Design Space Exploration and Optimization of Embedded Cache **Systems via a Compiler**

KRISHNA V. PALEM and RODRIC M. RABBAH
Georgia Institute of Technology

Recent years have witnessed the emergence of custom microprocessors that are embedded within a plethora of devices used in everyday life. While custom embedded solutions provide major opportunities for performance enhancement via instruction level parallelism, they magnify the needs for sustainable high rates of data delivery to the processing units. Hence we believe that the same trend of customization that is becoming prevalent in the microprocessor domain is just as important (if not more important) when one considers the design of a supporting memory or cache hierarchy. To a large extent, the design of embedded memory systems remains an ad hoc art, often relying on intuition and extensive simulations to choose the best match for a particular processing unit [1, 11]. It is our contention however, that the memory organization is a major specialization dimension that warrants careful consideration when application specific computing solutions are necessary.

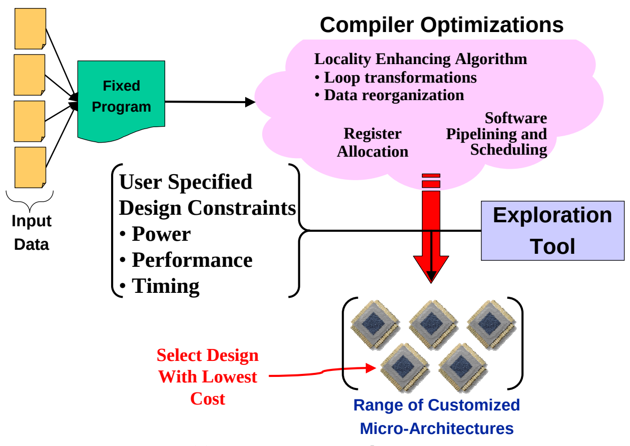
The memory hierarchy has been a ubiquitous component in the design of computing platforms since the introduction of the von Neumann machine [2, 12]. It has widely served to bridge the performance gap between processor and memory system, usually by employing deep cache hierarchies where each level trades off capacity for access speed. Unfortunately, as processors are increasingly used in the context of embedded systems, the cost of the memory hierarchy has been a limiting factor in its ability to play as central a role. Quite often, this limitation is because of the physical size as well as the implications to the complexity of the processors used in the embedded domain. As a result, the impact on the overall performance of a system can be overwhelming. Namely, given the massive parallelism that may be available in the custom processor, long access latencies dramatically decrease processor throughput, and hence strongly warrant the need for latency masking and ameliorating techniques.

Recognizing the need to overcome the stated limitation, we have proposed several novel and lightweight compiler optimizations [4, 6, 8, 9, 10] to help ameliorate the memory bottleneck. Furthermore, we are able to show that an innovative *data remapping* technique [9] allows a program to achieve the same overall running time with just half the cache resources, when compared to a program that has not been remapped. Hence, simpler machines can achieve the same level of performance as more complex and expensive machines in the absence of our techniques. Stated in simple terms, remapping is a reorganization of the application's data in memory, such that memory elements that are accessed contemporaneously are in fact placed together in memory. *Thus, remapping aims to improve the spatial locality of memory elements that in fact also share temporal locality.* The simple transformation achieves its impressive improvement by ensuring that the ratio of the number of items found in cache (cache-hits) to those that are fetched from main memory (cache-misses) remain the same, with half the memory size, without compromising the application execution time. Such a result offers the intriguing possibility of a *compiler playing a significant role in exploring and optimizing the design space of a custom memory system*. As shown in Figure 1(a), design space exploration involves exploring alternate hardware or architectural solutions to meet a specified performance constraint for a *fixed program* $\mathcal{P}$. Thus while the program is fixed, the optimization techniques are applied to find the hardware solution subject to the given performance or execution time constraint. By contrast and as shown in Figure 1(b), the dual of this problem is the domain of a traditional compiler optimization, wherein the applications or programs $\mathcal{P}_1, \mathcal{P}_2 \ldots \mathcal{P}_k$ are optimized to achieve the best possible performance on a *fixed target processor*. In Table 1, we illustrate how our technique may be regarded as a design space exploration tool which can help lower the memory of a custom design. In particular, given a "user-specified" performance constraint, remapping can yield a system meeting this goal whose cost is significantly

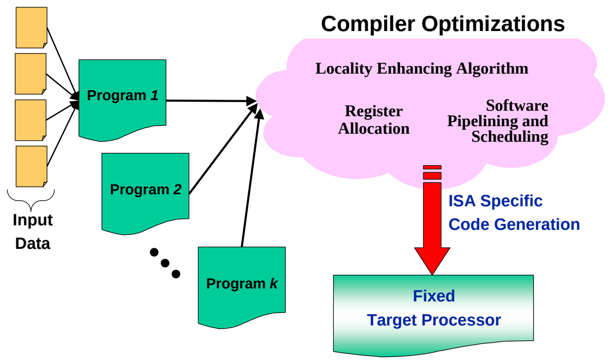
A complete version of this work will appear in a special issue of the ACM Transactions in Embedded Computing Systems. This work is supported in part by DARPA contract F33615-99-1499, Hewlett-Packard Laboratories and Yamacraw.

Table 1. Example impact of data remapping on cost during design space exploration.

Benchmark	Benchmark Suite	Performance Goal 10^6 Cycles	Before Remapping L2 Size	Before Remapping L2 Cost	After Remapping L2 Size	After Remapping L2 Cost	Saving
179.ART	SPEC2000	13,000	1.0Mb	\$19.38	0.0Mb	\$ 0.00	\$19.38
TREEADD	OLDEN	880	1.0Mb	\$19.38	0.5Mb	\$17.80	\$ 1.58
PERIMETER	OLDEN	520	2.0Mb	\$48.00	1.0Mb	\$19.38	\$28.62



(a) Design space exploration.



(b) Traditional compiler optimization.

Figure 1. Viewing design space exploration and compiler optimizations as duals of each other.

lower than that of a corresponding system satisfying the same goal, but without the benefits of remapping. For example, given a fixed execution time goal of 600 million cycles for the application PERIMETER, remapping allows us to use an L2 memory size of 1Mb instead of 2Mb for a total saving of \$28.62 which is 60% of the cost[1]. As another example, for the benchmarks 179.ART, data remapping allows the use of a memory hierarchy that only consists of a primary cache. More generally, we have demonstrated that this improvement is consistent in the context of several applications including floating-point and integer applications from the DARPA DIS suite as well as the well-known OLDEN and SPEC2000 suites.

In order to use the savings afforded by our compiler strategies as crucial steps in exploring the design space of an embedded cache system, we note that it is important to quantify the potentials for improvement using some fundamental variables over which the exploration and optimizations can proceed. *In a recent paper to appear in the ACM Transactions in Embedded Computing Systems, we provide a novel characterization of a set of such variables* [9]. For example, while prefetching is widely accepted in cache systems, its efficacy is clearly dependent on being able to match its prediction and replacement policies with the actual needs of the program. To this end, we quantify the degree of mismatch between the prefetchers in the architecture and the memory access patterns of the program. In addition, using our measures, we can quantify the improvements achieved by our optimizations along this dimension. Our methodology extends the current state-of-the-art design space exploration and optimization techniques [3, 5, 7] to pointer-centric applications ubiquitous to *C*-based applications and hence to embedded systems, and it will become increasingly important as the needs for application specific architectures become prevalent.

[1]For the 2MB L2 cache, we use two 1Mb Toshiba TC55W800FT-55 SRAM chips, each at a cost of \$24. For the 1Mb L2 cache, we use two 0.5Mb Toshiba TC55V400AFT7 SRAM chips, each at a cost of \$9.19. For the 0.5Mb L2 cache, we use four 128Kb Cypress CY62128VL-70SC SRAM chips, each at a cost of \$4.425.

REFERENCES

[1] S. Abraham and S. Mahlke. Automatic and efficient evaluation of memory hierarchies for embedded systems. In *Proceedings of the 32nd Annual International Symposium on Microarchitecture*, pages 114–125, Nov. 1999.

[2] A. Burks, H. Goldsteind, and J. von Neumann. Preliminary discussion of the logical design of an electronic computing instrument. Papers of John von Neumann, 1987.

[3] F. Catthoor, S. Wuytack, E. DeGreef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology. Exploration of Memory Organization for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.

[4] C. Hardnett, R. Rabbah, K. Palem, and W. Wong. Scheduling of load instructions for epic processors. Technical Report GIT-CC-02-18, Georgia Institute of Technology, Mar. 2002.

[5] C. Kulkarni, F. Catthoor, and H. Man. Advanced data layout organization for multi-media applications. In *Proceedings of the Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*, May 2000.

[6] K. Palem, R. Rabbah, V. Mooney, P. Korkmaz, and K. Puttaswamy. Design space optimization of embedded memory systems via data remapping. In *Proceedings of the Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems*, June 2002.

[7] P. Panda, N. Dutt, and A. Nicolau. Memory data organization for improved cache performance in embedded processor applications. *ACM Transactions on Design Automation of Electronic Systems*, 2(4):384–409, 1997.

[8] R. Rabbah, M. Ekpanyapong, W. Wong, and K. Palem. Ameliorating the memory bottleneck by speculative execution: The load dependence graph. Technical Report GIT-CC-02-22, Georgia Institute of Technology, Apr. 2002.

[9] R. Rabbah and K. Palem. Data remapping for design space optimization of embedded cache systems. Technical Report GIT-CC-02-10, Georgia Institute of Technology, Mar. 2002. To appear in the ACM Transactions in Embedded Computing Systems, 2002.

[10] R. Rabbah, K. Palem, V. Mooney, P. Korkmaz, and K. Puttaswamy. Design space optimization of embedded memory systems via data remapping. Technical Report GIT-CC-02-11, Georgia Institute of Technology, Mar. 2002.

[11] B. R. Rau and M. Schlansker. Embedded computing: New directions in architecture and automation. Technical Report HPL-2000-115, Hewlett Packard Laboratories, Sept. 2000.

[12] A. Taub. *Collected Works of John von Neumann*. The Macmillan Company, New York, 1963.