Application-Driven Synthesis of Memory-Intensive Systems-on-Chip

Darko Kirovski, Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith

***Abstract*— Due to the increasing popularity of multimedia and communications applications, requirements for application-specific systems typically include design flexibility and data management ability. Since the development of such systems is a market-driven task, reducing the time to market and manufacturing cost, while still satisfying application performance requirements, is an important system synthesis requirement.**

We have developed a new approach for area optimization of core-based systems. The approach uses basic block relocation in order to reduce the number of cache misses and, thus, enable hardware savings during system synthesis. Given a processor model, a cache model, and a set of nonpreemptive tasks with timing constraints, the goal of the synthesis framework is to select a system configuration (processor, I-cache, and D-cache) of minimal area that satisfies the performance constraints. The system synthesis framework has two key components. The first component is a code optimization engine that relocates basic blocks within a given assembly program in order to reduce the number of cache misses. The second component is a search mechanism that leverages the improvements in code performance obtained by the first component to select the most area-efficient system configuration.

In order to bridge the gap between the profiling and modeling tools, we have constructed a new performance evaluation platform. It integrates the existing modeling, profiling, and simulation tools with the developed system-level synthesis tools. The effectiveness of the synthesis approach is demonstrated on a variety of modern real-life multimedia and communication applications.

I. Introduction

MODERN application-specific systems are characterized by a need for increased design flexibility and large volume data management. The application and technology trends mutually influence the development of programmable systems with on-chip memory hierarchies and reusable design components. As the complexity and integration level of reusable components increase, efficient synthesis of such systems becomes an increasingly important design issue. Resource allocation in such systems depends on the target application and its performance requirements. The goal for designers of systems-on-chip (SOC) is to minimize product development and manufacturing cost, while meeting the timing constraints. Integrating programmable and hardwired cores reduces product development time. Similarly, silicon area plays an important role in the manufacturing cost. Therefore, the synthesis of programmable core-based systems has to be focused toward the integration of area-minimized application-driven designs.

A typical programmable SOC consists of a processor core, instruction and data cache and a number of hardware accelerators and control blocks. Application-specific integrated circuit (ASIC) floorplans reveal that most of the area in modern SOC's is dedicated to the processor core and associated caches [25]. Depending on the design abstraction, system synthesis and, in particular, evaluation of code optimization techniques, can be performed in a number of ways. For example, one way is oriented toward development of the actual system or its prototype (actual hardware or simulation system). The approach that we are introducing focuses on the development of reasonable high-level system model and measurement of its performance ahead of any prototypes. While the obvious advantage of the first approach is accuracy, the important advantage of the second one is that enables system tuning at early development stages.

We have developed a novel hierarchical modular approach for synthesis of programmable core-based data-intensive SOC's. The approach focuses on reducing the area assigned to the processor core and the cache subsystem, while meeting the performance requirements of a particular set of applications. The core of the optimization approach is a new global least-constraining/most-constrained probabilistic heuristic used at compile time for static relocation of basic blocks. Basic blocks in the program are repositioned in such a way that the mapping of basic block addresses to cache lines results in increased cache hit rates. The heuristic exploits the information derived from a profile that measures the spatial and temporal correlation among basic blocks during execution. The procedure for basic block relocation is used as a system performance optimization engine in the resource allocation algorithm. The goal of the resource allocation technique is to select an area-efficient processor and cache configuration that satisfies the target performance requirements for a single and a set of nonpreemptive applications.

Trace-driven simulation is used to accurately evaluate performance of the cache subsystem and to obtain the application execution profile. The Stanford on-line Cache Design Tool (CDT [13]) is used to estimate cache access latency and area based on properties such as cache total size, line size, feature size, replacement policy, and set associativity. Data extrapolated from the available literature for commercial programmable cores is used as a simplified model to approximate processor performance with respect to its area. The perfor-

Manuscript received July 6, 1998; revised November 24, 1998. This paper was recommended by Associate Editor G. Borriello.

D. Kirovski, C. Lee, and M. Potkonjak are with the Computer Science Department, University of California, Los Angeles, CA 90095-1596 USA.

W. H. Mangione-Smith is with the Department of Electrical Engineering, University of California, Los Angeles, CA 90095 USA.

Publisher Item Identifier S 0278-0070(99)06620-8.

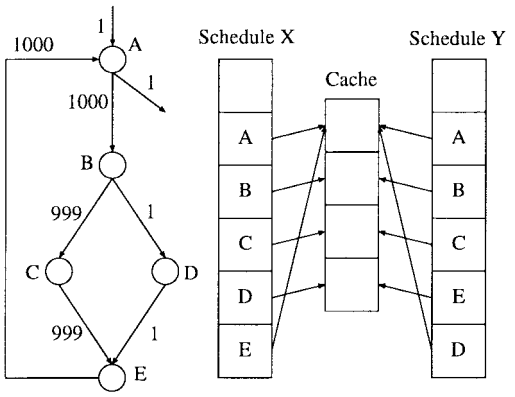

Fig. 1. Utilizing spatial correlation.

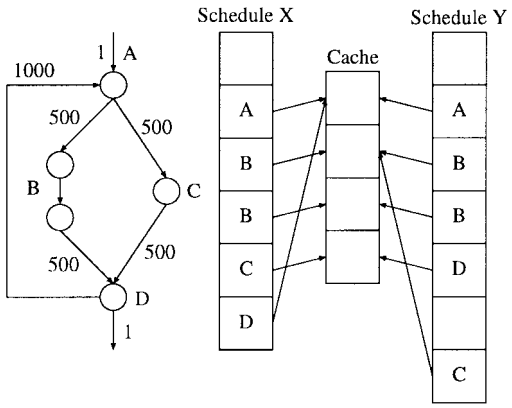

Fig. 2. Utilizing temporal correlation.

mance and area estimation models as well as the application profile drive the code optimization and resource allocation loop.

In order to bridge the gap between the profiling and modeling tools from the two traditionally independent synthesis domains [architecture and computer-aided design (CAD)], we have developed a new system performance evaluation platform. The platform integrates the existing modeling (Stanford CDT), profiling (SHADE [7]) and simulation (DINEROIII [16]) tools with the developed system-level synthesis tools. Application profiles are acquired through the use of SHADE augmented with our own trace analyzer. The result of the trace analysis is an application CFG augmented with information on spatial and temporal correlation of branch executions. We evaluate cache subsystem performance by running the application execution on the DINEROIII cache simulator. The effectiveness of the approach and the constructed simulation and evaluation platform is demonstrated on a variety of modern real-life applications [21].

Using the following motivational example, we briefly outline the key optimization insights that guide the static relocation of basic blocks. Fig. 1 illustrates a loop construct with an embedded conditional construct, where each node is a basic block and each arc indicates the number of times a certain path is taken. Each basic block is perfectly aligned to a cache line and is exactly as large as a line. Most existing compilers use a first-evaluated first-out algorithm to map basic blocks to memory (schedule X). If the profile of the program favors one branch of the IF-THEN-ELSE structure (e.g., C), then schedule X results in two conflict cache misses per iteration on a four-line direct mapped cache. Alternately, schedule Y results in no conflict misses. This schedule is developed by mapping the most frequently executed sequence of basic blocks to sequential locations in memory.

Temporal correlation is used to measure the average separation in time between activations of two basic blocks. Fig. 2 shows a loop construct with profile information. Accordingly, basic blocks B and C are executed equal number of times. However, the probability of executing either of these basic blocks in two consecutive iterations may be high. If all of the activations of B execute first, followed by all of the activations of C, then the two basic blocks can be mapped to the same cache line without impacting performance. However, such a mapping would reduce performance if the basic blocks are alternately activated. Standard compilers do not utilize this information and typically place the code as described in schedule X.

II. Related Work

We trace the related work along the following three lines: application-specific system optimization, processor and memory hierarchy design, and compilation strategies for embedded systems. SOC's are becoming an important focus area for both research and commercial developers. Due to market pressure, shortened design cycles have encouraged the use of predesigned processor cores. At the same time, market pressure to reduce system cost for consumer products has spurred the development of system-level synthesis techniques [5], [10], [28]. The developed system synthesis techniques have been focused on task scheduling and resource allocation algorithms [23] and interface design and verification [5]. Design-for-composability and ease of core synchronization has been addressed at the system and communication level [6].

As embedded applications have become more complex, hardware-software codesign has become increasingly important [10], [15]. Key optimization problems in hardware/software codesign have been identified as system component allocation, functional partitioning, quality metrics estimation, model refinement [6], [29], memory allocation using memory reuse for arrays and storage order [9], and control flow optimization for low power [18].

The increased interest in embedded system design with reusable core components has encouraged the development of high-level architecture and ASIC evaluation models. For example, *The Microprocessor Report* presents a monthly summary of the area and performance of numerous commercial processors [25]. The dominant impact of instruction and data cache size on system performance has been thoroughly studied [16], [17]. Memory hierarchy synthesis techniques for multimedia systems have been developed [22].

Recently, a number of compiler optimization strategies have been introduced for optimizing code generated for embedded systems [1]. Compiler techniques for reduction of cache misses in such studies have received significant attention in the research community. A dynamic address remapping for avoiding

TABLE I
A SAMPLE OF THE CACHE AREA AND LATENCY MODEL: A—AREA, L—LATENCY [13]

Cache		1KB	4KB	16KB	96KB
Direct Mapped	A (mm^2)	2.11	7.20	25.26	161
	L (ns)	3.79	4.16	4.76	6.79
2-way	A (mm^2)	2.38	7.28	25.36	156
	L (ns)	5.67	5.93	6.96	8.53
4-way	A (mm^2)	3.04	7.85	26.3	154
	L (ns)	6.04	6.34	7.51	8.83

conflict misses in large direct-mapped instruction caches has been proposed [3]. Static code repositioning by using cache line coloring at the procedure or basic block level has been an alternative approach proposed and evaluated in [12], [13], and [27]. Similar technique for profile-driven data repositioning has been proposed in [26].

III. PERFORMANCE MODELING

In this section, we describe the hardware performance models for caches and processor cores. Three factors combine to influence system performance: cache miss rates, processor performance, and system clock speed. The approach that we use here is to leverage existing models to estimate the area and performance of both caches and processor cores. This approach allows the synthesis system to be rapidly updated and applied to new technologies.

The Stanford on-line CDT is used to evaluate the impact of cache design choices on area and access latency. Caches typically found in current embedded systems range from 128B to 32KB. Since higher associativity can result in significantly higher access time, in this work we consider only direct mapped and two-way set associative caches. We used a fixed cache line size of 32 bytes. This decision aims to eliminate the cache penalty tradeoff problem. Large cache lines generally result in increased latency due to fetching data from main memory. Alternately, short cache lines increase access latency due to greater control hardware. We estimated the cache miss penalty based on the operating frequency of the system and external bus width and clock for each system investigated. This penalty ranges between four and 20 system clock cycles. Write-back is adopted in contrast to write-through, since it provides superior performance in uniprocessor systems though at increased hardware cost [17]. Each of the processors considered is constrained by the Flynn limit [13], and thus is able to issue at most a single instruction per clock period. As a consequence, the caches are designed to have a single access port. We used blocking I-caches since even for general purpose applications it has been shown that intelligent traditional cache structure may compensate for most of the benefits provided by nonblocking caches [23]. A sample of the cache model is presented in Table I. Cache access latency and area is computed for a number of organizations and sizes, all with implementation feature size fixed at 0.5 μm and typically six transistors per CMOS static-RAM (SRAM) cell.

We have built a polynomial processor area-performance model in the following way. We have collected performance data for a number of state-of-the-art microcontroller cores from the CPU Center Info web site[1] and *The Microprocessor Report* [25]. The list of initial processors is presented in Table II. The range of microprocessor core area varies from 2–8.4 mm^2, while the clock frequencies range within 24–266 MHz. Due to the inconsistencies in the technology parameters, we have scaled processor area and performance according to the supply voltage [4], clock frequencies, and feature size

TABLE II
A SAMPLE OF THE PROCESSOR PERFORMANCE VERSUS AREA MODEL [4], [25]

Microprocessor	Clock MHz	MIPS	Feature size μm	Area mm^2
StrongARM	233	266	0.35	4.3
ARM, 7	40	36	0.6	5.9
LSI Logic, TR4101	81	30	0.35	2
LSI Logic, CW4001	60	53	0.5	3.5
LSI Logic, CW4011	80	120	0.5	7
DSP Group, Oak	80	80	0.6	8.4
NEC, R4100	40	40	0.35	5.4
Motorola, 68000	33	16	0.5	4.4
PowerPC, 403	33	41	0.5	7.5

$$
\begin{aligned}
&AREA\_SCALED(nominal\_feature\_size)\\
&\quad = \alpha \cdot X + \beta \cdot X^2\\
&X = CPU\_AREA \cdot scaling\_area(feature\_size, nominal\\
&\qquad (feature\_size))\\
&MIPS(nominal\_voltage, nominal\_clock\_freq,\\
&\qquad nominal\_feature\_size)\\
&\quad = \gamma \cdot Y + \delta \cdot Y^2\\
&Y = CPU\_MIPS \cdot scaling\_mips(voltage, feature\_size,\\
&\qquad clock\_freq, nominal(voltage, feature\_size,\\
&\qquad clock\_freq)).
\end{aligned}
$$

The roadmap for the scaling process (functions *scaling_area* and *scaling_mips*) has been adopted from the state-of-the-art technology scaling reference [8]. All processors have been scaled to operate at the same operational voltage (3.3 V), have the same feature size (0.5 μm), and run at the same frequency (50 MHz). The scaling constants ($\alpha \cdots \delta$) in the binomials that define the given performance-area model have been bootstrapped such that the correlation between the model and the individual cores is maximized. This model has been statistically validated using the resubstitution statistical validation technology [11]. We have achieved that the maximal relative error in area-performance for any processor is within a 10% bound.

IV. THE NEW SYSTEM-LEVEL SYNTHESIS APPROACH

In this section, we describe the role of each module in the developed synthesis framework illustrated in Fig. 3. The core module in the synthesis flow is a global least-constraining/most-constrained heuristics that guides the algorithm for basic block relocation. Basic blocks are repositioned statically, at compile time, in a manner that reduces the incidence of conflict

[1] http://infopad.eecs.berkeley.edu/CIC/.

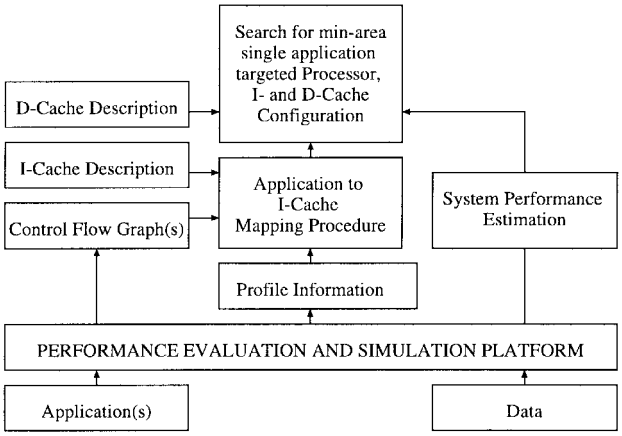

Fig. 3. The hierarchical modular synthesis approach.

misses. The temporal correlation of branch outcomes is used to improve cache performance by putting additional constraints on the basic block mapping. Code repositioning is accomplished with negligible increase in the static program memory size. The modification required to the program involves basic block motion, branch target updating, and occasionally branch insertion.

The application-driven search for a core and cache system with minimal area requires using trace-driven cache simulation for each promising point considered in the design space. In order to reduce the number of considered design configurations, we use a bounded divide-and-conquer search algorithm with pessimistic sharp bounds.

Our synthesis tools are augmented with the ability to produce a programmable SOC system that satisfies the performance requirements of multiple applications. This system requirement represents a realistic design expectation for most modern nonpreemptive multitask application-specific systems (e.g., audio/video processing, speech detection, error correction). The resource allocation technique considers each microprocessor core and selects the most area-efficient instruction and data cache design that satisfies the individual performance requirements of all applications. Finally, the configuration that has the smallest total processor and cache area is returned as the best solution.

System performance is evaluated using a platform that integrates existing simulation, modeling, and profiling tools. SHADE is a tracing tool that allows users to define custom trace analyzers and thus collect rich information on runtime events. The executable binary program is dynamically translated into host machine code. The tool also provides a stream of data to the translated code that is directly executed to simulate and trace the original application code. We have written a custom analyzer composed of approximately 2000 lines of C code. This analyzer is linked to SHADE to control and analyze the trace information. The analyzer reads relevant trace information from SHADE and builds a control flow graph (CFG) that corresponds to the dynamically executed code. The analysis consists of two passes. The first pass determines the boundaries of basic blocks, while the second pass constructs a CFG by adding control flow information between basic blocks. The custom analyzer also collects the frequencies of control transfers through each basic block, and calculates branch temporal correlation.

Once the CFG is obtained, an algorithm is employed to reposition application basic blocks in such a way that instruction cache misses are reduced. In our experiments, we use a basic block relocation look-up table to simulate the relocation of basic blocks in main memory. An entry in the basic block relocation table consists of two elements: the original and relocated starting address of the basic block. To simulate cache performance of a given application and data stream, we use a trace-driven cache simulator DINEROIII. System caches are described using a number of qualitative and quantitative parameters such as instruction and data cache size, replacement policy, set associativity, etc.

The system optimization process is composed of a sequence of activations of each of these tools. The SHADE analyzer traces program and data memory references as well as the CFG. The CFG is used to drive the code reposition module that produces a new application mapping table. Stream of references are sent to a program that uses the basic block relocation look up table to map from the original address space into the relocated address space. The remapped trace of addresses, along with all unmodified data memory references, is sent to DINEROIII for cache simulation. The final system performance is computed using the following formula: *CyclesPerInstruction* = (*System Clock Frequency/MIPS*) + *Cache Miss Ratio* · *Cache Miss Penalty* where *Cache Miss Ratio* is computed during the trace-driven simulation of the cache subsystem. *Cache Miss Penalty*, *System Clock Frequency*, and *MIPS* are system parameters introduced in Section III.

V. Synthesis Algorithms

This section discusses the problems encountered in synthesis of a single chip system and competitive optimization algorithms. First, the algorithm for basic block relocation is discussed. Second, based on the obtained code repositioning table, and therefore, improved cache performance, we assemble an area-efficient system configuration (core and cache structure) that satisfies the application-specific performance requirements. Finally, a case study is performed involving system optimization for a set of nonpreemptive applications.

The core of the proposed application-driven system-level synthesis technique involves basic block repositioning based upon profile information. Block repositioning aims for application execution on fixed hardware resources with minimal number of cache misses. Since basic blocks may be, in general, larger than a single cache line we partition such basic blocks into basic subblocks. The partitioning is performed in the following way. If a single basic block BB is smaller or equal to the size S of a cache line then BB is also a basic subblock (BS). Otherwise, BB is partitioned into consecutive partitions of size S where the last partition is equal or smaller in size to/than S. Each of these partitions is denoted as one BS. The basic (sub)block relocation phase is fed a CFG of the application. The profiling information for each branch (edge in the CFG) encapsulates the following information.

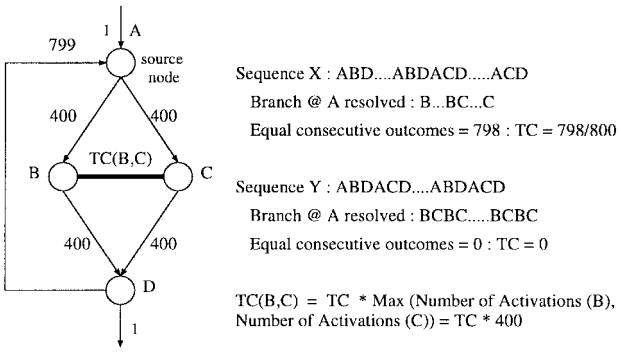

Fig. 4. Example of temporal correlation.

- An *intensity* $I_{i,j}$ of an edge connecting two nodes N_i and N_j is defined as the number of times control flowed from N_i to N_j.
- A *weighted temporal correlation* $TC_{i,j}$ is defined for each pair of disjoint-exclusive blocks N_i and N_j, i.e., blocks that cannot be sequentially executed when all feedback arcs in the CFG are removed (e.g., blocks B and C in the IF-THEN-ELSE structure in Fig. 4). In order to clearly define the weighted temporal correlation between two blocks we introduce the notion of a source node. The source node of two disjoint-exclusive blocks N_i and N_j is a node nearest to both N_i and N_j, from which both N_i and N_j can be reached. Weighted temporal correlation $TC_{i,j}$ between nodes N_i and N_j is defined as a product of the probability that the path from the source node of N_i and N_j toward node $N \in \{N_i, N_j\}$ is taken in two consecutive iterations and the number of times those paths are executed (two different cases for temporal correlation are presented in Fig. 4). For the sake of brevity, in the remainder of the paper, we mean weighted temporal correlation when referring to temporal correlation.

The result of the first phase in the optimization is a look-up table that maps each basic block to an address relative to the program starting address. The relative addresses are used by the compiler and linker when the executable binary is created. In order to develop an efficient methodology for finding an effective mapping function we analyze and transform the CFG and apply a global probabilistic least-constraining/most-constrained heuristic.

In the first phase, the correlation between basic blocks is expressed in the CFG by adding positive weight edges between nodes that are expected to be executed sequentially with high probability and negative weight edges between nodes that have high disjoint-exclusive temporal correlation. For example, assume that the profile of the IF-THEN-ELSE structure presented in Fig. 4 reveals that both basic blocks B and C are executed an equal number of times. While many possible execution patterns exist, two extremes are worth considering: blocks B and C are executed in alternation or first all of B's activations occur and then all of C's. While in the former case blocks B and C should be mapped onto different cache lines, in the latter case they could be mapped onto the same cache line. The CFG is transformed according to the pseudocode presented in Fig. 5. The goal of this transformation is to identify highly probable execution paths and to connect their source nodes with all basic blocks in the bodies of those paths. The process of connecting highly spatially correlated basic blocks is terminated when the following occur (see Fig. 6).

- The total size of basic blocks considered thus far is larger than the size of the I-cache.
- The probability that the traversed path is going to be taken is less than a predefined constant P. This condition restricts the search from following too many basic blocks that may result in a globally less probable path.

The weight of an edge drawn between any two nodes N_i and N_j is equal to the probability that the promoted path from N_i to N_j is going to be taken (*current_probability*) times the total number of times when the source branch is taken toward the destination node N_j. We augment the CFG with information about temporal correlation of disjoint-exclusively executed basic blocks by adding edges between them. The weight of those edges is negative and equal to their temporal correlation. As depicted in Fig. 7, the best candidates to be mapped to the same cache line where node F is mapped, are E or B (weight $= -100$).

Once the CFG is transformed, basic blocks are selected for mapping to particular addresses relative to the starting address of the program. The goal of the mapping function is to reduce the number of cache misses for the application, i.e., to map basic blocks that are likely to be executed sequentially into successive memory locations and to map basic blocks that are executed disjoint-exclusively with high temporal correlation onto cache-equivalent memory locations. The addressed optimization problem is stated in the following way.

PROBLEM: Basic block repositioning for optimum cache performance.

INSTANCE: Cache of size C, number of sets S, and cache line size L. Transformed CFG ($TCFG$). Each node N_i (in $TCFG$) has its size N_i^{size} represented in cache lines. Each edge E_i^j has its weight W_i^j.

QUESTION: Is there a partitioning of nodes such that each partition contains a fixed number of nodes, node of size M cache lines is spread over M consecutive partitions, and the sum of weights of edges connecting nodes in different partitions is greater than Q?

Since the problem of finding the optimum basic block mapping is computationally intractable [20], we opt to employ a heuristic search to obtain a good solution. Although greedy heuristics seem to have acceptable solution quality versus search run-time ratio, experiments show that least-constraining/most-constrained heuristics provide significantly better results both in terms of solution quality and run-time. The pseudocode of the heuristic is shown in Fig. 8.

The algorithm searches for a set of nodes that should be mapped to the same cache line. These nodes are selected in such a way that the probability of sequentially executing any pair of nodes from the set is low. The selection is guided by a simple cost function. This cost function is computed

Path promotion:

For each node BB with $\sum_{i=allNodes} E_{BB,i} < MIN$
  $current\_BB = BB$
  $current\_probability = \max(\frac{current\_BB.left.intensity}{current\_BB.previous.intensity}, \frac{current\_BB.right.intensity}{current\_BB.previous.intensity})$
  $page\_size = current\_BB.size$
  While ($current\_probability > P$ and $page\_size < CACHE\_SIZE$)
    $current\_BB = max\_intensity\_direction(current\_BB)$
    $connect\_nodes(BB, current\_BB)$:
      $edge(BB, current\_BB).intensity \mathrel{+}= current\_probability * max\_intensity\_direction(BB).intensity$
    $page\_size \mathrel{+}= current\_BB.size$
    $current\_probability = max(\frac{current\_BB.left.intensity}{current\_BB.previous.intensity}, \frac{current\_BB.right.intensity}{current\_BB.previous.intensity})$

Weight initialization:

For each edge E
  Weight(E) = $E.intensity$

Merging the information on temporal correlations:

For each node BB
  If $max\_intensity\_direction(BB).intensity < INTENSITY_{low}$ and $max\_intensity\_direction(BB).TC > TC_{low}$
    Connect sequentially executed nodes from both branches of the conditional
    with negative edge weight proportional to their temporal correlation (see Figure 7).

Note that function $max\_intensity\_direction(current\_BB)$
returns a pointer to the node with higher intensity only if that node
is not included in the so forth promoted path.

Fig. 5. Pseudocode for transformation of the CFG.

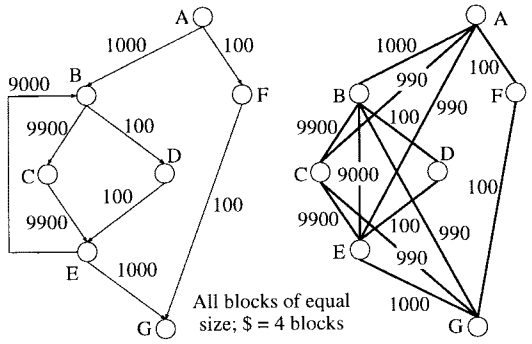

Fig. 6. Termination of path promotion. All blocks in the CFG are of equal size and the targeted cache size totals four blocks.

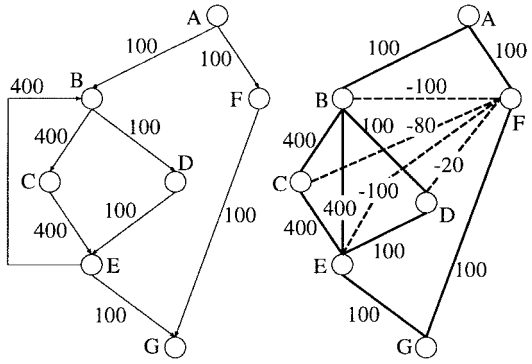

Fig. 7. An example of inserting temporal correlation into the CFG. Dotted weighted edges quantify the temporal correlation between two blocks.

locally, which is the key to efficiency. The efficiency of the algorithm depends solely on the cost function. Its first argument, $\Sigma_{i=NodesInSET,j=NodesNotInSET}\ W^2_{i,j}$, accepts only partitions which have large weights of edges coming out from the partition. The sum of edges which connect nodes within one partition, $\Sigma_{i=NodesInSET,j=NodesInSET}\ W^j_i$, forces creation of partitions which have few internal edges. This is an optimistic expectation which enables the heavyweight edges to stay in the partition selection process. Finally, since large basic blocks might cause fitting problems, we force their early selection by scaling the cost function with the total size of included basic blocks. Thus, the most constrained basic blocks are selected early and assigned to the least-constraining partitions.

In this phase of the synthesis approach a search is conducted for an area-efficient system configuration that satisfies the performance requirements of the target application. The search algorithm is described using the pseudocode shown in Fig. 9. For each microprocessor core type, we perform a search for the smallest overall cache structure which satisfies the performance requirements. The search starts by selecting a set of instruction cache sizes among which it is reasonable to expect that the best solution is found. For each of the sample instruction cache sizes, a search is performed in order to find the smallest data cache size which results in cache miss ratio smaller than some value defined by the timing constraints. Since the function that describes a cache miss ratio monotonically decreases with the increase of the cache size we employ binary search on the logarithmic scale of data cache sizes. The final solution is a configuration of a core, I-cache and D-cache which has the smallest overall area. The search is terminated every time the current solution is dominated by some identified solution. Dominated solutions are found in one of the following two ways.

1) *Fixed core A:* Best cache subsystem so far recorded for A totals Q bytes. Fixed I-cache equals P bytes. We do not evaluate D-caches larger than $Q - P$ bytes.
2) *Best core-cache configuration totals R square millimeters:* If we use core A of area X, we terminate the search whenever the sum of the I-cache and D-cache exceeds $R - X$ square millimeters.

Since trace-driven simulation has to be performed every time a particular cache system is evaluated, we opt to use an

For each cache entry E_i
  Randomly select a set SET of $\frac{Program.size \cdot L}{C}$ nodes
  Set best solution $SET^o = SET$
  Repeat $GLOBAL$ times
    Replace $STEP$ nodes of the current set SET with nodes in the rest of the graph
    If Cost(SET) < Cost(SET^o)
      $SET^o = SET$
      Repeat $LOCAL$ times
        $SET^+ = SET^o$
        Replace $STEP$ / 2 nodes of the current set SET^+ with nodes from the rest of the graph
        If Cost(SET^+) < Cost(SET^o) $SET^o = SET^+$
    If $GLOBAL$ mod $DIFF == 0$ $STEP = STEP - 1$
  Best solution is SET^o. Map all basic blocks in SET^o to
  cache entry E_i and propagate each basic block BB to the next $BB.size - 1$ cache lines

where Cost(SET) is computed using the following formula:

$$Cost(SET) = \frac{\sum_{i=NodesInSET, j=NodesNotInSET} W_{i,j}^2}{\sum_{i=NodesInSET, j=NodesInSET} W_{i,j}} \cdot \sum_{i=NodesInSET} N_i^{size}$$

This function is chosen among set of similar cost functions.
Its performance is statistically evaluated and validated.

Fig. 8. Pseudocode for the search for efficient basic block repositioning.

TABLE III
APPLICATIONS USED IN THE EXPERIMENTATION

Application	Source	Dynamic instructions
JPEG encoder	JPEG Group	3.9 million
JPEG decoder	JPEG Group	13.8 million
PGP encryption	P. Zimmermann	68.8 million
MPEG decoder	MPEG Software	1.1 billion
G.721	Sun Microsystems	62.9 million
GSM	TU Berlin	148.4 million

For each type of processor core
  For $I\_cache :: size = 128B..maxB, sets = 1..2$
    Selection: Perform Binary Search for D_cache of minimal size such that the configuration ($core, I\_cache, D\_cache$) satisfies the application performance requirements.
Return the overall configuration of minimal total area.

At any point during the algorithm, search is terminated along paths dominated by any other already pessimistically evaluated solution.

Fig. 9. Pseudocode for the resource allocation procedure.

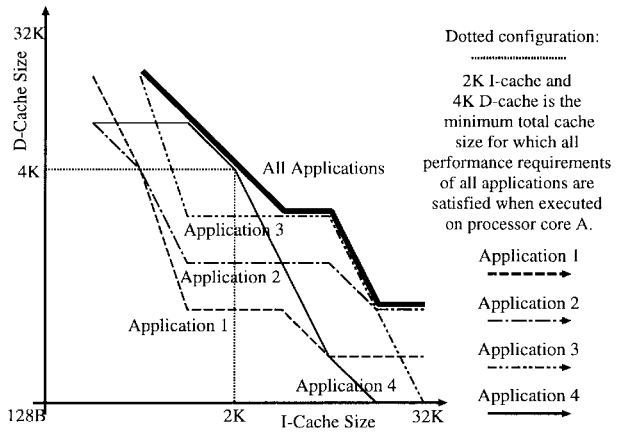

Fig. 10. Best instruction cache setups for a single core, four applications with fixed performance requirements, and a set of fixed data cache sizes.

approximation method in order to get faster but still accurate configuration performance measurements. In order to generate optimistic estimations we analyze the quality of application's CFG partitioning based on the profile information. The cost function used to evaluate the solution is

$$Cost(TCFG, SoP) = \frac{\alpha \cdot \sum_{i,j \in SoP, i \neq j} W_{a \in i, b \in j} - \beta \cdot \sum_{i \in SoP} W_{a \in i, b \in i}}{|i, j \in SoP, i \neq j|}.$$

This function is directly proportional to the sum of weights of edges that connect nodes between any two different cache line partitions in the set of all partitions SoP and sum of weights of edges that connect nodes in the same cache line partition. The cost function is normalized with the cardinality of the set of edges which connect nodes in different cache line partitions. Parameters α and β are statistically determined and validated using the meta-algorithmics methodology [19]. The core of the meta-algorithmics approach is the parameter evaluation procedure. This procedure performs multistart gradient search as the basic parameter optimization mechanism. The program for parameter evaluation first sets the parameters to random values. Then, it checks the optimized function Cost( ) when each parameter is positively and negatively offset by $k\%$ of its value. In our experiments, we used $k = 2$. The parameter that results in the smallest value Cost( ) is then offset to the value that caused this improvement, following the steepest descent paradigm. Next, the procedure exponentially increases or decreases the selected parameter using the golden ratio search approach. The golden ratio search is terminated once the improvement is smaller than a user specified value. Once the golden ratio search is completed, the list of the parameters is updated to this new value and the procedure is iteratively repeated as long as local minimum is not reached. The performance estimation function is used in our experiments in the following way. For a fixed I-cache size and a given microprocessor core we used the performance estimation technique to select the W most promising solutions ($W = 3$). Then, for those W solutions we have used trace-driven simulation to find the D-cache size that results in the least number of overall cache misses.

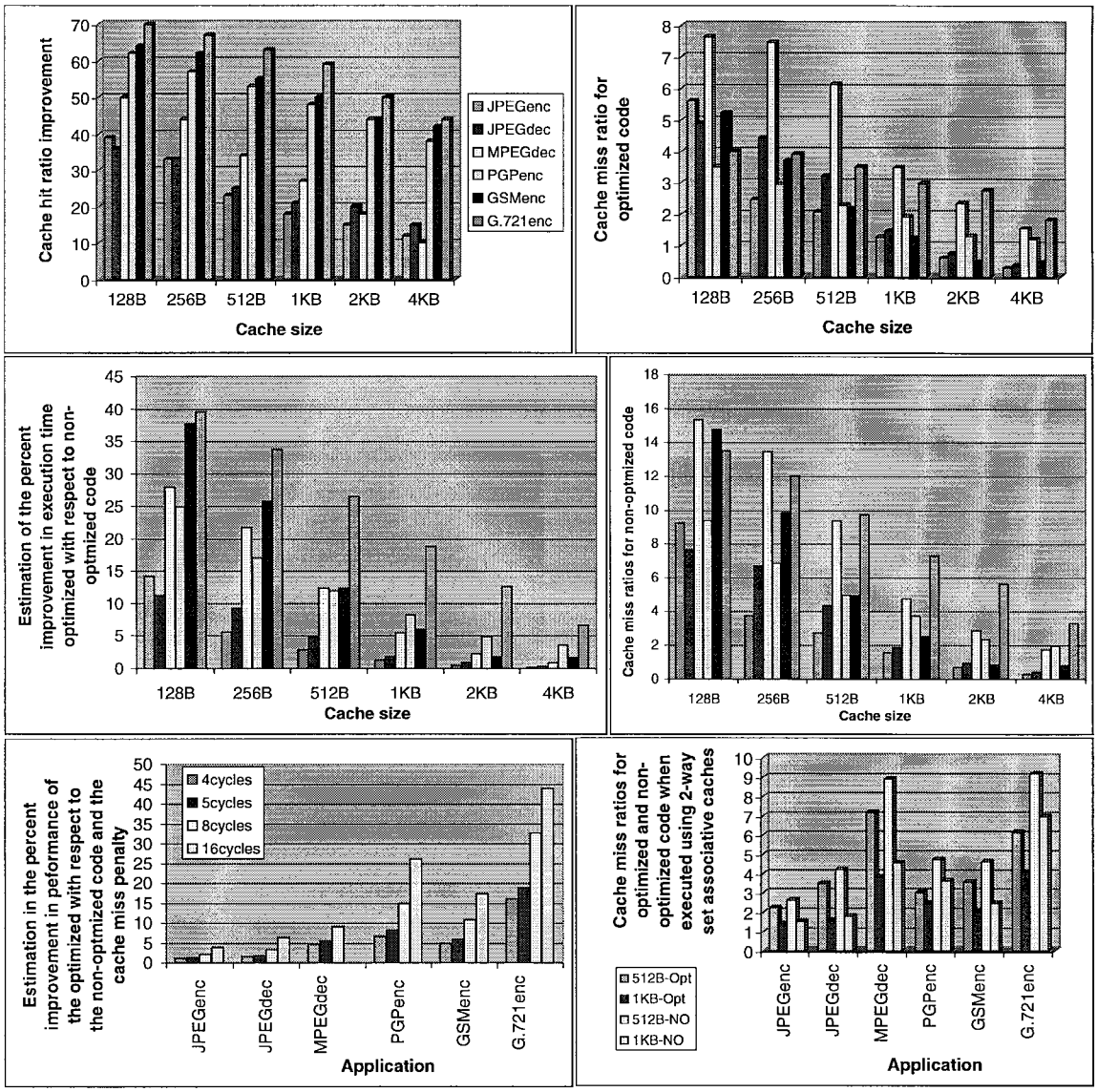

Fig. 11. Applications used in the experimentation.

Finally, the previous algorithm is augmented with a capability of finding the smallest core and cache configuration such that performance requirements of multiple nonpreemptive applications are satisfied. Such synthesis task corresponds to the real-life case since many embedded communications and computing systems are composed of a set of nonpreemptive real-time applications (e.g., speech and video encoding/decoding, encryption, etc.). In order to provide multiple application targeted resource allocation, the algorithm presented in Fig. 9 needs only be augmented with an improved selection step. The modification includes considering a conjunction of application requirements, rather than fulfilling a single performance deadline (see Fig. 10).

VI. Experimental Results

We have used six applications to demonstrate the effectiveness of the approach [21]. Table III gives a brief summary of the applications.[2] The source-code of the applications (written in C) was optimized for performance by the authors of the code. The code was compiled as Sun SPARC binaries using the GNU C compiler with optimization argument $-$O2. Therefore, the compiler tried to reduce code size and execution time. Optimizing the code with argument $-$O3 resulted in slightly worse cache hit ratios due to function inlining and other speed-size tradeoff optimizations.

The power of the global probabilistic least-constraining/most-constrained heuristic used in the graph partitioning algorithm for mapping an application to a given cache structure can be observed in Fig. 11. This figure contains six subcharts that quantify (from top left to bottom right): percentage improvement of the I-cache hit ratio when basic block relocation is used, the actual cache miss ratios when the code was optimized using basic block relocation, the estimation of the percent improvement in execution time of application when basic block relocation is used in a system with five cycles cache miss latency, the cache miss ratio with the code compiled using GNU C compiler (gcc), the estimation

[2] Details are available at http://www.cs.ucla.edu/~leec/mediabench.

TABLE IV
AREA-EFFICIENT SYSTEM CONFIGURATIONS FOR A GIVEN APPLICATION

Application	Basic blocks relocated	Processor Core	I-Cache	D-Cache	Area	Area improvement
JPEG encoding	NO	LSI Logic CW4001	2KB	1KB	9.4 mm^2	
	YES	LSI Logic CW4001	1KB	1KB	7.7mm^2	18%
JPEG decoding	NO	LSI Logic CW4011	1KB	2KB	12.9 mm^2	
	YES	LSI Logic CW4001	1KB	2KB	9.4 mm^2	27%
PGP encryption	NO	LSI Logic CW4001	1KB	1KB	7.7 mm^2	
	YES	LSI Logic TR4101	512B	1KB	5.0 mm^2	35%
MPEG decoding	NO	StrongARM	2KB	512B	13.8 mm^2	
	YES	StrongARM	1KB	512B	12.1 mm^2	12%
G.721 encoding	NO	LSI Logic CW4001	1KB	256B	6.6 mm^2	
	YES	LSI Logic TR4101	1KB	128B	4.3 mm^2	35%
GSM encoding	NO	LSI Logic CW4011	1KB	128B	7.0 mm^2	
	YES	LSI Logic CW4001	512B	128B	4.7 mm^2	33%

TABLE V
AREA-EFFICIENT SYSTEM CONFIGURATIONS FOR THREE SETS OF NONPREEMPTIVE APPLICATIONS

Application	Basic blocks relocated	Processor Core	I-Cache	D-Cache	Area	Area improvement
Application Mix 1.	NO	StrongARM	2KB	1KB	14.6 mm^2	
MPEG decoding	YES	StrongARM	1KB	1KB	12.9 mm^2	12%
JPEG decoding						
PGP encryption						
G.721 encoding						
Application Mix 2.	NO	LSI Logic CW4011	2KB	2KB	14.6 mm^2	
JPEG decoding	YES	LSI Logic CW4001	1KB	2KB	9.4 mm^2	35%
JPEG encoding						
PGP encryption						
GSM encoding						
Application Mix 3.	NO	StrongARM	2KB	2KB	16.3 mm^2	
JPEG encoding	YES	StrongARM	1KB	2KB	14.6 mm^2	10%
MPEG decoding						
PGP encryption						
G.721 encoding						

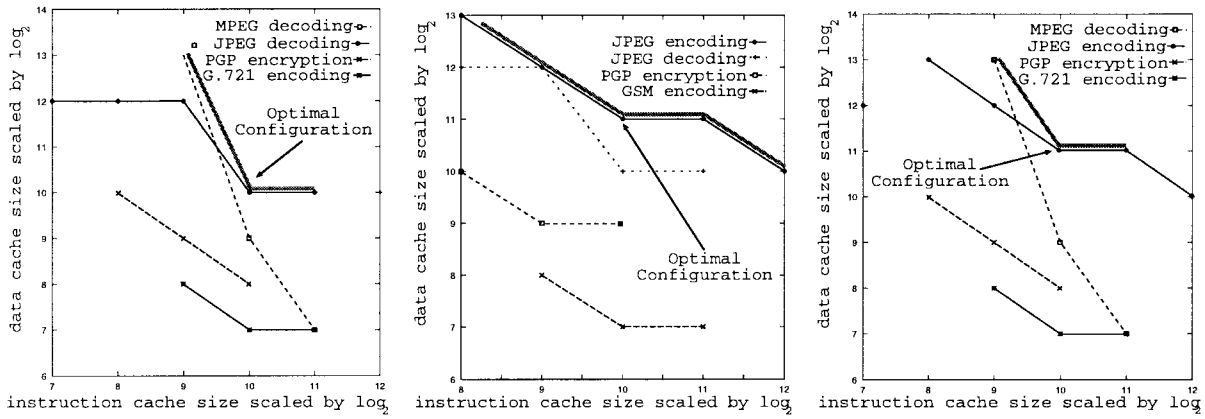

Fig. 12. Comparison of cache and system performance for relocated and nonoptimized code. Axis label for each chart specifies the performance property that the chart depicts.

of the percent improvement in execution time of application when basic block relocation is used in a system with different cache miss latencies, and the cache miss ratio for optimized and nonoptimized code when two-way set associative I-caches are used. In the first five subcharts, we present data only for direct mapped caches since they dominate set-associative caches in both latency and area when encountered cache sizes are small (see Table I). Moreover, code mapping improves system performance more significantly when direct-mapped rather than set-associative caches are used [26]. In our experiments, we did not notice a single improvement when using set-associative caches instead of equal-sized direct-mapped.

Since in smaller instruction caches conflict misses dominate other effects, basic block repositioning results in significant improvement in overall system performance. Depending on

the application, as shown in Table II, the performance speedup ranges from 40%–11.2% for 128B I-cache. The synthesis system considers this improvement and accordingly reduces the system's I-cache size while still achieving basic application performance requirements. For example, if the MPEG application timing constraint allows 10% I-cache miss ratio, then nonoptimized code results in 512B (miss ratio equal 9.37%) of I-cache while the optimized code requires only 128B (miss ratio equals 7.64%). On the other hand, code relocation provides negligible performance improvement when larger caches are used. This is mainly due to the decreased gap between the quality of the solution provided by the basic block relocation algorithm when the number of cache entries increases.

The developed application-driven synthesis approach resulted in area improvements for single application systems as presented in Table IV. The first two columns present the application and whether it was optimized using basic block relocation. The next three columns describe the allocated core and cache configuration followed by the total required area for the system. The last column specifies the area improvement of a system that has optimized code (lower subrow for each application) with respect to a system which runs the original gcc code (upper subrow for each application).

The size of the resulting cores supports the contention that defining an area-efficient cache subsystem for a particular application is a goal of major importance for the system designer. The results presented in Tables I, II, and IV show that a significant part of area of the synthesized ASIC is reserved for the cache subsystem. The diversity of selected cache structures implies that finding the appropriate area-efficient cache subsystem is not a trivial task. To describe further the necessity of considering application timing constraints in the optimization and synthesis of the entire system, and to give an overview on the resource allocation algorithm performance, we present a set of best candidate cache subsystems for the described benchmark suite.

We have assembled three application mixes and searched for area-efficient configurations that satisfy performance requirements of all applications in the mix. The configuration properties are presented in Table V and the equivalent diagrams generated during the search for the best cache system are shown in Fig. 12. The columns in Table V correspond to the columns in Table IV. The area-efficient cache configuration for each application mix is selected among cache subsystems with dominating performance (bold line in Fig. 12).

Since the employed search algorithm during the synthesis step is exhaustive, its main qualitative property is the effectiveness of the upper and lower bounds that terminate the search process. In our experiments, we have reduced the expected performance simulation times significantly (2–10 times depending on the starting I-cache values) while obtaining competitive search results.

The percentage of the total integrated circuits area dedicated to the cache subsystem, as well as strong diversity among selected area-efficient cache subsystems indicate the necessity of involving system-level synthesis approaches in order to reduce the product's total area while still satisfying product's quality of service.

VII. Conclusion

Trends in the application and semiconductor industry indicate that future application-specific single-chip systems will have much of the die area dedicated to the cache subsystem. We have developed a hierarchical approach for synthesis of programmable core-based devices. The synthesis approach is built around an efficient algorithm for mapping applications to I-caches. The algorithm is used in the search for an application-optimized area-efficient processor core, and instruction and data cache. The approach synthesizes an area-optimal processor and cache system for the most common system environment, a set of nonpreemptive target applications.

To support the evaluation of our synthesis paradigm we have developed a performance estimation platform that integrates existing modeling, profiling and simulation tools with the new system-level synthesis tool. Simulations have demonstrated that by optimizing code for cache effectiveness, even small caches (size 128B–256B) are sufficient to achieve 90+% instruction cache hit ratios for many multimedia and communications applications. The efficiency of the synthesis approach that facilitates this phenomenon has been tested on a set of multimedia and communication benchmarks.

References

[1] G. Araujo and S. Malik, "Optimal code generation for embedded memory nonhomogeneous register architectures," in *Proc. Int. Symp. System Synthesis*, 1995, pp. 36–41.

[2] N. Bellas, I. Hajj, and C. Polychronopoulos, "Architectural and compiler support for energy reduction in the memory hierarchy of high performance microprocessors," in *Proc. Int. Symp. Low-Power Electronics and Design*, 1998, pp. 70–75.

[3] B. N. Bershad, D. Lee, T. H. Romer, and J. B. Chen, "Avoiding conflict misses dynamically in large direct-mapped caches," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, vol. 29, no. 11, pp. 158–70, 1994.

[4] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, pp. 498–523, Apr. 1995.

[5] M. Chiodo, P. Giusto, A. Jurecska, H. C. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno, "Hardware-software codesign of embedded systems," *IEEE Micro.*, vol. 14, no. 4, pp. 26–36, 1994.

[6] P. Chou and G. Borriello, "Modal processes: Toward enhanced retargetability through control composition of distributed embedded systems," presented at "Design Automation Conference," 1998.

[7] B. Cmelik and D. Keppel, "Shade: A fast instruction-set simulator for execution profiling," in *Proc. SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, vol. 22, no. 1, pp. 128–137, 1994.

[8] B. Davari, R. H. Dennard, and G. G. Shahidi, "CMOS scaling for high performance and low power-the next ten years," *Proc. IEEE*, vol. 83, pp. 595–606, Apr. 1995.

[9] E. De Greef, F. Catthoor, and H. De Man, "Memory size reduction through storage order optimization for embedded parallel multimedia applications," *Parallel Computing*, vol. 23, no. 12, pp. 1811–1837, 1997.

[10] G. De Michelli and R. K. Gupta, "Hardware/software co-design," *Proc. IEEE*, vol. 85, pp. 349–365, Mar. 1997.

[11] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap.* New York: Chapman and Hall, 1993.

[12] S. McFarling, "Program optimization for instruction caches," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 1989, pp. 183–191.

[13] M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design.* Boston, MA: Jones and Bartlett, 1996.

[14] A. H. Hashemi, D. R. Kaeli, and B. Calder, "Efficient procedure mapping using cache line coloring," *SIGPLAN Notices*, vol. 32, no. 5, pp. 171–182, 1997.

[15] J. Henkel and R. Ernst, "A path-based technique for estimating hardware runtime in HW/SW-cosynthesis," in *Proc. Int. Symp. System Synthesis*, 1995, pp. 116–121.

[16] M. D. Hill and A. J. Smith, "Evaluating associativity in CPU caches," *Trans. Comput.*, 1989, pp. 1612–1630.
[17] N. P. Jouppi, "Cache write policies and performance," in *Proc. Int. Symp. Computer Architecture*, vol. 21, 1993, no. 2, pp. 191–201.
[18] K. S. Khouri, G. Lakshminarayana, and N. K. Jha, "IMPACT: A high-level synthesis system for low power control-flow intensive circuits," in *Proc. Design, Automation and Test Conf.*, 1998, pp. 848–854.
[19] D. Kirovski and M. Potkonjak, "System-level synthesis of low-power hard real time systems," in *Proc. Design Automation Conf.*, 1997, pp. 697–702.
[20] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Application-driven synthesis of core-based systems," in *Proc. Int. Conf. Computer-Aided Design*, 1997, pp. 104–107.
[21] C. Lee, D. Kirovski, I. Hong, and M. Potkonjak, "DSP quant: Design, validation, and applications of DSP hard real-time benchmark," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 1, pp. 679–682, 1997.
[22] Y.-T. S. Li, S. Malik, and A. Wolfe, "Cache modeling for real-time software: Beyond direct mapped instruction caches," in *Proc. Real-Time Systems Symp.*, 1996, pp. 254–263.
[23] Y. Li and W. Wolf, "A task-level hierarchical memory model for system synthesis of multiprocessors," in *Proc. Design Automation Conf.*, 1997, pp. 153–156.
[24] D. Lioupis and S. Milios, "The effects of cache architecture on the performance of operating systems in multithreaded processors," in *Proc. Euromicro Workshop Real-Time Systems*, 1997, pp. 72–79.
[25] *Microprocessor Report*, all issues, 1997.
[26] P. Panda, N. Dutt, and A. Nicolau, "Memory organization for improved data cache performance in embedded processors," in *Proc. Int. Symp. System Synthesis*, 1996, pp. 90–95.
[27] K. Pettis and R. C. Hansen, "Profile guided code positioning," in *Proc. Conf. Programming Language Design and Implementation*, vol. 25, 1990, no. 6, pp. 16–27.
[28] M. Potkonjak and W. H. Wolf, "Cost optimization in ASIC implementation of periodic hard-real time systems using behavioral synthesis techniques," in *Proc. Int. Conf. Computer-Aided Design*, 1995, pp. 446–451.
[29] J. Wilberg and R. Camposano, "VLIW processor codesign for video processing," *Design Automat. Embedded Syst.*, vol. 2, no. 1, pp. 79–119, 1997.

Darko Kirovski received the M.Sc. degree from the Computer Science Department at the University of California, Los Angeles, in 1997. Currently, he is pursuing the Ph.D. degree at the same university.

He is collaborating with Microsoft Research, Redmond, WA, and Conexant Systems, Newport Beach, CA. His interests include various aspects of design and debugging of systems-on-chip: combined simulation and emulation, engineering change, intellectual property protection, and content protection systems.

Mr. Kirovski has been awarded the Microsoft Research Graduate Fellowship and the DAC Graduate Scholarship.

Chunho Lee received the B.S. degree in Statistics from Korea University, Korea, in 1990. He is currently working toward the Ph.D. degree in computer science at University of California, Los Angeles.

His research interests include workload characterization and benchmark design; and system-level synthesis based on multiple-instruction-issue machines and their instruction-level parallelism compilers.

Miodrag Potkonjak received the Ph.D. degree in electrical engineering and computer science from University of California, Berkeley, in 1991.

He is an Associate Professor in the Computer Science Department at the University of California, Los Angeles (UCLA). In 1991, he joined C&C Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been with Computer Science Department at UCLA. He has published more than 150 papers in leading CAD, real-time, and signal processing, journals and conferences. His research interests include system design, embedded systems, computational security, and intellectual property protection.

Dr. Potkonjak received the TRW/School of Engineering and Applied Science at UCLA Excellence in Teaching Award in 1998.

William H. Magione-Smith attended the University of Michigan, Ann Arbor, where he received the B.S.E. degree in electrical engineering in 1987 and an M.S.E. and Ph.D. degrees in computer science and engineering in 1992.

From 1991–1995, he was with by Motorola Inc., where he participated in the design of the Envoy Personal Digital Assistant. Since 1995, he has been an Assistant Professor in the Electrical Engineering Department at the University of California, Los Angeles. His research interests include using dynamic circuits to implement configurable computing systems, low-power processor and system design, multimedia and communications processing, and all techniques for leveraging instruction-level parallelism.