM devices [2], energy-aware page allocation is a viable approach for systems with a limited number of memory banks and static reference patterns. However it cannot respond to changing access characteristics or co-exist with demand paging.
- Page migration [2]:
  Page migration is used to dynamically aggregate the working set of a process into a fewer number of active memory modules and to overcome the scattering effects of shared memory regions. By applying page migration in a system with energy-aware page allocation, an additional 20-40% of energy savings could be reported [2]. Nonetheless page migration has a number of flaws:
  - The information about page references is counted at the wrong location and with a poor accuracy. Typical MMUs just register if a page has been referenced at all, but not the number of references. To support energy-aware page migration the number of page references resulting from cache misses and those resulting from DMA operations is essential. However, this information is not provided by contemporary architectures.
  - Migration of shared pages requires a complex analysis of the reverse mapping to find candidates for migration. After copying the pages, multiple page tables have to be updated. Both operations are expensive in time and energy.
  - Page migration has to be deeply integrated into the operating system code. Consequently comprehensive restructuring and implementation efforts are required for each operating system to become energy-aware. This is a major hurdle in the

embedded market with its rich flavor of specialized operating systems.

- Page migration requires a memory management unit, which is not available in many low-power controllers.

To overcome the drawbacks of contemporary energy-aware memory-management policies, this position paper describes the architecture and the benefits that arise from relocating physical memory management from the main operating system to the system software of a dedicated memory management controller (MMC).

2. Memory Architecture

The memory management controller (MMC) is located between the system bus and the main memory. A translation unit within the MMC maps physical addresses (issued by the main CPU and the I/O devices) to memory addresses of the associated memory modules (see figure 1). The mapping is done on the granularity of pages.

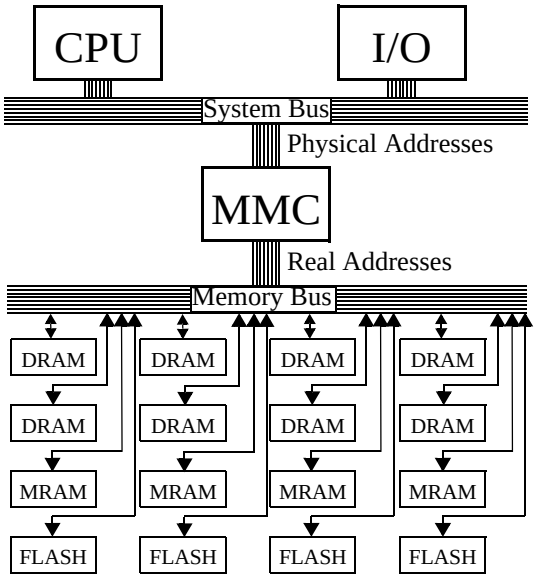

Figure 1: System architecture for energy-aware memory management

The MMC is responsible for the following tasks:

- Mapping from physical addresses to real addresses
- Counting of read- and write-references to real memory pages
- Averaging the access gap for each memory bank
- Managing the power states of each individual memory module
- Swapping memory pages between memory banks of different energy- and access characteristics
- Efficient copying of memory pages on request of the operating system running on the main CPU
- Efficient filling with zeroes
- Compressing/uncompressing of pages

We propose the following architecture for a first implementation of the MMC.

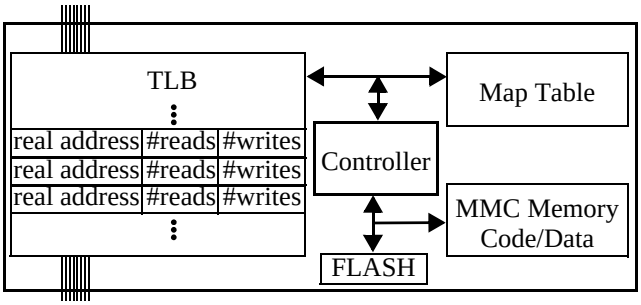

Figure 2: Memory Management Controller (MMC)

- The TLB within the MMC is responsible for the address mapping and the counting of read and write references (e.g., with 32 bit counters). Because the latency of main memory is higher than the latency of caches, performance of the MMC-TLB is not critical in comparison to the performance of a TLB translating addresses for access to a physical cache. Furthermore the reference counts are written back to the map table in case of the eviction of an entry due to a TLB miss or an MMC-TLB flush operation. We recommend a hardware TLB walk through the map table.
- The map table is a direct mapped table for the translation of physical pages to real pages. The size of the physical address space is exactly the size of the real address space. The page size should be small (e.g., 1 KB) to allow fine grained page migration.
- The controller within the MCC does not require its own MMU but it should contain hardware support for energy efficient memory copying and for memory compression/uncompression thereby resurrecting the ideas of hardware compressed main memory [4]. With compression, the portion of the total memory-device density needed for data retention is reduced which offers – after compacting the memory in a module – the opportunity for a partial array self refresh (PASR) [5] to save energy.
- The MMC memory contains code and data of the MMC controller. The controller boots an initial version of its system software from FLASH. However the main operating system has the option to update the MMC system software to instantiate novel interfaces and policies.

3. OS Architecture

The virtual memory management in a classical operating system has to decide which virtual address is mapped to a physical memory region and which physical pages have to be stored and retrieved on/from a backing store.

Our approach to energy-aware memory management is characterized by an additional level of indirection and a clear separation of management policies. While the core OS running on the main CPU is responsible for the aspect of protection and sharing, the energy-aware system software running on the MMC is accountable for an energy-aware mapping of physical addresses to addresses of memory cells in the available memory banks. The mapping should respect the performance- and energy-specific properties of each bank's technology (e.g., DRAM, MRAM, FLASH). Additionally, the MMC system software has to tune the power states of each individual memory module according to power and performance demands.

The memory management in the core OS is active whenever a change in the mapping is required, due to allocation, release, sharing and page-in requests. Periodic activities are limited to low frequency page-out operations to write back modified pages or to provide a certain amount of free memory.

The MMC system software periodically analyzes the reference patterns for each memory bank as well as for each memory page. This information is suitable to determine the threshold for switching a memory bank to a low-power state. Furthermore access characteristics of individual pages are the basis of page migration and aggregation decisions. Additionally unused pages are candidates for compression.

The design space for the system software of the MMC offers several options for an interface to the MMC:

1. The MMC is fully autonomous. The main OS is completely unaware of the energy-aware MMC. The MMC boots from its internal FLASH with default policies and settings, discovers the available memory and behaves like a single memory module from the point of view of the main CPU and the I/O devices.

2. The MMC boots a default system. However there is an interface (e.g., via memory mapped MMC memory and configuration registers) between the main OS and the MMC so that the main OS can download a specific MMC system software and configuration parameters. This approach supports MMC policies which are adapted to the power and performance demands of the system. After the MMC is brought up by the main OS it acts on its own initiative.

3. Additionally to option 2.) there is a run-time interface that permits the main OS to give hints and submit requests to the MMC:
   - When allocating memory, the memory management of the main OS notifies the MMC of physical pages building a contiguous virtual memory region. This hint helps to map these physical pages to real pages of the same module.
   - After releasing a memory region, the memory management of the main OS notifies the MMC of unused physical pages. These unused pages do not have to be copied in case of a page swap between modules and can be mapped to memory areas that do not require refreshing.
   - DMA regions or data/code of time-critical applications should reside in modules with adequate performance settings.
   - Shared regions like DLLs should be aggregated in a small number of modules.
   - The main OS can request efficient copy or zero-filling operations.

4. Benefits

The switching activity in a system has to be minimized to improve energy efficiency. This can be done if all hardware components are performing their job in the optimal mode of operation with a minimal amount of wasted cycles. Our proposal relieves the main CPU from periodic memory reorganizations that interfere with the normal execution of application and operating system code. The job of energy-aware memory management is done by an HW/OS co-design that does not have to make many trade-offs. Our approach is optimized to a few goals:

- Efficient determination of memory reference characteristics
- Efficient mapping of pages to the appropriate location and technology
- Efficient swapping of memory pages
- Efficient memory power-mode control

Additional to the gains in efficiency we see benefits in the support of legacy systems. Extending the memory management of legacy operating systems is an expensive and error-prone venture. Deploying a MMC with default system software, the legacy operating system is right away part of an architecture supporting dynamic power management. Finally, the use of an MMC paves the way for systems originally designed with processors without an MMU to become energy-aware.

5. Conclusions

The more the memory management knows about the reference patterns the better it can improve the placement of data and code in real memory while considering power and performance demands. The main CPU neither offers the required fidelity of reference counting nor the obligatory energy efficiency for fine-grained page-swapping. By separating the energy-aware memory management to a dedicated controller, the main CPU can focus on the tasks it was designed for without additional periodic interruption. Although policy off-loading to a dedicated controller could not keep up with the main processor concerning performance in the past, we believe that memory controllers with sophisticated management policies will prove useful for energy-centric memory management going far beyond the results published in the related work.

References

[1] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for DRAM power management," in *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*, August 2001.

[2] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," in *Proceedings of the 2003 USENIX Annual Technical Conference*, June 2003.

[3] A. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'00*, November 2000.

[4] B. Abali, M. Banikazemi, X. Shen, H. Franke, D. E. Poff, and T. B. Smith, "Hardware compressed main memory: Operating system support and performance evaluation," *IEEE Transactions on Computers*, vol. 50, November 2001.

[5] Micron Technology, "Mobile SDRAM's power-saving features," Tech. Note TN-48-10, 2002.