# Hardware-Assisted Virtual Memory Management

## Improving page replacement and migration with on-line memory access information

Raphael Neider
Karlsruhe Institute of Technology
neider@kit.edu

Frank Bellosa
Karlsruhe Institute of Technology
bellosa@kit.edu

## 1. Motivation

Ever since its inception, virtual memory support in operating systems has relied solely on two bits per page to drive page replacement strategies: The *referenced* bit and the *modified* bit associated with every virtual memory page in the system respectively indicate whether a page has been accessed or (possibly) even modified since the last time these bits had been reset by the OS. On top of this little information, policies such as *least recently used* (LRU) are approximated in software, e.g., using a variant of the two-handed clock algorithm, which is both imprecise and costly.

Due to the rise of new memory technologies with different characteristics than today's DRAM regarding read/write latency, endurance, energy usage, and persistence of data, and due to the integration of such memories into the memory hierarchy, even more placement decisions have to be made by the operating system:

- Read-only pages (e.g., code) can be held in energy-efficient flash memory,
- read-mostly data such as warehouse databases could be held in memory that is cheaper than DRAM in terms of energy or price per volume, similarly fast on reads but slow and/or energy intensive on writes,
- frequently accessed data could be placed into fast SRAM, and
- less frequently accessed data could be placed into DRAM modules that could often remain in a low-power state.

A good decision as to which page to store in what kind of memory technology requires even more detailed knowledge of the memory access patterns at runtime—information that is not available in today's systems.

## 2. Hardware-Assisted Memory Management

We propose to integrate a dedicated hardware module (the "memory profiling unit" or MPU) to monitor all memory accesses and to collect the required information. Such a module can

- record last access times and/or access frequencies per page to facilitate true LRU page replacement.
- record read and write accesses separately to support the selection of the appropriate memory technology per page.
- collect memory traces (e.g., on cache misses) for on-line analysis by the OS to support cache-aware page allocation ("cache coloring").
- collect memory traces and export them for off-line analysis (e.g., system simulation).

Challenges for such a hardware module are numerous:

- How/where can the timestamps/access counters be stored? With 4 GiB of memory and 4 KiB pages, 1 mio. counters need to be stored and updated frequently.
- What should the interface to the OS be? If we simply expose 1 mio. counters to the OS, finding good candidates for page replacement or migration is still costly. On the other hand, letting the hardware select a number of candidates is easy for the most recently/frequently accessed pages, thus supporting page migration. Finding the least recently/frequently pages for page replacement is more difficult, since the candidate list cannot be updated only on memory accesses.
- If the MPU is to export memory access traces, how can the required bandwidth be reduced to a feasible level?

While we envisage the MPU to collect physical memory references in order to support page replacement/migration and thus locate the MPU after the caches/right in front of the memory controller, one could also investigate uses of placing the MPU before the TLBs to trace program references or after the TLBs but before the caches to record physical memory references without cache effects.

## 3. Prototype

Research in this area is hindered by the fact that today's computing systems are not easily extensible at their memory access path. For that reason, we developed the OPENPROCESSOR platform [1], an FPGA-based system-on-chip with a RISC CPU core, separate software-controlled translation look-aside buffers for instructions and data, separate caches, a DDR memory controller, several I/O device controllers to interact with a host computer, and a bus interconnect between the components. All parts of the platform, especially the memory access paths, are easy to extend or adjust to facilitate research in this area.

We have augmented the OPENPROCESSOR platform with a prototypical implementations of a memory profiling unit and are currently conducting research on all of the issues mentioned above.

## References

[1] R. Neider. The OpenProcessor platform: Fostering research on the hardware/software boundary. Technical Report TR 2011,1, Karlsruhe Institute of Technology, Department of Informatics, Jan. 2011. URL http://digbib.ubka.uni-karlsruhe.de/volltexte/1000021677.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*ASPLOS '11* March 5–11, 2011, Newport Beach, California
Copyright © 2011 ACM . . . $10.00