## Guest Editorial

# Cache exploitation in embedded systems

Jingling Xue
*School of Computer Science and Engineering, The University of New South Wales, 2052 Australia*
*Tel.: +61 2 9385 4889; Fax: +61 2 9385 5995; E-mail: jxue@cse.unsw.edu.au*

Memories determine to a large extent the performance and energy cost in contemporary embedded systems. In most high-end embedded systems, on-chip instruction and data caches are introduced to improve the performance and energy of a program by exploiting the available locality in the program. Hardware-managed caches allow easy integration and are effective for applications that exhibit sufficient locality in their memory accesses. However, applications with excessive cache misses often suffer from poor performance and energy efficiency. Scrachpads are more energy-efficient than caches since they do not need complex tag-decoding logic. Unlike caches, however, scratchpads require explicit support from the compiler. To exploit the benefits of these two approaches, some high-end embedded microprocessors such as ARM10E and ColdFire MCF5 include both on-chip caches and a scratchpad.

Unlike in the case of general-purpose desktop systems, power consumption is an essential concern for embedded systems and is also an important design constraint where power supply capacity is limited. In addition, embedded systems are designed to run a set of well-defined applications, which can be highly tuned by the application programmer.

This special issue aims at providing a snapshot of the current state of the art in the areas of cache analysis techniques, energy-aware code optimizations, and energy-efficient cache architectures for improving the performance and energy of embedded applications. The first two papers are related to instruction caches, the next four papers on data caches, and the last paper on scratchpads.

In "Procedure Placement using Temporal-Ordering Information: dealing with Code Size Expansion," **Guillon**, **Rastello**, **Bidault** and **Bouchez** address the intelligent procedure placement in a direct-mapped instruction cache, for reducing conflict misses. Reducing conflict misses without resulting in a large code size expansion is important for reducing energy costs in embedded systems. The authors improve Gloy and Smith algorithm by making it conscious of code size expansion. They prove that the cache-placement phase is NP-complete and that the memory-placement phase is optimal. They obtain nearly the same cache miss reduction with far less code size increase (only 8% compared to previous 177%). Moreover, this work provides a useful theoretical underpinning for the approach used.

The instruction cache on modern microprocessors consumes a significant fraction of the total processor power. Tag comparison elimination (TCE) techniques have been proposed to achieve power reduction by avoiding unnecessary tag comparisons. In their article, "Reducing I-cache Energy of Multimedia Applications through Low Cost Tag Comparison Elimination," **Zhang** and **Yang** present two TCE solutions to reduce the hardware cost of TCE in the design of a low-energy instruction cache.

Some embedded processors such as Intel StrongARM SA-1110 and Intel XScale include a mini-data cache to help avoid thrashing of the main data cache for frequently changing data streams. In addition, these processors also provide flexible control over the cache management to achieve better cache utilization. Programs can specify the cache mapping policy for each virtual page by mapping it to the main cache, the mini-cache or neither. In "Page Mapping for Heterogeneously Partitioned Caches: Complexity and Heuristics," **Li** and **Xu** deal with the problem of deciding where virtual pages are mapped, in the main cache, mini-cache or by-passed from main memory. They show that the optimal cache mapping assuming the trace of all memory access is known is NP-complete. In addition, they provide a mapping heuristic that improves the performance and energy of benchmark programs compared to the default policy that maps all virtual maps to the main data cache.

In “Improving Power Efficiency with Compiler-Assisted Cache Replacement,” **Yang**, **Govindarajan**, **Gao** and **Hu** present a compiler approach to making a better data cache replacement decision for processors that support cache hints or cache locking/unlocking mechanism. They formulate the optimization problem as a 0/1 Knapsack problem and solve it using a dynamic programming algorithm. In comparison with data prefetching on an Intel XScale processor, their approach yields improved performance at a similar or reduced level of power consumption.

Leakage energy consumption in caches is significant since they contain a significant fraction of the on-chip transistors in a microprocessor. In “Exploiting Loop Behavior for Data Cache Leakage Reduction,” **Zhang** introduces a compiler-directed approach to reducing the data cache leakage energy for loop-oriented programs. By placing cache lines into low leakage mode during the execution of the innermost loops, the author shows that this software approach is competitive in terms of improved energy consumption and energy-delay product, compared to a recently proposed hardware-based solution.

In “Iterative Compilation for Energy Reduction,” **Gheorghita**, **Corporaal** and **Basten** study the applicability of iterative compilation to reduce energy consumption, extending previous work on iteration compilation that looked at performance only. They show that in most cases, optimizing for performance and optimizing for energy can coincide, and sometimes there can be a noticeable difference between the two objectives. Thus, they recommend a combined energy-performance factor to be used to evaluate the compiled code in order to have good compromises.

In “Heap Data Allocation to Scratch-Pad Memory in Embedded Systems,” **Dominguez**, **Udayakumaran** and **Barua** present a compiler-assisted approach to allocating a portion of the heap data to scratchpad memory. When compared to placing all heap variables in DRAM and only global and stack data in scratchpad for the same-sized scratchpad, they achieve better average execution times and energy savings for benchmark programs. In addition, their approach equals or slightly outperforms a cache-only architecture and provides slightly better runtime and energy in a cache $+$ scratchpad architecture.

Finally, I would like to thank the authors of all submitted papers, including the authors of papers that could not be included in this special issue. The tight publication schedule did not allow any extensive revisions, so several good papers could not be included due to the requests for such revisions by the reviewers. I would also like to thank the reviewers of the submitted papers for their dedication and effort in providing timely and thorough reviews. This special issue would not have been possible without their hard work. I hope that the readers find this issue enlightening and enjoyable.