Application-Driven Synthesis of Core-Based Systems

Darko Kirovski*, Chunho Lee*, Miodrag Potkonjak*, and William Mangione-Smith°
* Computer Science Department, ° Electrical Engineering Department
University of California, Los Angeles, CA 90095-1596

Abstract

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

A typical application-specific core-based system consists of a processor core, instruction and data cache cores, and number of hardware accelerators and control blocks. Numerous ASIC manuals and documents outline the fact that most of the IC area is dedicated to the processor core and associated caches. We present a novel hierarchical fully modular approach for synthesis of core-based data-intensive ASICs that focuses on minimizing area assigned to the processor core and the cache subsystem with respect to the performance requirements of a set of target applications. The system-level approach for design of core-based implementations uses as a performance optimization technique a global least-constraining most-constrained probabilistic heuristic as a compile-time strategy for basic block static relocation [Kir97]. The goal of the synthesis approach is to select an area-minimal processor core and cache configuration that satisfy the performance requirements of a set of target applications.

Trace-driven simulation is used to accurately evaluate performance of the cache subsystem and to obtain the application execution profile. The on-line Stanford Cache Design Tool (CDT [Fly96]) is used to estimate cache access latency and area based on properties such as cache total size, line size, 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.

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 [Hill88]) tools with the developed system-level synthesis tools. Application profiles are acquired through the use of SHADE augmented with custom 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 protocols.

2 Previous Work

The related work can be traced through two areas: application-specific system optimization and evaluation.

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 [Pot95, Chi96, Sub96]. As embedded applications have become more sophisticated and commercially relevant, system level synthesis has also become increasingly important [Gup93, Gaj94, Hen95, Lee96].

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]. Instruction and data caches, as the highest level of the memory hierarchy, have been thoroughly studied in [Hill88, Jou93].

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 to enable hardware savings.

3 Preliminary Discussion

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 approach to be rapidly updated and applied to new environments with new technology.

The on-line Stanford CDT is used to evaluate the impact of cache design choices on area and latency. Caches typically found in current embedded systems range from 128B to 32KB. Since higher associativity can result in significantly higher access time, we consider only direct mapped, and 2-way set associative caches. We used a fixed cache line size of 32 bytes. This decision attempts

0-89791-993-9/97 $10.00

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 and external bus width for each system investigated. This penalty ranges between 4 and 20 system clock cycles. Write-back was adopted instead of write-through due its superior performance in uniprocessor systems [Jou93], though at increased hardware cost. Each of the processors considered is constrained by the Flynn limit, and thus is able to issue at most a single instruction per clock period. The caches were designed to have a single access port. 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 6 transistors per CMOS SRAM cell.

Cache	Property	1KB	2KB	4KB	8KB	96KB
Direct Mapped	Area(mm^2)	2.11	3.81	7.20	13.96	161.7
	Clk(ns)	3.79	3.97	4.16	4.5	6.79
2-way	Area(mm^2)	2.38	4.02	7.29	13.8	156.1
	Clk(ns)	5.67	5.75	5.94	6.23	8.54

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

Microprocessor	MHz	MIPS	Tech.(μm)	mm^2
ARM 810	75	80	0.5	29
Motorola 403GC	33	41	0.5	30
ARM StrongArm	200	230	0.35	33
i486GXSF	33	18	0.8	18
MIPS4650	133	175	0.6	32
Hitachi SH7708	100	100	0.6	34
LSI Logic CW4001	60	53	0.5	3.5
LSI Logic TR4101	81	30	0.35	2

Table 2: A sample of the processor model.

Information on microprocessor performance and area was collected 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.

4 The New Synthesis Approach

Figure 1 illustrates the synthesis framework. In this section we describe the role of each module and how modules are combined into an integrated synthesis system.

The core module in the synthesis flow is a global least-constraining most-constrained heuristic that guides the algorithm for basic block relocation [Kir97]. Basic blocks are repositioned statically, in a way that frequently sequentially executed basic blocks are mapped into different cache lines. 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.

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 conservative sharp bounds and by providing powerful 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. 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. The configuration which has the minimum processor and cache total area is returned as the best solution.

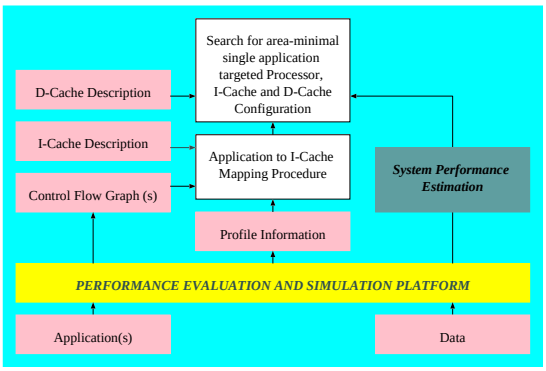

Figure 1: The hierarchical modular synthesis approach.

System performance is evaluated using a platform which integrates simulation, modeling, and profiling tools (see Figure 2). 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 provides to the translated code a stream of data 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. We also collect the frequencies of control transfers through each basic block, and branch temporal correlation [Kir97]. 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.

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 into the optimized address space. The remapped trace of addresses, along with all unmodified data memory references, are sent to DINEROIII for cache simulation.

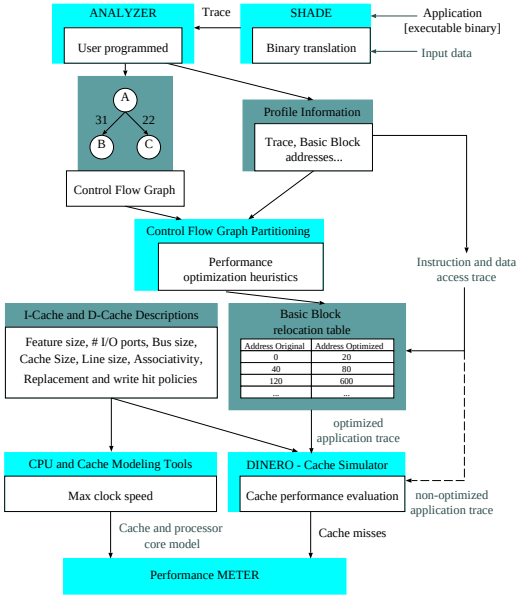

Figure 2: Synthesis flow of application-driven programmable core-based systems.

The final system performance was computed using the following formula: $CPI = \frac{SysClkFreq}{MIPS} + CacheMissRatio \cdot CacheMissPnlty$, where $CacheMissRatio$ was computed during the trace driven simulation of the cache subsystem. $CacheMissPnlty$, $SysClkFreq$, and $MIPS$ are system parameters introduced in Section 3.

5 Synthesis Optimization Algorithms

The problems encountered in the synthesis of systems on silicon and competitive optimization algorithms are described in this section. 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 application-driven system resource allocation a set of non-preemptive target applications.

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.

Figure 3: Pseudo code for the resource allocation procedure.

The proposed application-driven system-level synthesis technique employs basic block repositioning based upon profile information for optimization of the program execution performance. Block repositioning aims for application execution on fixed hardware resources with minimal number of cache misses. The optimization problems involved in the basic block relocation are defined, their computational complexity is established, and an efficient algorithm is proposed in [Kir97].

The second phase of the synthesis strategy conducts a search 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 3.

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 I-cache sizes, a search is performed in order to find the smallest D-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 total area. The search is terminated when the current solution is dominated by some existing minimal solution. Dominated solutions are found in one of the following two ways:

- For Fixed core A, the 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 is 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:

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

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 a 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. In

order to provide multiple application targeted resource allocation, the algorithm presented in Figure 3 needs only be augmented with an improved selection step. The modification includes considering a conjunction of application requirements, rather than fulfilling a single performance constraint (see Figure 4).

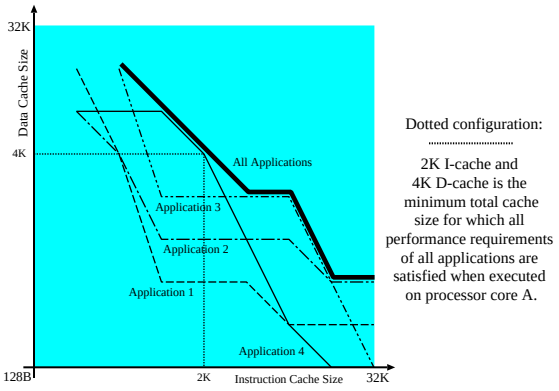

Figure 4: Best instruction cache setups for a single core, four applications with fixed performance requirements and a set of fixed data cache sizes.

6 Experimental Results

We used six applications to demonstrate the effectiveness of the approach: JPEG software (Independent JPEG Group), MPEG-1 and MPEG-2 software (MPEG Software Simulation Group), PGP software, and implementation of the GSM and G.721 wireless communication protocols. The developed application driven synthesis approach resulted in area-minimal system configurations as presented in Table 3. The first column contains the application description. The next three columns describe the allocated core and cache configuration followed by the total required area for the system. The obtained results acknowledge the domination of LSI Logic cores due to their exceptional performance vs. area ratio. The size of these cores 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 diversity of selected cache structures implies that finding the area-minimal cache subsystem is an important task.

Appl.	Core	I-$	D-$	mm^2	$/Total
JPEG enc.	CW4001	1KB	1KB	7.7	43%
JPEG dec.	CW4001	1KB	2KB	9.4	63%
PGP encr.	TR4101	512B	1KB	5.0	60%
MPEG dec.	StrongArm	1KB	512B	36	8%
G.721 enc.	TR4101	1KB	128B	4.3	53%
GSM enc.	CW4001	512B	128B	4.7	25%

Table 3: Area-minimal system configurations for a given applications.

Appl.	Processor	I-$	D-$	mm^2	$/Total
G.721 enc. MPEG dec. JPEG dec. PGP encr.	StrongARM	1KB	1KB	37	9%
GSM enc. JPEG dec. JPEG enc. PGP encr.	CW4001	1KB	2KB	9.4	63%

Table 4: Area-minimal system configurations for two application mixes.

We assembled two 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 4. The columns in Table 4 correspond to the columns in Table 3. The area-optimal cache configuration for each application mix is selected among cache subsystems with dominating performance. 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 timing constraints.

7 Conclusion

We developed and evaluated a novel hierarchical approach for synthesis of programmable core-based applications. The approach synthesizes an area-optimal processor and cache system for the most common system environment, a set of non-preemptive target applications. The efficiency of the synthesis approach was tested on a set of modern multimedia and communication applications.

References

[Bol94] I. Bolsens, K. Rompaey, H. De Man, "User requirements for designing complex systems on silicon", VLSI Signal Processing VII, 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.

[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/

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

[Gaj94] D.D. Gajski, et al. "A System-Design Methodology: Executable Specification Refinement", Euro-DAC, 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", 8th International Symposium on System Synthesis, pp. 116-121, 1995.

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

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

[Kir97] D. Kirovski, et al., "Synthesis of core-based application-specific systems", Technical Report, CSD, UCLA, 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.

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

[MPR96] Microprocessor Report, all isues, 1996.

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

[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, pp. 147-155, 1996.