Application-Driven Synthesis of Core-Based Systems

Abstract

The trend of integrating systems on chip using programmable and application specific cores has become predominant in the development of modern application specific systems. Due to the increasing popularity of multimedia and wireless communications applications, requirements for application specific systems typically include high design flexibility and data management ability. Since the development of such systems is highly market-driven, reducing the time to market and manufacturing cost, while still satisfying application performance requirements, is the most important system synthesis task.

We developed a new hierarchical modular approach for synthesis of area-minimal core-based data-intensive systems. The highlight of the optimization approach is a novel global least-constraining most-constrained heuristic used to guide the algorithm for minimization of instruction cache misses for a given application, instruction cache size and organization. Based on this performance optimization technique, we developed a strategy to search for a minimal area processor core, and an instruction and data cache which satisfy the performance characteristics of a single or a set of non-preemptive target applications.

In order to bridge the gap between the profiling and modeling tools from the two traditionally independent synthesis domains (architecture and CAD), we develop a new synthesis and evaluation platform. The platform integrates the existing modeling, profiling, and simulation tools with the developed system-level synthesis tools. The effectiveness of the approach is demonstrated on a variety of modern real-life multimedia and communication applications.

1 Introduction

1.1 Motivation

The ever increasing popularity of anywhere, anyform, anytime communications has established wide set of applications such as Internet browsing, video-conferencing, wireless communications, as target for most developers of consumer electronics. Systems emerging from such market demands and trends are characterized by a need for exceptionally high design flexibility and large volume data management requirements. On the other hand, the progress in the semiconductor technology has allowed high level integration of processors and memory structures on a single dice. The application and technology trends mutually influence the development of programmable systems with on-chip memory hierarchies and reusable design components. Since the complexity and integration level of reusable components rapidly increase, optimal synthesis of such systems becomes a major design issue.

Resource allocation in such systems depends solely on the target application and its performance requirements. The goal for designers of systems on silicon is to minimize product development and manufacturing cost, while still providing system's timing constraints. Integrating programmable and hardwired cores significantly reduces product's time-to-market. Similarly, product's die area plays a major role in the manufacturing cost. Obviously, core-based system synthesis has to be streamlined towards application-driven area-minimized integrated designs.

1.2 The New Synthesis Approach

A typical core-based system consists of a processor core, instruction and data cache cores and a number of hardware accelerators and control blocks as depicted in Figure 1. Numerous ASIC manuals and documents outline the fact that most of the IC area is dedicated to the processor core and associated caches. Therefore, the developed novel hierarchical fully modular approach for synthesis of core-based data-intensive ASICs focuses on minimizing the area assigned to the processor core and the cache subsystem with respect to the performance requirements of a particular set of applications.

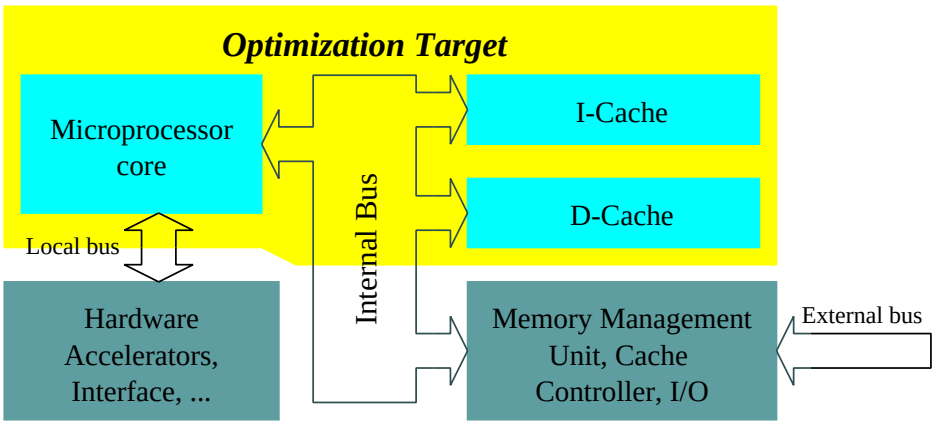


Figure 1: A typical core-based single-chip application specific system.

The highlight of the optimization approach is a new global least-constraining most-constrained probabilistic heuristic used in an attractive compile-time strategy for basic block static relocation.

Basic blocks in the program are repositioned in such a way that the mapping of basic block addresses to cache lines results in high cache hit rates. The employed heuristic exploits the information derived from a novel execution profiling approach which measures the spatial and temporal correlation among basic blocks. The procedure for basic block relocation is used as a system performance optimization engine in the synthesis algorithm. The goal of the synthesis approach is to select an area minimal processor core, and cache configuration that satisfy the target application performance requirements. Finally, an optimization approach is developed for synthesis of application specific systems targeted for a set of non-preemptive tasks.

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 [Fly96]) is used to estimate cache access latency and area based on properties such as cache total size, line size, semiconductor feature size, replacement policy, and associativity. Data extrapolated from the available literature for commercial programmable cores is used as a simplified model to approximate processor core performance with respect to its area. The performance and area estimation models as well as the application profile drive the optimization loop which attempts to minimize the total system area while still meeting the hard system performance requirements.

In order to bridge the gap between the profiling and modeling tools from the two traditionally independent synthesis domains (architecture and CAD), we developed a new synthesis and evaluation platform. The platform integrates the existing modeling (Stanford CDT), profiling (SHADE [Cme94]) and simulation (DINEROIII [Hill89]) 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 control flow graph 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, such as JPEG and MPEG codecs, GSM speech encoders and public key cryptographic protocols.

1.3 Motivational Example

For the sake of brevity and simplicity, we briefly outline only the key optimization insights which guide static basic block relocation.

Figure 2 illustrates a loop construct with an embedded IF-THEN-ELSE construct, where each node is a basic block and each arc indicates the number of times a certain path was taken. For the sake of simplicity, assume that each basic block is aligned on 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 4-line direct mapped cache. On the other hand, schedule Y results in no conflict misses. This schedule was developed by mapping the most frequently executed sequence of basic blocks to sequential locations in memory.

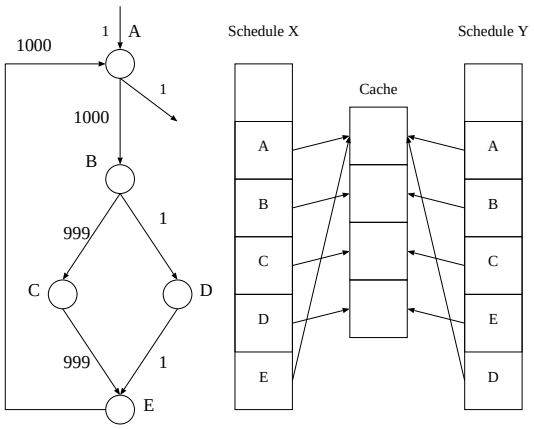


Figure 2: Utilizing spatial correlation.

Temporal correlation is used to measure the average separation in time between activations of two basic blocks. Consider Figure 3, which shows a CFG with profile information and reveals that 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, and then 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 alternated activations. Standard compilers do not utilize this information and typically place the code as described in schedule X. This new idea of exploiting the temporal correlation of basic block activations is developed further in the text.

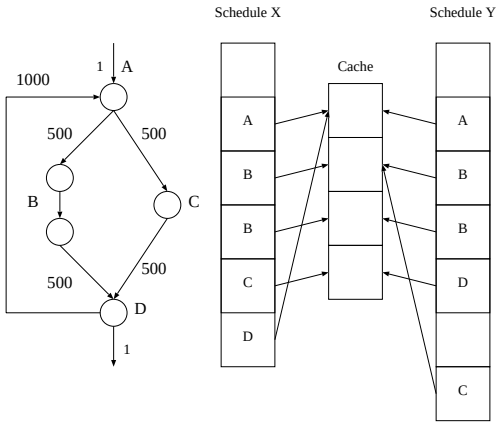


Figure 3: Utilizing temporal correlation.

1.4 Paper Organization

The rest of the paper is organized in the following way. Section 2 presents the related work in the field of system level synthesis, embedded processor and cache modeling, and cache performance optimization strategies. Section 3 explains the used microprocessor and cache models as well as the profiling methodology. The design flow of the novel synthesis approach is presented in Section 4. In Section 5 the optimization problems are recognized and their computational complexity is established.

In the same section the synthesis algorithms are proposed and explained in detail. Section 6 presents experimental data and discussion to evaluate the effectiveness of the approach. The paper is summarized in Section 7.

2 Previous Work

The related work can be traced through three areas: application-specific system optimization, processor and memory hierarchy design and evaluation, and compilation strategies for embedded systems.

Systems on silicon are becoming an important focus area for both research and commercial developers [Bol94]. Shortened design cycles, due to market pressure, 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 [Wol94], [Pot95], [Chi96], [Lin96], [Sub96]. As embedded applications have become more sophisticated and commercially relevant, hardware-software codesign and techniques for system level synthesis have also become increasingly important [Gup93], [Gaj94], [Ism94], [Hen95], [Lee96], [Chu96].

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 [MPR96]. Application driven microprocessor design and its performance issues have been studied in [Fis96]. Instruction and data caches, as the highest level of the memory hierarchy, have been thoroughly studied [Smi83], [Hill88], [Hill89], [Jou93]. Various memory hierarchy synthesis techniques for multimedia systems have been developed [Fu96], [Li96].

Recently, a number of compiler optimization strategies have been developed for optimizing the code generated for embedded systems [AM95], [Lie96]. Compiler directed techniques for minimizing number of cache misses in such studies has received significant attention in the research community. Bershad et al. have proposed dynamic address remapping for avoiding conflict misses in large direct-mapped instruction caches [Ber94]. Static code repositioning by using cache line coloring at the procedure or basic block level has been an alternative approach proposed and evaluated in [Hwu89], [Far89], [Pet90], [Tor96], [Tom96]. Similar technique for profile driven data repositioning has been proposed in [Pan96].

The work presented here makes two specific contributions. First, a new algorithm is presented for coloring cache lines in order to improve hit rates. This algorithm produces a significant performance improvement to the unoptimized program mappings and also exceeds the performance of previous algorithms. Second, we have developed a new unified framework for synthesis and optimization that ties together some of the existing methods for system modeling and optimization. To the best of our knowledge, this is the first synthesis to use software code mapping in order to enable hardware savings.

3 Preliminary Discussion

In this section we describe the hardware performance models for caches and processor cores, and present an outline of the profiling methodology. 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 approach to be rapidly updated and applied to new environments with new technology.

The Stanford on-line Cache Design Tool is used to evaluate the impact of cache design choices on area and access time. 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 2-way set associative caches. We used a fixed cache line size to 32 bytes. This decision attempts to eliminate the well known cache penalty tradeoff problem. Large cache lines generally result in increased latency while fetching from main memory, while 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 4 and 20 system clock cycles. Write-back was adopted in oppose to write-through, since it is proven to provide superior performance in uniprocessor systems [Jou93] though at increased hardware cost. Each of the processors considered is constrained by the Flynn limit [Fly96], and thus is able to issue at most a single instruction per clock period. Thus, the caches were designed to have a single access port.

Cache	Property	1KB	2KB	4KB	8KB	96KB
Direct	Area (mm^2)	2.107036	3.807370	7.199157	13.962853	161.704666
Mapped	Latency (ns)	3.79491	3.97239	4.16078	4.50642	6.78866
2-way	Area (mm^2)	2.385426	4.021449	7.286730	13.799528	156.061787
	Latency (ns)	5.67454	5.74781	5.93936	6.22891	8.53658
4-way	Area (mm^2)	3.042479	4.645923	7.850276	14.245449	154.000343
	Latency (ns)	6.04178	6.23715	6.33964	6.57738	8.83248

Table 1: A sample of the cache area and latency model.

A sample overview of the estimated cache model is presented in Table 1. 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 typical six transistors per CMOS SRAM cell.

Microprocessor	Date	Clock (MHz)	MIPS	Feature size (μm)	Area (mm^2)
ARM 810	1995	75	80	0.5	29
Motorola 403GC	1995	33	41	0.5	30
ARM SA-110	1996	200	230	0.35	33
i486GXSF	1996	33	18	0.8	18
MIPS4650	1996	133	175	0.6	32
Hitachi SH7708 (III)	1996	100	100	0.6	34
LSI Logic CW4001	1996	60	53	0.5	3.5
LSI Logic TR4101	1997	81	30	0.35	2

Table 2: A sample of the processor performance vs. area model.

Microprocessor performance and area information was culled from datasheets as well as from the CPU Center Info web site [CPU]. A sample of the collected data is presented in Table 2. The range of microprocessor core area varies from 2 to 34 mm^2, while the clock frequencies range within 33-200MHz.

Profile information is collected using the SHADE program from executions on a SPARC processor

architecture. DINERO is used to simulate the cache performance for various configurations, applications and program mappings. The benchmark suite used in this paper is composed of typical applications for portable data-rich communications devices: multimedia codecs, encryption, and protocol processing.

4 The New System Level Synthesis Approach

Figure 4 illustrates the synthesis framework. In this section we describe the role of each module and how modules are combined into a tightly integrated optimizing synthesis system.

4.1 The Synthesis Flow

The core module in the synthesis flow is a global least-constraining most-constrained heuristics which guides the algorithm for basic block relocation. Basic blocks are repositioned statically, at compile time, in such a way that frequently sequentially executed basic blocks are mapped into different cache lines. Also, the temporal correlation of branch outcomes is used to further 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 branch insertion.

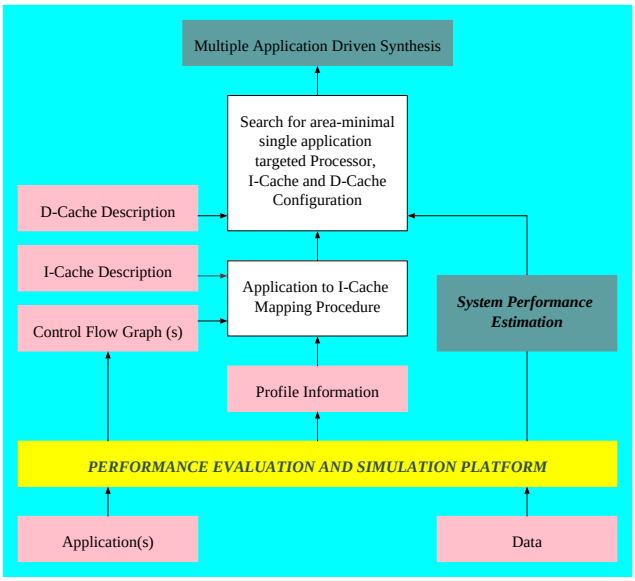


Figure 4: The hierarchical modular synthesis approach.

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. We attack these problems by carefully searching the space using bounded divide-and-conquer search algorithms with pessimistic sharp bounds and by providing powerfull performance estimation techniques.

The developed synthesis tools are augmented with the ability to synthesize a single-chip programmable ASIC system which satisfies the requirements of multiple applications. This system requirement represents a realistic design expectation for most modern non-preemptive multitask application specific systems (e.g. speech activity detection, speech compression, encoding). The synthesis technique considers each microprocessor core and selects the instruction and data cache design with minimal area

that satisfies the individual performance requirements of all applications. Finally, the configuration which has the minimum processor and cache total area is returned as the best solution.

4.2 Performance Evaluation Platform

System performance is evaluated using a platform which integrates simulation, modeling, and profiling tools as shown in Figure 5.

SHADE is a tracing tool which allows users to define custom trace analyzers and thus collect rich information on runtime events. SHADE currently profiles only SPARC binaries. The executable binary program is dynamically translated into host machine code. The tool also provides a stream of data to the translated code which is directly executed to simulate and trace the original application code. A custom analyzer composed of approximately 2,000 lines of C code, is linked to SHADE to control and analyze the generated trace information. The analyzer sources relevant trace information from SHADE and builds a control flow graph (CFG) corresponding 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.

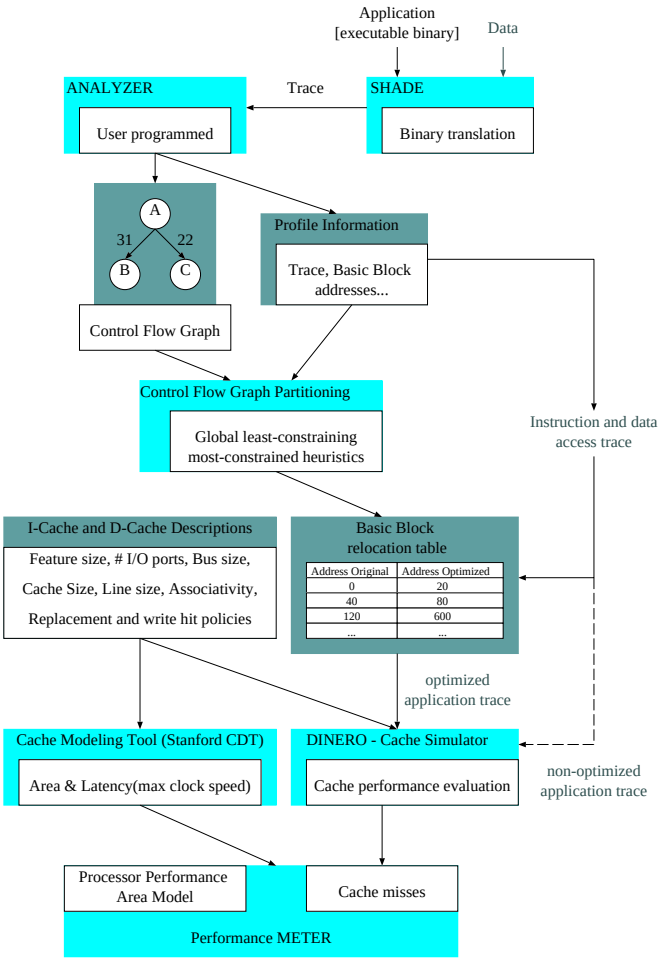


Figure 5: Synthesis flow of application-driven programmable core-based systems.

We also collect the frequencies of control transfers through each basic block, and branch temporal

correlation (discussed later). Table 3 shows a typical CFG generated by the analyzer. The first two columns show the basic block starting and ending address. Branch type (conditional[1], unconditional[2], procedure call[4], or indirect branch target jmpl[3]) and the number of possible target addresses appear in the next two columns. The target address, and the number of control transfers to it (activations) are given for both targets in next four columns. The temporal correlation of branch activations and size of the appropriate basic block (in bytes) are shown in the last two columns.

Once the CFG is obtained, an algorithm is employed to reposition application basic blocks in such a way that instruction cache misses are minimized. Our experimentation uses 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 optimized 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 cache is described using a number of qualitative and quantitative parameters such as instruction and data cache size, replacement policy, associativity, etc.

Branch				Targets					
Start	Stop	Type	Targets	T1 address	T1 taken	T2 address	T2 taken	TempCorr	Size
10074	1008c	[1]	2	10098	04532	101aa	00005	0.999	28
10098	100a4	[4]	2	100a8	04527	1032a	00007	0.999	16
100a8	100ac	[4]	2	100b0	04520	101fc	02220	0.824	8
100b0	100d4	[4]	2	100e8	02300	10fa4	00053	0.991	40
100e8	100f0	[4]	2	100f8	02247	10182	01235	0.923	12
100f8	10118	[1]	2	1012c	01012	10160	00324	0.183	36

Table 3: Analyzer output format.

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 which 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 optimized address space. The remapped trace of addresses, along with all unmodified data memory references, is sent to DINERO for cache simulation.

The final system performance was computed using the following formula:

$$CyclesPerInstruction = \frac{SystemClockFrequency}{MIPS} + CacheMissRatio \cdot CacheMissPenalty$$

where $CacheMissRatio$ was computed during the trace driven simulation of the cache subsystem. $CacheMissPenalty$, $SystemClockFrequency$, and $MIPS$ are system paprameters introduced in Section 3.

5 Synthesis Optimization Algorithms

The problems encountered in synthesis of a single chip system and competitive optimization algorithms are described in the following subsections. First, the algorithm for basic block relocation is discussed. Then, based on the obtained code repositioning table and therefore, improved cache performance, we assemble an area-efficient system configuration (core and cache structure) which satisfies

application specific performance requirements. Finally, a case study is performed involving optimizing a system for a set of non-preemptive applications.

5.1 An Efficient Method to Map an Application to an I-Cache

The heart 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. This phase is fed a control flow graph of the application. The profiling information for each branch (edge in the control flow graph) encapsulates the following information:

- 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 temporal correlation $TC_{i,j}$ is defined for each pair of disjoint-exclusive blocks N_i and N_j, i.e. blocks which cannot be sequentially executed when all feedback arcs in the control flow graph are removed (e.g. blocks B and C in the IF-THEN-ELSE structure in Figure 6). In order to clearly define the 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. 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 towards 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 Figure 6).

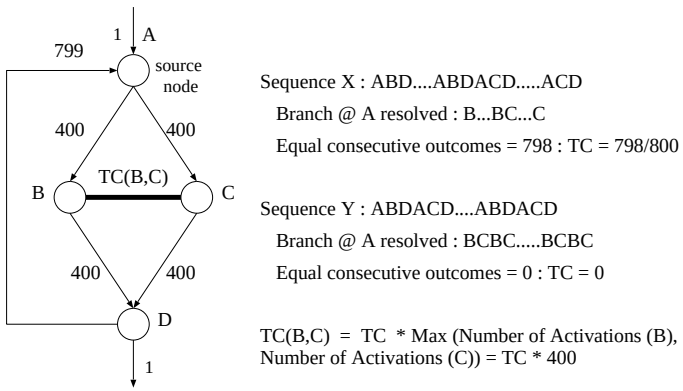


Figure 6: Example of temporal correlation.

This step produces a table which 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 novel global probabilistic least-constraining most-constrained heuristic.

In the first phase, the correlation between basic blocks is expressed in the control flow graph by adding positive weight edges between nodes which are expected to be executed sequentially with high

probability and negative weight edges between nodes which have high disjoint-exclusive temporal correlation. For example, assume that the profile of the IF-THEN-ELSE structure presented in Figure 6 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 to be mapped onto different cache lines, in the latter case they should be mapped onto the same cache line. The CFG is transformed according to the pseudo-code presented in Figure 7.

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 (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.

Figure 7: Control flow graph transformation pseudo-code.

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 (Figure 8):

- The total size of basic blocks considered thus far is larger than the size of the cache. This condition terminates the path search once the traversed blocks cannot be stored into a single cache page.
- 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 towards 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 Figure 9, the best candidates to be mapped to the same cache line where node F is mapped, are E or B (weight = -100).

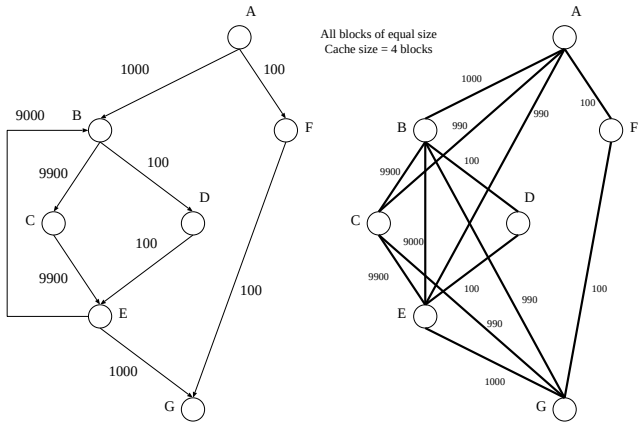


Figure 8: Path promotion example.

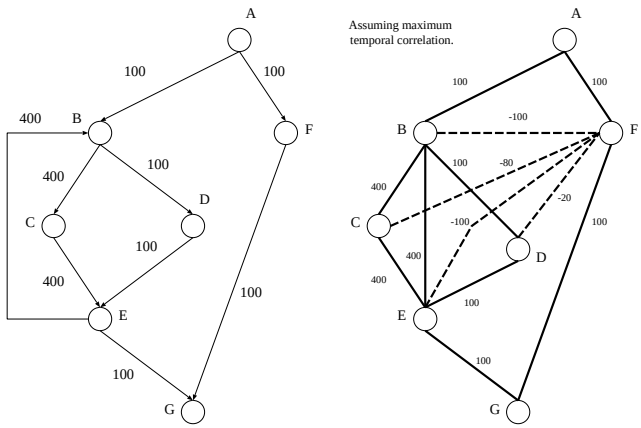


Figure 9: An example of inserting temporal correlation into the control flow graph.

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 minimize 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 which are executed disjoint-exclusively with high temporal correlation onto cache-equivalent memory locations.

The addressed optimization problem is stated using the standard Garey-Johnson format [Gar79]:

PROBLEM: Basic block repositioning for optimum cache performance.

INSTANCE: *Cache of size C, number of sets S, and cache line size L. Transformed control flow graph ($TCFG$). 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 each partition contains part of fixed number of nodes, node of size M cache lines is spread over M consecutive partitions, the sum of weights of edges connecting nodes in different partitions is greater than Q?*

THEOREM: Basic block repositioning for optimum cache performance is an NP-complete problem.

PROOF: Restrict to GRAPH PARTITIONING ([ND14], page 209, [Gar79]) by forcing all nodes to be of fixed size equal to one cache line.

Since the problem of finding the optimum basic block mapping is computationally intractable, we opt to employ a heuristic search algorithm to obtain an approximate solution. Although greedy heuristics seem to have acceptable solution quality vs. search run-time ratio, experimentations show that the global probabilistic most-constrained least-constraining heuristic provides significantly better results both in terms of the solution quality and run-time efficiency. The pseudo-code for this heuristic is shown in Figure 10.

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.

Figure 10: Search for the optimal basic block repositioning.

The algorithm searches for a set of nodes which 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 locally, which is the key to efficiency. The efficiency of the algorithm depends solely on the cost function. Its first argument,

$$\sum_{i=NodesInSET, j=NodesNotInSET} W_{i,j}^2,$$

accepts only partitions which have large weights of edges coming out from the partition. The sum of edges which connect nodes within one partition,

$$\sum_{i=NodesInSET, j=NodesInSET} W_i^j,$$

forces creation of partitions which have few internal edges. This is an optimistic expectation which enables the heavy-weight 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.

5.2 Search for Optimal Resource Allocation

In this phase of the synthesis approach a search is conducted for a minimal-area system configuration which satisfies the performance requirements of the target application. The search algorithm is described using the pseudo-code shown in Figure 11.

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 which are dominated by any other already pessimistically evaluated solution.

Figure 11: Pseudo code for the resource allocation procedure.

For each microprocessor core type we perform a search for the smallest overall cache structure which satisfies the application's 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 application's guaranteed 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 existing minimal solution. Dominated solutions are found in one of the following two ways:

- 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.
- 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 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:

$$Cost(TCFG, SetOfPartitions) = \frac{\alpha \cdot \sum_{i,j \in SetOfPartitions, i \neq j} W_{a \in i, b \in j} - \beta \cdot \sum_{i \in SetOfPartitions} W_{a \in i, b \in i}}{\sum_{i,j \in SetOfPartitions, i \neq j} 1}$$

This function is directly proportional to the sum of weights of edges that connect nodes between any two different cache line partitions 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 were experimentally determined and validated using the meta-algorithmics methodology [Kir97]. The resulting performance estimation offset was within 5% of the simulated performance measurements.

The performance estimation function was 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 used trace driven simulation to find the D-cache size which results in the minimal number of overall cache misses.

Finally, previous algorithm is augmented with a capability of finding the minimal area-optimal core and cache configuration such that performance requirements of multiple non-preemptive 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 non-preemptive 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 Figure 11 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 Figure 12).

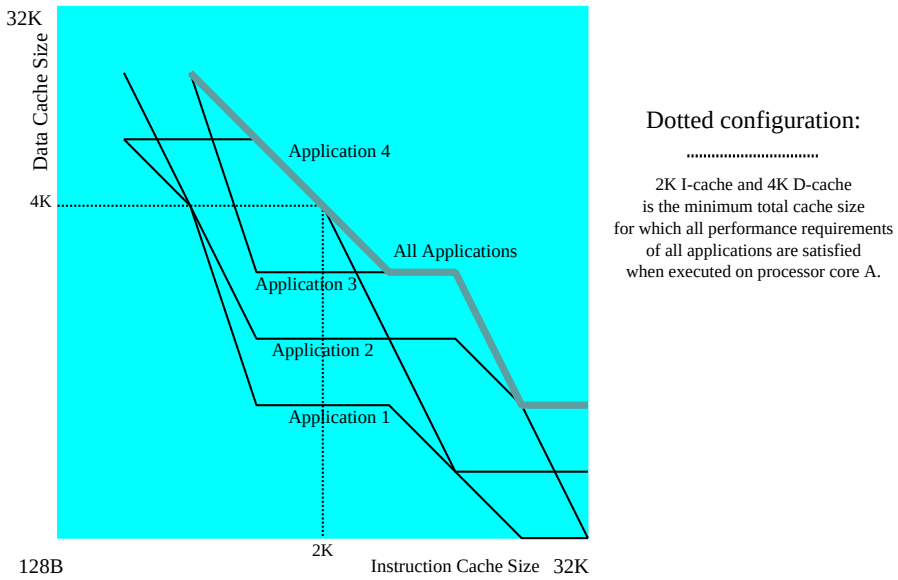


Figure 12: Best instruction cache setups for a single core, four applications with fixed performance requirements and a set of fixed data cache sizes.

6 Experimental Results

Presentation of our experimental results is organized in the following way. First, the set of applications and corresponding data used for the experimental evaluation of the approach is described in detail. The effectiveness of the graph partitioning algorithm is demonstrated. After resource allocation results are discussed, we summarize the main conclusion obtained from our experimental data.

We used six applications to demonstrate the effectiveness of the approach. All six applications are in public domain and can be obtained via the Internet. JPEG software from the Independent JPEG Group implements JPEG baseline, extended-sequential, and progressive compression processes. We used integer DCT for decoding a JPEG file to a PPM file. Integer DCT and progressive compression process options were used for compression. MPEG software from MPEG Software Simulation Group

decode MPEG-1 and MPEG-2 video bitstreams. We obtained and used an MPEG animation file for our experimentation. For simulation of PGP software we used an ASCII text file. The data file for GSM and G.721 encoding was obtained from Fine Arts Graphics Lab, Ball State University. The data file is in Sun u-law format. Table 4 gives a brief summary of the applications, while summary of data used is given in Table 5.

Application	Version	Source	Dynamic instructions
JPEG encoder	6a	Independent JPEG Group	3.9 million
JPEG decoder	6a	Independent JPEG Group	13.8 million
PGP encryption	2.6.2.	Philip Zimmermann	68.8 million
MPEG decoder	1.1b	MPEG Software Simulation Group	1.1 billion
G.721	N/A	Sun Microsystems, Inc.	62.9 million
GSM	06.10	Technische Universitaet Berlin	148.4 million

Table 4: A brief description of applications used in the experimentation.

Benchmark	File size	Format	Description
JPEG encoding	101,484 bytes	PPM	an uncompressed bit map
JPEG decoding	5,770 bytes	JPG	a JPEG compressed
PGP encryption	194,723 bytes	ASCII	a plain text
MPEG decoding	1,567,055 bytes	MPEG	14.98 s., 476 frames
GSM encoding	26,372 bytes	Sun .au u-law	one sentence (>2sec.)
G.721 encoding	26,372 bytes	Sun .au u-law	one sentence

Table 5: A brief description of data used in the experimentation.

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 from Table 6. The first column of the table specifies the application name. Then, in the next four consecutive columns speedup, improvement in cache miss ratio with respect to the non-optimized code, and cache miss ratios for both the optimized and non-optimized code are presented for a 128 bytes instruction cache. Speedup in the application execution is obtained by assuming a single instruction issue architecture and 5 clock cycles miss penalty. The remaining columns represent the same information for five different instruction cache sizes. 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 1). Moreover, code mapping improves system performance more significantly when direct-mapped rather than set-associative caches are used [Pan96]. In our experimentations 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, basic block repositioning results in significant overall system performance improvement. For example, as shown in Table 6, the performance speedup ranges depending on the application from 40% to 11.2% for 128 bytes instruction cache. System synthesis considers this improvement and accordingly lowers system's instruction cache size while still achieving basic application performance requirements. For example, if the MPEG application timing constraint allows 10% instruction cache miss ratio, then non-optimized code results in 512 bytes (miss

ratio equal 9.37%) of instruction cache while the optimized code requires only 128 bytes (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 optimized and non-optimized control flow graph partitioning algorithm when the number of entries (partitions) increases.

The developed application driven synthesis approach, based on the ability to optimize the cache performance and adjust its size to the particular application needs, resulted in area-minimal system configurations as presented in Table 7. The first column presents the application description. The next three columns describe the allocated core and cache configuration followed by the total required area for the system.

	128B				256B				512B			
Program	SU	Imp	Opt	NOpt	SU	Imp	Opt	NOpt	SU	Imp	Opt	NOpt
JPEGenc	14.23%	39%	5.6%	9.25%	5.59%	33%	2.46%	3.72%	2.91%	23%	2.07%	2.72%
JPEGdec	11.2%	36%	4.89%	7.68%	9.24%	33%	4.43%	6.69%	4.84%	25%	3.21%	4.34%
MPEGdec	27.92%	50%	7.64%	15.36%	21.74%	44%	7.46%	13.44%	12.42%	34%	6.12%	9.37%
PGPenc	24.9%	62%	3.5%	9.36%	17.06%	57%	2.94%	6.86%	11.97%	53%	2.28%	4.95%
GSMenc	37.75%	64%	5.23%	14.76%	25.85%	62%	3.71%	9.85%	12.39%	55%	2.18%	4.93%
G.721enc	39.57%	70%	4%	13.51%	33.79%	67%	3.92%	12.01%	26.48%	63%	3.51%	9.75%
	1KB				2KB				4KB			
JPEGenc	1.32%	18%	1.28%	1.57%	0.56%	15%	0.61%	0.73%	0.19%	12%	0.28%	0.32%
JPEGdec	1.84%	21%	1.46%	1.86%	0.93%	20%	0.74%	0.94%	0.33%	15%	0.36%	0.43%
MPEGdec	5.48%	27%	3.48%	4.77%	2.3%	18%	2.34%	2.86%	0.86%	10%	1.54%	1.73%
PGPenc	8.26%	48%	1.92%	3.74%	4.9%	44%	1.3%	2.35%	3.59%	38%	1.2%	1.97%
GSMenc	5.98%	50%	1.24%	2.52%	1.84%	44%	0.47%	0.85%	1.7%	42%	0.47%	0.82%
G.721enc	18.8%	59%	2.97%	7.3%	12.61%	50%	2.75%	5.63%	6.63%	44%	1.83%	3.28%

Table 6: Comparison of cache and system performance for relocated and non-optimized code. *SU* - Speedup of the execution of optimized code with respect to the non-optimized code. Five cycles miss penalty is assumed as well as a single instruction issue processor. *Imp* is the improvement in cache miss ratio after basic block relocation. *Opt* and *NOpt* are columns which represent cache miss ratios for optimized, and non-optimized code executions respectively.

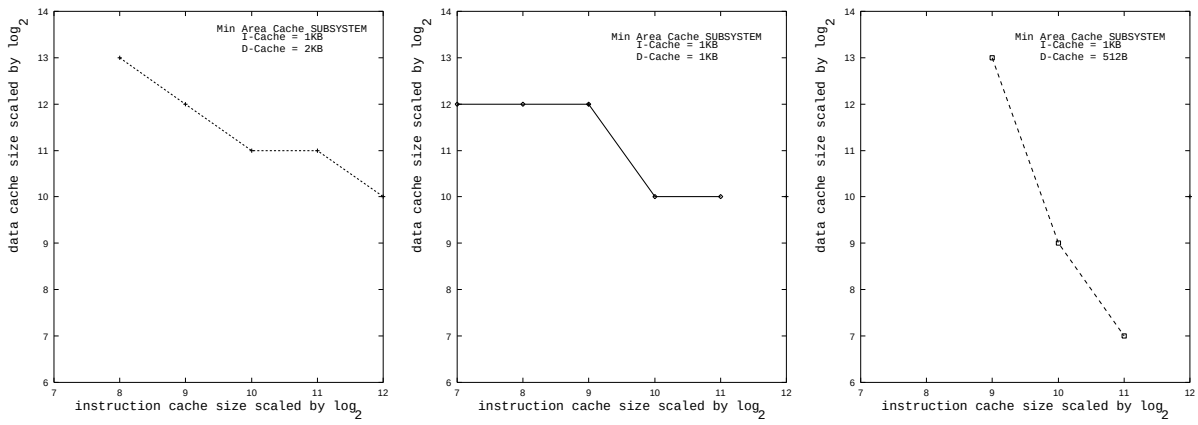
Application	Processor Core	I-Cache	D-Cache	Area	Cache Area vs. Total Area
JPEG encoding	LSI Logic CW4001	1KB	1KB	7.7 mm^2	43%
JPEG decoding	LSI Logic CW4001	1KB	2KB	9.4 mm^2	63%
PGP encryption	LSI Logic TR4101	512B	1KB	5.0 mm^2	60%
MPEG decoding	ARM SA-110	1KB	512B	36 mm^2	8%
G.721 encoding	LSI Logic TR4101	1KB	128B	4.3 mm^2	53%
GSM encoding	LSI Logic CW4001	512B	128B	4.7 mm^2	25%

Table 7: Area-minimal system configurations for a given applications.

The obtained results acknowledge the domination of LSI Logic cores due to their exceptional performance vs. area ratio. The size of those cores, on the other hand points out to the fact that defining an area minimal cache subsystem for a particular application is a goal of major importance for the system designer. The results presented in Tables 1, 2 and 7 show tha 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-minimal 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.



(a) JPEG decoder. LSI Logic CW4001. 8% overall cache miss ratio.

(b) JPEG encoding. LSI Logic CW4001. 4.2% overall cache miss ratio.

(c) MPEG decoding. ARM SA-110. 10% overall cache miss ratio.

(d) PGP encryption. LSI Logic TR4101. 8% overall cache miss ratio.

(e) GSM encoding. LSI Logic TR4101. 10% overall cache miss ratio.

(f) G.721 encoding. LSI Logic CW4001. 11% overall cache miss ratio.

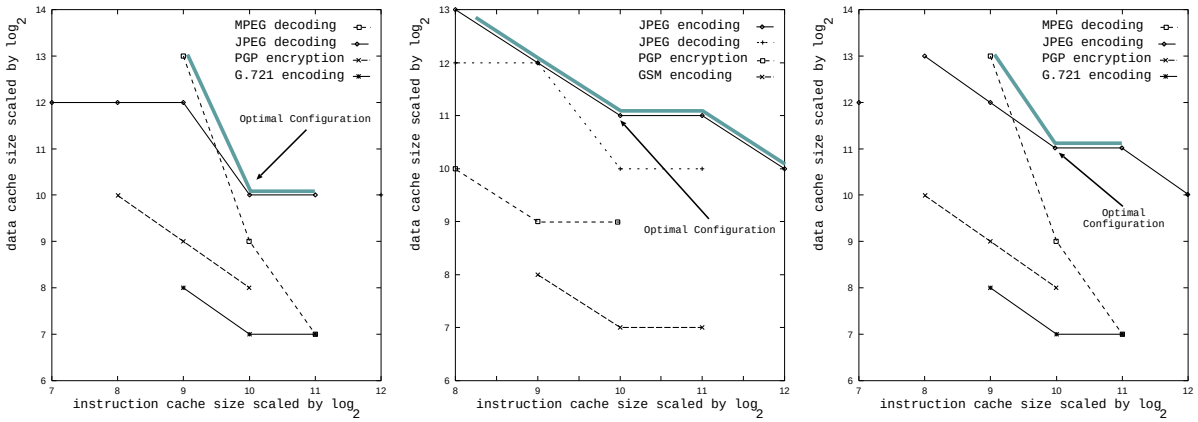
Figure 13: Trace driven simulation for area minimal cache subsystem determination for single application.

As described in Subsections 5.1 and 5.2, for each selected processor core, we performed trace-driven cache simulations in order to find area-optimal I-cache and D-cache configuration. Figure 13 shows

the simulation results for all candidate cache subsystems. Each diagram in Figure 13 presents minimal D-cache sizes which correspond to I-cache sizes which altogether build a cache system that satisfies the overall application requirement in cache misses for the selected processor core. For example, Table 13(a) presents five possible cache subsystem candidates for the JPEG decoder application. The application requirement was represented as 8% overall cache miss ratio for LSI Logic CW4001 core.

Application	Processor Core	I-Cache	D-Cache	Area	Cache Area vs. Total Area
Application Mix 1. MPEG decoding JPEG decoding PGP encryption G.721 encoding	ARM SA-110	1KB	1KB	37 mm^2	9%
Application Mix 2. JPEG decoding JPEG encoding PGP encryption GSM encoding	LSI Logic CW4001	1KB	2KB	9.4 mm^2	63%
Application Mix 3. JPEG encoding MPEG decoding PGP encryption G.721 encoding	ARM SA-110	1KB	2KB	39 mm^2	10%

Table 8: Area-minimal system configurations for two application mixes.



(a) Application Mix 1. (see Table 8).

(b) Application Mix 2. (see Table 8).

(c) Application Mix 3. (see Table 8).

Figure 14: Trace driven simulation for area minimal cache subsystem determination for application mixes.

We assembled three application mixes and performed search for area minimal configurations that satisfy performance requirements of all applications in the mix. The results are presented in Table 8 and the equivalent diagrams are shown in Figure 14. The columns in Table 8 correspond to the columns in Table 7. The area-optimal cache configuration for each application mix is selected among cache

subsystems with dominating performance (bold line in Figure 14).

The percentage of the total IC's area dedicated to the cache subsystem, as well as strong diversity among selected minimal area cache subsystems for different applications and different timing constraints indicate the necessity of involving sophisticated system level synthesis approaches in order to minimize product's total area while still satisfying product's quality of service.

7 Conclusion

Trends in the application and semiconductor industry indicate that future application specific single-chip systems will have most of the die area dedicated to the cache subsystem. We developed a novel hierarchical approach for synthesis of programmable core-based applications. The synthesis approach is built around an efficient algorithm for mapping applications to I-caches. The algorithm is used as an engine in the search for an application-optimized area-minimal 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 non-preemptive target applications.

To support the evaluation of our synthesis paradigm we developed a performance estimation platform which integrated existing modeling, profiling and simulation tools with the new system-level synthesis tool. The efficiency of the synthesis approach was tested on a set of modern multimedia and communication applications.

References

[AM95] G. Araujo, S. Malik, "Optimal code generation for embedded memory non-homogeneous register architectures", Proceedings of the Eighth International Symposium on System Synthesis, Cannes, France pp. 36-41, 1995.

[Ber94] B.N. Bershad, et al. "Avoiding conflict misses dynamically in large direct-mapped caches", Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, Vol.29, No.11, pp. 158-70, 1994.

[Bol94] I. Bolsens, K. Rompaey, H. De Man, "User requirements for designing complex systems on silicon", VLSI Signal Processing VII, La Jolla, CA, pp. 63-72, 1994.

[Chi96] M. Chiodo, et. al. "A case study in computer-aided co-design of embedded controllers", Design Automation for Embedded Systems, Vol.1, No.1-2, pp. 51-67, 1996.

[Chu96] K.C. Chung, R.K. Gupta, C.L. Liu, "An algorithm for synthesis of system-level interface circuits", International Conference on Computer-Aided Design, San Jose, CA, pp. 442-7, 1996.

[Cme94] B. Cmelik, D. Keppel, "Shade: a fast instruction-set simulator for execution profiling", SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Vol.22, No.1, pp. 128-37, 1994.

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

[Far89] S. McFarling, "Program optimization for instruction caches", Third International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, pp. 183-191, 1989.

[Fis96] J.A. Fisher, et al. "Custom-fit processors: letting applications define architectures", Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-29, Paris, France, pp. 324-335, 1996.

[Fly96] M.J. Flynn, "Computer Architecture: Pipelined and Parallel Processor Design", Jones and Bartlett, Boston, MA, 1996. (http://umunhum.Stanford.EDU/tools/cachetools.html)

[Fu96] S.T. Fu, D.F. Zucker, M.J. Flynn, "Memory hierarchy synthesis of a multimedia embedded processor", International Conference on Computer Design, Austin, TX, pp. 176-184, 1996.

[Gaj94] D.D. Gajski, et al. "A System-Design Methodology: Executable Specification Refinement", Euro-DAC '94, Paris, France, pp. 458-463, 1994.

[Gar79] M.R. Garey, D.S. Johnson, "Computers and intractability : a guide to the theory of NP-completeness", W. H. Freeman, San Francisco, CA, 1979.

[Gup93] R.K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems", IEEE Design and Test of Computers, Vol.10, No.7, pp. 29-41, 1993.

[Hen95] J. Henkel, R. Ernst, "A path-based technique for estimating hardware runtime in HW/SW-cosynthesis", Proceedings of the Eighth International Symposium on System Synthesis, Cannes, France, pp. 116-121, 1995.

[Hill88] M.D. Hill, "A case for direct-mapped cache", IEEE Computer, pp. 25-40, 1988.

[Hill89] M.D. Hill, A.J. Smith, "Evaluating Associativity in CPU Caches," IEEE Trans. on Computers, pp.1612-1630, 1989.

[Hwu89] W.W Hwu, P.P Chang, "Achieving high instruction cache performance with an optimizing compiler", 16th Annual International Symposium on Computer Architecture, Jerusalem, Israel, pp. 242-251, 1989.

[Ism94] T. B. Ismail, K. O'Brien, A. Jerraya, "Interactive System- Level Partitioning with PARTIF", Euro-DAC '94, Paris, France, pp. 464-468, 1994.

[Jou93] N.P. Jouppi, "Cache write policies and performance", 20th Annual International Symposium on Computer Architecture ISCA '20, San Diego, CA, Vol.21, No.2, pp. 191-201, 1993.

[Kir97] D. Kirovski, M. Potkonjak, "System level synthesis of low-power hard real time systems", to appear, 34th Design Automation Conference, Annaheim, CA, 1997.

[Lee96] T.-G. Lee, W.-J. Fang, A.C.-H. Wu, "The design and implementation of a cooperative design-view environment for interactive partitioning applications", Software - Practice and Experience, Vol.26, No.10, pp. 141-160, 1996.

[Li96] Y.-T. S. Li, S. Malik, A. Wolfe, "Cache modeling for real-time software: beyond direct mapped instruction caches", 17th IEEE Real-Time Systems Symposium, Los Alamitos, CA, pp. 254-263, 1996.

[Lie96] C. Liem, P. Paulin, A. Jerraya, "Address calculation for retargetable compilation and exploration of instruction-set architectures", 33rd Design Automation Conference, Las Vegas, NV, USA, 1996.

[Lin96] B. Lin, S. Vercauteren, H. De Man, "Embedded architecture co-synthesis and system integration", Fourth International Workshop on Hardware/Software Co-Design, Pittsburgh, PA, pp. 2-9, 1996.

[MPR96] Microprocessor Report, all isues, 1996.

[Pan96] P. Panda, N. Dutt, A. Nicolau, "Memory organization for improved data cache performance in embedded processors", 9th International Symposium on System Synthesis, La Jolla, CA, pp. 90-95, 1996.

[Pet90] K. Pettis, R.C. Hansen, "Profile guided code positioning", ACM SIGPLAN '90 Conference on Programming Language Design and Implementation, White Plains, NY, Vol.25, No.6, pp. 16-27, 1990.

[Pot95] M. Potkonjak, W.H. Wolf, "Cost Optimization in ASIC implementation of Periodic Hard-Real Time Systems using Behavioral Synthesis Techniques", ICCAD95, San Jose, CA, pp. 446-451, 1995.

[Smi83] J.E. Smith, J.R. Goodman, "A study of instruction cache organizations and replacement policies", IEEE Transactions on Software Engineering, Vol.9, No.3, pp. 132-137, 1983.

[Sub96] P.A. Subrahmanyam, A. Sharma, "Issues in developing real-time multimedia applications for a multiprocessor system", 4th Intl. Workshop on Parallel and Distributed Real-Time Systems, Honolulu, HI, pp. 147-155, 1996.

[Tom96] H. Tomiyama, H. Yasuura, "Size-constrained code placement for cache miss rate reduction", 9th International Symposium on System Synthesis, La Jolla, CA, pp. 96-101, 1996.

[Tor96] C. Xia, J. Torrellas, "Instruction prefetching of systems codes with layout optimized for reduced cache misses", The 23rd Annual International Conference on Computer Architecture, Philadelphia, PA, Vol.24, No.2, pp. 271-82, 1996.

[Wol94] W.H. Wolf: "Hardware-Software Co-Design of Embedded Systems", Proceedings of the IEEE, Vol.82, No.7, pp. 967-989, 1994.