# New Directions in Compiler Technology for Embedded Systems [†]

Nikil Dutt  Alex Nicolau  Hiroyuki Tomiyama[‡]  Ashok Halambi
Architectures and Compilers for Embedded Systems (ACES) Laboratory
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
{dutt, nicolau, tomiyama, ahalambi}@cecs.uci.edu
http://www.cecs.uci.edu/~aces

**Abstract— Traditionally, compiler technology has focused on the generation of code with the goal of improving performance for a variety of applications running on general-purpose processor architectures. In the embedded system space, compiler technology is faced with many new challenges, including: code generation for specialized architectural features, requiring a highly flexible degree of retargetability; memory-aware code generation that exploits the timing and structure of the embedded system's memory organization; optimizing software to meet both real-time and performance constraints; energy- and power-aware software generation, both from the context of energy minimization, as well as power modulation; code size minimization for memory-constrained embedded systems; coarse-grain transformations for tightly-coupled, memory-constrained multi-processor architectures; and interaction with the operating system for active management of embedded system resources. This paper discusses new directions for compiler technology, surveys some of the current research efforts and illustrates proposed solutions to selected issues.**

## I. Introduction

Conventional compiler technology has reached a level of maturity and acceptance in the software, architectural and system world. For commodity processor platforms, manufacturers often provide a software development environment that encapsulates compilers for frequently used languages such as C, C++ and Fortran. Many third-party software vendors offer additional off-the-shelf compiler products that work with commodity processor platforms. Traditional compiler design is now a requirement for any computer science and engineering undergraduate curriculum, and several standard textbooks exist on this topic [1, 2]. Furthermore, several open-source compiler infrastructures (e.g., GNU CC[3] and LCC[4]) allow experts, as well as novices to learn, experiment, and tune code generation back-ends for common processor platforms. Hence conventional thinking may lead us to believe that there is nothing new in compiler technology, or that no further research is warranted on this topic.

While this is partly true for PC-based processor platforms, the situation is *entirely different* for embedded-processor-based systems. Embedded system design brings to bear a whole new set of design goals, constraints, architectural and system organizations that require a critical rethinking of how we approach compiler technology. Indeed, there are many new issues and directions that compiler technology must address to satisfactorily support the design, development and deployment of programmable embedded systems. To name a few, these include: heterogeneous and specialized architectural features; memory-aware compilation; real-time issues for compilers; energy- and power-aware software generation; and interaction with the run-time/operating system for the embedded application. In this paper we discuss some of these challenges, survey ongoing research efforts in solving the problems, and identify open problems that need further research.

## II. Compiler Technology for Embedded Systems

### A. Code Generation for Specialized Architectures

Embedded Systems processors typically have very irregular structures which makes it hard to produce good quality code using traditional compiler techniques. For example, DSP processors typically exhibit a limited amount of instruction-level parallelism, and have special-purpose register, address generation units, etc. Recent processors for embedded systems tend to combine features from different processor classes (such as DSP, VLIW, RISC/Superscalar) to form a hybrid processor with superior performance/cost benefits. An example of such a processor is the Texas Instruments' TMS320C6201 [5]

[†] This work is supported in part by grants from NSF (MIP-9708067), DARPA (F33615-00-C-1632), and Motorola Inc.

[‡] Research at UCI is supported by JSPS postdoctoral fellowships for research abroad.

VLIW processor with DSP features (such as partitioned register files).

The traditional retargetable compiler work assumes a fairly fixed model of the architecture with the ability to change locally certain architecture features. However, such retargetable approaches typically cannot generate good quality code for emerging hybrid architectures. Recent work on retargetable compilers has focused on DSP, VLIW and RISC/Superscalar architecture models.

In the DSP domain, research has focused on efficient instruction selection, register allocation for partitioned register files, and also address computation techniques. Some projects which have resulted in compilers for the DSP domain include the CodeSyn project, RECORD compiler project using the MIMOLA [6] Architecture Description Language (ADL), the CHESS [7] compiler project using the nML [8] ADL and projects based on the LANCE front-end. RECORD and CHESS compilers tackle the dual goal of retargetability and quality code generation for DSPs. The LANCE [9] front-end has been used to develop back-ends for many DSPs including TI C5x and the M3 SIMD DSP. In the VLIW domain, research has focused on developing instruction level parallelism (ILP) techniques, and handling SIMD. The Trimaran [11] compiler using MDes ADL produces code for a parameterizable EPIC architecture. In the RISC/Superscalar domain, research has focused on conditional execution (predication) models that help alleviate the problem of large branch delay cycles. The LANCE system has been used to develop support for conditional instructions, while Trimaran handles predicated execution in EPIC architectures. Software tools that handle the new, hybrid style architectures include the TI compiler for C62X, and the EXPRESS/EXPRESSION [12] project that incorporates speculative and predicated execution techniques and aims to provide compiler retargetability for a wide variety of architecture domains.

Another aspect of code generation is the difficult problem of ordering the mutually dependent phases of Instruction Scheduling, Register Allocation and Scheduling. Traditionally, compilers have used a fixed phase ordering based on the heuristics used to drive the general purpose processor design. However, for embedded systems that employ heterogeneous architectures, there is a critical need for customizing the phase ordering based on the relative importance of different architecture resources and constraints. Several techniques have been proposed for (partially) solving the phase-ordering problem. These include the integrated register allocation and ILP scheduling techniques ([13, 14]), and Mutation Scheduling [15] which dynamically adapts code to conform to the resource constraints.

As embedded system processors increasingly adopt hybrid architectural styles and novel memory organizations, highly retargetable compilers capable of generating code for various constraints (such as power, code size, etc.) become very important tools in the system designer's arsenal. An ADL based approach thus becomes critical for specifying the detailed architecture, and for use in generation of a high quality toolkit [16, 12]. In the next few sections, we will analyze the issues and problems currently being researched, and also briefly investigate new directions in the area of code generation for various constraints.

## B. Memory-Aware Code Generation

Traditionally, DRAM has been used as main memory for many systems and has been located outside the processor-based Systems-on-Chip (SOCs) as a separate chip. Since the gap of speed between processor and DRAM has been increasing every year, memory accesses have been a major bottleneck in high-performance SOCs. To fill the gap, many SOCs employ instruction cache and data cache in the same chip. Since cache misses require extra-cycles and energy to transfer code or data between the caches and the main memory, minimization of cache misses is a crucial problem in code generation.

Several code placement techniques have been proposed for minimization of instruction cache misses, and earlier works are summarized in [2] and [17]. For example, Hwu and Chang presented some techniques: *function inlining* replaces function calls with high execution frequency with their function body; *trace selection* places basic blocks in sequence which tend to execute in sequence; *global layout* places functions in sequence which are close to each other in the execution order [18]. McFarling proposed more sophisticated techniques for function layout and function inlining [19, 20]. Code placement techniques proposed in [21] aggressively exploit program profiles and globally place basic blocks using an integer linear programming formulation.

So far a number of compiler optimization techniques for improving data cache performance were proposed. Most of them are targeted to loops which manipulate large arrays. [2], [17], and [22] gives good tutorials on previous work: *loop interchange* exchanges an inner loop with an outer loop; *loop fusion* combines multiple loops into a single loop; *loop unrolling* expand a loop by copying the loop body and reducing the number of iterations; *loop tiling* (or *blocking*) divides the iteration space into *tiles* (or *blocks*) so that the data set involved in a computation fits into the data cache; *software prefetching* predicts data which will be accessed in future and inserts an instruction which loads the data into the cache. Recently, Panda et al. proposed a memory location alignment technique for arrays as well as scalar variables [22]. Then, the technique was extended so that it can be well incorporated into loop tiling. More recently, Grun et al. presented a new compiler optimization which uses accurate memory access timing information for both cache hits and misses, and schedules instructions so that memory accesses are efficiently overlapped [23].

In addition to caches, some SOCs may have Scratch-Pad memory which is a fast SRAM residing on-chip. Since Scratch-Pad memory is mapped into an address space disjoint from the off-chip memory, it guarantees a single-cycle access, whereas an access to the cache may be missed. Therefore, properly allocating a data set into Scratch-Pad memory and main memory reduces execution time and energy consumption. Panda et al. also proposed such techniques in [22].

In some cases, DRAM (which resides on-chip or off-chip) may be accessed directly from the processor. Contemporary DRAMs exhibit efficient access modes such as page mode, burst mode, and pipelined access, in addition to random access. Efficiently utilizing these access modes can drastically improve system performance. Grun et al. presented such a compiler optimization technique which extracts accurate timing of DRAM accesses and schedules instructions so that these access modes are efficiently utilized [24].

Since memory has been (and will continue to be) a primary driver in terms of performance, power/energy, and silicon area, the importance of memory-aware compilation has also been increasing. So far, various kinds of memory modules in terms of access protocols as well as memory devices have been developed, and new memory modules will continue to emerge in reality. Therefore, future memory-aware compilers will have to exploit the timing behavior of these memory modules for better optimization.

## C. Software Optimization for Performance and Real-Time Constraints

Traditionally, most efforts on compiler optimization have aimed at performance improvement in terms of average execution speed. Such optimization techniques have been extensively studied in a large number of literature, for example [1], [2], and [17]. However, many embedded systems are real-time systems in which tasks have their own timing constraints called deadlines. In the design of such systems, analyzing and bounding the Worst-Case Execution Time (WCET) is more important than minimizing average-case execution time. WCET analysis for a single task has been of great concern and some recent work such as [25] and [26] accounts for cache effects in their WCET analysis. Up to now, however, few research efforts have explicitly addressed compiler technology for WCET minimization.

A real-time system generally consists of a set of tasks. Some tasks may have timing constraints in terms of execution rates (or periods) and/or deadlines. Some other tasks may not have explicit timing constraints, but must be executed so that the timing constraints of the other tasks are all satisfied. Then, the goal of the compiler is, given program codes of tasks and timing constraints, to compile and optimize the codes so that all the timing constraints are satisfied. To achieve this, first, timing constraints of all tasks need to be derived from partially specified timing constraints. In other words, given an overall time budget, we need to distribute the time budget to the individual tasks. Dasdan has studied this problem in [27]. Then, each task must be compiled and optimized so that its WCET never exceeds the time budget assigned to the task. To do this, the time budget need to be further decomposed into function-, thread- or basic block-level so that state-of-the-art compiler optimization techniques can be applied.

This problem becomes further complicated in case of preemptive systems where a task may be preempted by higher priority tasks during its execution. The interval from a task's arrival time to its completion time is called response time, which includes not only the execution time of the task but also the execution time of preempting tasks. In the design of preemptive real-time systems, therefore, analyzing and bounding the Worst-Case Response Time (WCRT) is important. For more than a decade, WCRT analysis has been studied, and some recent work takes account of inter-task cache conflicts [28, 29]. Also, an attempt has been made to minimize inter-task cache conflicts (hence, WCRT) by properly placing tasks' codes into main memory [30].

Analysis of WCET and WCRT has been extensively studied by many researchers for a long time. On the other hand, however, code generation for bounding or minimizing WCET and WCRT has not been addressed so far. Therefore, there remain a lot of opportunities for compiler optimization in this area. In order to better optimize the real-time software, interactions with real-time operating systems (RTOSs) will be necessary because the WCRT is affected by the scheduling policy, the overhead for context switching, and some other features of the RTOS. It will be also effective, especially for Just-In-Time (JIT) compiler systems, to utilize feedbacks from the RTOS about the dynamic behavior of the software and to optimize it on-the-fly.

## D. Code Size Minimization Techniques

SOCs are usually very constrained for memory space, both for storing instructions and for recording/manipulating data. Therefore, there has been tremendous interest in reducing memory size by reducing the code size and/or by reusing storage locations.

A lot of traditional compiler techniques for performance improvement (e.g., ILP scheduling, loop unrolling, etc.) also increase the code size. Therefore it is necessary for the system designer to trade-off code size with performance while using these techniques. Recently, there has been some research in memory size reduction techniques that do not compromise performance. These include the utilization of SIMD features to pack data more efficiently [9], optimal address computation (using auto-increment and decrement features) to increase performance and also reduce code size [10], etc.

The problem of data placement and code scheduling to reduce the maximum data memory requirement has been extensively studied. The problems that have been studied include array placement and array access ordering to minimize memory, and loop transformations that order accesses with the objective of reducing memory. [31] presents techniques for memory size estimation of systems containing multi-dimensional arrays and parallel constructs. [32] presents a technique for memory size reduction through storage order optimization for multimedia applications.

Code compression is another technique (peripherally related to code generation) that encodes the binary code and thus reduces size in memory. [33] provides a comprehensive survey of code compression techniques.

As discussed in earlier sections, memory issues will continue to dominate the problem space for embedded systems. The system designer needs tools and techniques that will allow him to perform trade-offs between memory-size, power and performance.

## E. Coarse-Grain Transformations

With the continuing explosion in the number of devices on a chip, future embedded systems will continue to use a variety of complex, customized SOCs that will employ a moderate degree of on-chip multiprocessing. This, coupled with the move towards increasing instruction word length, allows both a high degree of coarse-grain parallelism (between processors), as well as increased instruction-level parallelism (ILP). Initial research has focused on migrating the compilation techniques developed in the traditional multi-processor domain to multi-processor systems in the embedded systems domain. These include coarse-grain techniques for task allocation/scheduling and source-to-source transformations for enhanced parallelism. [34] presents a framework to exploit coarse- and fine-grain parallelism in embedded systems using source-to-source transformations. Most previous research has addressed improving performance and not code size or power for parallel processor systems. One work that addresses power reduction in the context of multi-processor systems is [35] which presents a combined task and data parallelism technique for reducing power.

With the introduction of on-chip multi-processing, the need for aggressive instruction-level speculative compiler technology, coupled with coarse-grain parallelization technology, will become critical. One consequence of this will be that an even tighter integration of traditionally higher-level transformations with ILP transformations will be required in a high quality compiler, to effectively utilize the multiple levels of parallelism available in these machines. Also, new techniques will have to be developed to take advantage of new features (e.g., fast communication/synchronization) of these multi-processors on a chip. Integration of operating systems with compiler technologies will become increasingly important for such embedded multi-processors.

## F. Energy- and Power-Aware Software Generation

The traditional goals of a compiler were either to generate code that resulted in improved processor performance, or to minimize the compilation time for an acceptable level of processor performance. Programmable embedded systems, on the other hand, often require careful attention towards power dissipation and energy consumption.

From the viewpoint of compilers, several opportunities for power reduction/management exist, particularly for multimedia embedded systems. Such applications are typically characterized by data-intensive computations operating on (multi-dimensional) arrayed data structures stored in off-chip memories. Thus compiler transformations that aim to reduce off-chip memory traffic often simultaneously improve performance, while reducing power.

Early experiments by Tiwari et al. [36] demonstrated reduced energy consumption (and higher performance) through improved register allocation, resulting in fewer spills to memory. Compiler techniques that improve data locality through coarse-grain transformations [37] and data layout optimization [22, 38, 39], result in significantly fewer cache misses, leading again to improved performance and lower power dissipation. Similarly, instruction scheduling techniques to reduce instruction cache misses have been developed [40], resulting in reduced bus transitions per off-chip memory transfer. Recent work in memory-aware compilation [23, 24, 41] aims to better exploit memory access protocols of contemporary DRAMs for improving the memory bandwidth of applications.

The effects of such compiler optimizations (as well as many other contemporary compiler transformations) on power dissipation require a comprehensive measurement or simulation environment, since the relationship between performance and power or energy is not easily predictable. New efforts in building architectural power/energy-aware simulation [42, 43, 44] will help quantify the effects of compiler optimizations on power and energy.

Finally, compiler-controlled power management techniques are beginning to appear [45, 46], that dynamically tradeoff power for performance. While energy/power minimization is one important goal for many embedded systems, energy/power *management* is another very important goal. Embedded systems will need to follow specific power/energy profiles, depending upon available power/energy and OS policies for dynamic power/performance management. The compiler, through a combination of static analysis, profile-driven data and feedback-driven optimization, can thus modify the power/performance characteristics of an architecture, in consort with system-level power management schemes. Thus a tighter interaction between the runtime/operating system and the compiler technology is an emerging area of research.

## III. Summary

Programmable embedded systems run the gamut from simple embedded microcontroller-based systems to complex multi-processor based systems dedicated for demanding applications. Such embedded systems exhibit novel architectural features, are often memory-constrained, must perform within tight real-time constraints, have to be power/energy efficient, and must do all of this within an acceptable level of performance. These constraints open up a new set of problems in the compiler domain. In this paper we outlined and surveyed some of the new directions faced by compilers for embedded systems. Many other issues remain open, including the incorporation of reliability metrics, dynamic adaptation of code based on environmental constraints, and integration of compilation issues within a larger networked, embedded infrastructure.

## Acknowledgment

The authors would like to thank members of the ACES Laboratory: Jonas Astrom, Partha Biswas, Radu Cornea, Peter Grun, Illya Issenin, Mahesh Mamidipaka, Prabhat Mishra, Nick Savoiu, Aviral Shrivastava and Srikanth Srinivasan for their contribution to this work and the EXPRESS/EXPRESSION project. The authors would also like to thank all the members of the Center for Embedded Computer Systems (CECS), Irvine for interesting discussions.

## References

[1] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addition-Wesley, 1986.

[2] S. S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann Publishers, 1997.

[3] R. M. Stallman, *Using and Porting GNU CC (for version 2.95)*, Free Software Foundation, Inc., 1999.

[4] C. Fraser and D. Hanson, *A Retargetable C Compiler: Design and Implementation*, Addison-Wesley, 1995.

[5] Texas Instruments, *TMS320C62x/C67x CPU and Instruction Set Reference Guide*, 1998.

[6] S. Bashford, U. Bieker, B. Harking, R. Leupers, P. Marwedel, A. Neumann, and D. Voggenauer, *The MIMOLA Language Version 4.1*, University of Dortmund, 1994.

[7] D. Lanneer, J. Van Praet, A. Kifli, K. Schoofs, W. Geurts, F. Thoen, and G. Goossens, "CHESS: Retargetable code generation for embedded DSP processors," *Code Generation for Embedded Processors* (P. Marwedel and G. Goossens, ed.), Kluwer Academic Publishers, 1995.

[8] M. Freericks, *The nML Machine Description Formalism*, Fachbereich Informatik, TU Berlin, 1991.

[9] R. Leupers, *Code Optimization Techniques for Embedded Processors: Methods, Algorithms, and Tools*, Kluwer Academic Publishers, 2000.

[10] S. Liao, S. Devadas, K. Keutzer, S. Tjiang, and A. Wang, "Storage Assignment to Decrease Code Size," *ACM TOPLAS*, Vol. 18, No. 3, 1996.

[11] Trimaran Release, "The MDES user manual," *SIA NTRS 1997 Edition*, http://www.trimaran.org, 1997.

[12] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau. "EXPRESSION: A language for architecture exploration through compiler/simulator retargetability," In *Proc. of DATE*, 1999.

[13] A. Nicolau, R. Potasman, and H. Wang, "Register allocation, renaming and their impact on parallelism," *Languages and Compilers for Parallel Computing*, 1994.

[14] D. Berson, R. Gupta, and M. L. Soffa, "Resource spackling: A framework for integrating register allocation in local and global schedulers," In *Proc. of PACT*, 1994.

[15] S. Novack and A. Nicolau, "Mutation scheduling: A unified approach to compiling for fine-grain parallelism," *Languages and Compilers for Parallel Computing*, 1994.

[16] A. Halambi, P. Grun, H. Tomiyama, N. Dutt, and A. Nicolau, "Automatic software toolkit generation for embedded systems-on-chip," In *Proc. of ICVC*, 1999.

[17] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd edition, Morgan Kaufmann Publishers, 1996.

[18] W. W. Hwu and P. P. Chang, "Achieving high instruction cache performance with an optimizing compiler," In *Proc. of 16th ISCA*, 1989.

[19] S. McFarling, "Program optimization for instruction caches," In *Proc. of 3rd ASPLOS*, 1989.

[20] S. McFarling, "Procedure merging with instruction caches," In *Proc. of PLDI*, 1991.

[21] H. Tomiyama and H. Yasuura, "Code placement techniques for cache miss rate reduction," *ACM TODAES*, Vol. 2, No. 4, 1997.

[22] P. R. Panda, N. Dutt, and A. Nicolau, *Memory Issues in Embedded Systems-on-Chip*, Kluwer Academic Publishers, 1999.

[23] P. Grun, N. Dutt, and Alex Nicolau, "MIST: An algorithm for memory miss traffic management," In *Proc. of ICCAD*, 2000.

[24] P. Grun, N. Dutt, and A. Nicolau, "Memory aware compilation through accurate timing extraction," In *Proc. of 37th DAC*, 2000.

[25] Y.-T. S. Li and S. Malik, *Performance Analysis of Real-Time Embedded Software*, Kluwer Academic Publishers, 1999.

[26] C. Ferdinand and R. Wilhelm, "Efficient and precise cache behavior prediction for real-time systems," *J. Real-Time Systems*, Kluwer Academic Publishers, Vol. 17, No. 2–3, 1999.

[27] A. Dasdan, *Timing Analysis of Embedded Real-Time Systems*, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1999.

[28] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *IEEE Trans. Computers*, Vol. 47, No. 6, 1998.

[29] H. Tomiyama and N. Dutt, "Program path analysis to bound cache-related preemption delay in preemptive real-time systems," In *Proc. of 8th CODES*, 2000.

[30] A. Datta, S. Choudhury, A. Basu, H. Tomiyama, and N. Dutt, "Task layout generation to minimize cache miss penalty for preemptive real time tasks: An ILP approach," In *Proc. of 9th SASIMI*, 2000.

[31] P. Grun, F. Balasa, and N. Dutt, "Memory size estimation for multimedia applications," In *Proc. of CODES/CASHE*, 1998.

[32] E. De Greef, F. Catthoor, and H. De Man, "Memory size reduction through storage order optimization for embedded parallel multimedia applications," In *Proc. of Workshop on Par. Proc. and Multimedia, Int'l. Par. Proc. Symp (IPPS)*, 1997.

[33] R. van de Wiel, *Code Compaction Bibliography*, http://www.win.tue.nl/rikvdw/bibl.html, 2000.

[34] I. Karkowski and H. Corporaal, "Exploiting fine- and coarse-grain parallelism in embedded programs," In *Proc. of PACT*, 1998.

[35] K. Danckaert, K. Masselos, F. Catthoor, and H. D. Man, "Strategy for power efficient combined task and data parallelism exploration illustrated on a QSDPCM video codec," *Journal of Systems Architecture*, Vol. 45, No. 10, 1999.

[36] V. Tiwari, S. Malik, A. Wolfe, and T.C. Lee, "Instruction level power analysis and optimization of software," *J. VLSI Signal Processing Systems*, Vol. 13, No. 2, August 1996.

[37] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*, Kluwer Academic Publishers, 1998.

[38] W. Shiue and C. Chakrabarti, "Memory exploration for low power embedded systems," In *Proc. of 36th DAC*, 1999.

[39] C. Kulkarni, F. Catthoor, and H. De Man, "Advanced data layout organization for multi-media applications," In *Proc. of IPDPS Workshop on Parallel, Distributed Computing in Image Processing, Video Processing and Multimedia*, 2000.

[40] H. Tomiyama, T. Ishihara, A. Inoue, and H. Yasuura, "Instruction scheduling for power reduction in processor-based system design," In *Proc. of DATE*, 1998.

[41] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory access scheduling," In *Proc. of 27th ISCA*, 2000.

[42] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," In *Proc. of 27th ISCA*, 2000.

[43] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using SimplePower," In *Proc. of 27th ISCA*, 2000.

[44] M. Kandemir, N. Vijaykrishnan, M. Irwin, and W. Ye, "Influence of compiler optimizations on system power," In *Proc. of 37th DAC*, 2000.

[45] D. Marculescu, "Profile-driven code execution for low power dissipation," In *Proc. of ISLPED*, 2000.

[46] *The COPPER Project: Compiler-Controlled Continuous Power- Performance Management*, The Center for Embedded Computer Systems, University of California, Irvine, http://www.cecs.uci.edu/~copper.