Exploring Hypermedia Processor Design Space

Chunho Lee†, Johnson Kin‡, Miodrag Potkonjak† and William H. Mangione-Smith‡

†Department of Computer Science and ‡Department of Electrical Engineering
University of California, Los Angeles, CA, USA

E-mail: {leec, miodrag}@cs.ucla.edu, {johnsonk, billms}@icsl.ucla.edu

Abstract

Distributed hypermedia systems that support collaboration are important emerging tools for creation, discovery, management and delivery of information. These systems are becoming increasingly desired and practical as other areas of information technologies advance. A framework is developed for efficiently exploring the hypermedia design space while intelligently capitalizing on tradeoffs between performance and area. We focus on a category of processors that are programmable yet optimized to a hypermedia application.

The key components of the framework presented in this paper are a retargetable instruction-level parallelism compiler, instruction level simulators, a set of complete media applications written in a high level language, and a media processor synthesis algorithm. The framework addresses the need for efficient use of silicon by exploiting the instruction-level parallelism found in media applications by compilers that target multiple-instruction-issue processors.

Using the developed framework we conduct an extensive exploration of the design space for a hypermedia application. We find that there is enough instruction-level parallelism in the typical media and communication applications to achieve highly concurrent execution when throughput requirements are high. On the other hand, when throughput requirements are low, there is little value in multiple-instruction-issue processors. Increased area does not improve performance enough to justify the use of multiple-instruction-issue processors when throughput requirements are low.

The framework introduced in this paper is valuable in making early architecture design decisions such as cache and issue width trade-off when area is constrained, and the number of branch units and instruction issue width.

1. Introduction

In the last decade multimedia services have found increase use in application systems such as education and training, office and business, information and point of sales [26]. In the recent years, the wide spread use of the World Wide Web has produced a fertile ground for multimedia services [4, 5].

Hypermedia represents a combination of hypertext and multimedia technologies [17, 20, 29]. The concept of hypertext was proposed more than 50 years ago by V. Bush [6] but T. Nelson [37] is generally credited as the first to use the term "hypertext" [18].

There are two very different models for hypermedia [24]. One model uses hypermedia to deliver rigidly constrained embedded applications, e.g. stand-alone CD-ROM based applications. The other model is distributed hypermedia as a wide-area system for information discovery and management, e.g. World Wide Web [5] and Hyper-G [23].

The common model for distributed hypermedia information systems have three distinct roles that are supported: *passive participant*, *information/service provider*, and *active participation* [24]. The last role is an emerging model for collaboration. From the hypermedia processor designer's standpoint, the roles hypermedia systems must support present a unique challenge since most media tasks are computationally demanding and execute concurrently, yet require reliable and predictable operation. For example, tasks such as video and audio encoding/decoding, text processing, image processing, authentication and encryption/decryption should run in parallel to support hypermedia systems for collaboration.

We present an approach to distributed hypermedia system design space exploration for applications that support collaboration roles. We focus on a category of processors that are programmable yet optimized to run a hypermedia application. The approach utilizes advances in compiler technology and architectural enhancements. Advances in compiler technology for instruction-level parallelism (ILP) have significantly increased the ability of a microprocessor to exploit the opportunities for parallel execution that exist in various programs written in high-level languages. State-of-the-art ILP compiler technologies are in the process of migrating from research labs to product groups [3, 12, 21, 33, 34]. At the same time, a number of new microprocessor architectures have been introduced. These devices have hardware structures that are well matched to most ILP compilers. Architectural enhancements found in commercial products include predicated instruction execution, VLIW execution and split register files [8, 43]. Multi-gauge arithmetic (or variable-width SIMD) is found in the family of MPACT architectures from Chromatic [22] and the designs from MicroUnity [19]. Most of the multimedia extensions of programmable processors also adopt this architectural enhancement [28, 39].

The key components of the framework presented in this paper are a retargetable ILP compiler, instruction level simulators, a set of complete media applications written in a high level language and a media processor synthesis algorithm.

We discuss the related works and our contributions in Section 2. Section 3 presents the preliminary materials including the area model, the media application set and the experiment platform such as tools and procedures of measuring application characteristics using the tools. Section 4 formulates the search problem and establishes its complexity. Based on the problem formulation, we lay out the overall approach to area efficient hypermedia processor synthesis. The solution space exploration strategy and algorithm is described in Section 5. The tools and algorithms are extensively studied through experimentation in Section 6. Finally, Section 7 draws conclusions.

2. Related Works and Our Contributions

There are many articles that summarize research efforts in hypermedia systems [17, 20, 29]. The most common Internet-based hypermedia technologies, World Wide Web [5] and Hyper-G [23], represent distributed hypermedia systems that are widely available.

The concept of distributed hypermedia systems that support collaboration are emerging as an important idea. Distributed hypermedia systems are becoming increasingly desired and practical as other areas of information technologies advance [24].

Since the early 90's, there has been a number of efforts related to the design of application-specific programmable processors and application-specific instruction sets. Comprehensive survey of the works on computer-aided design of application specific programmable processors can be found in the literature ([16], [38], and [35]). In particular, a great deal of effort has been made in combining retargetable compilation technologies and design of instruction sets [1], [32], [42], [31], [30]. Several research groups have published results on the topic of selecting and designing instruction set and processor architecture for a particular application domains [44], [25]. Potkonjak and Wolf introduced hard real-time multitask ASIC design methodology by combining techniques from hard real-time scheduling and behavioral synthesis [40, 41].

Early work in the area of processor architecture synthesis tended to employ ad hoc methods on small code kernels, in large part due to the lack of good retargetable compiler technology. Conte and Mangione-Smith [9] presented one of the first efforts to consider large application codes written in a high-level language (SPEC). While they had a similar goal to ours, i.e. evaluating performance efficiency by including hardware cost, their evaluation approach was substantially different. Conte, Menezes, and Sathaye [10] further refined this approach to consider power consumption. Both of these efforts were limited by available compiler technology, and used a single applications binary scheduled for a scalar machine for execution on superscalar implementations. Fisher, Faraboschi and Desoli [13] studied the variability of applications-specific VLIW processors using a highly advanced and retargetable compiler. However, their study considered small program kernels rather than complete applications. They also focused on finding the best possible architecture for a specific application or workload, rather than understanding the difference between best architectures across a set of applications.

Unfortunately, however, we know of no work addressing synthesis of distributed hypermedia processors that can

support collaboration. As quality requirements changes, it is necessary to take into account not only timing and synchronization requirements, but also throughput requirements when designing a hypermedia processor. The benefits of advances in compiler technology and architectures also should be incorporated in designing hypermedia systems as low-level programming is increasingly less practical.

We focus on a category of processors that are programmable yet optimized to a hypermedia application for varying degrees of throughput. We use a state-of-the-art ILP compiler and simulation tools to evaluate performance of media applications on a variety of multiple-instruction-issue processors. We develop an efficient optimal solution algorithm for the synthesis of hypermedia processors. Although the synthesis problem is NP-complete, we find that the optimal solution algorithm is practical for a typical hypermedia scenario.

The design space exploration experiment reported in this paper does not include dynamic resource allocation where instances of media tasks arrive and resources are dynamically allocated for arriving tasks. Our main objective is to study the viability of design space exploration for hypermedia processors using media applications written in a high-level language.

3. Preliminaries

In this section we provide the definitions of terms and present existing foundations. After describing the target architecture and the area model, the media workload is introduced. In the last subsection, we explain the experimental platform including tools and procedures of measuring application characteristics using the tools.

3.1. Definitions and Assumptions

We use the terms task, individual application and media application interchangeably to refer to a task in a hypermedia application. We assume that an integrated circuit can be physically partitioned to run different tasks in parallel. Hence, each partition should be a complete processor. Processor, machine and machine configuration are used interchangeably throughout the paper. Depending on the context they are used in, they refer either to a single partition within a processor for a hypermedia application or to the entire set of processors for a hypermedia application.

On the run time measurement platform, individual applications are divided into fixed run time units called *quanta*. In our experiment, we use as one quantum length the time taken to encode or decode 4 MPEG-2 frames. For a typical throughput of 15 frames per second, the quantum size is approximately 0.267 seconds. The performance constraints used to drive our experiments are based on the number of processor cycles equivalent to at least one quantum.

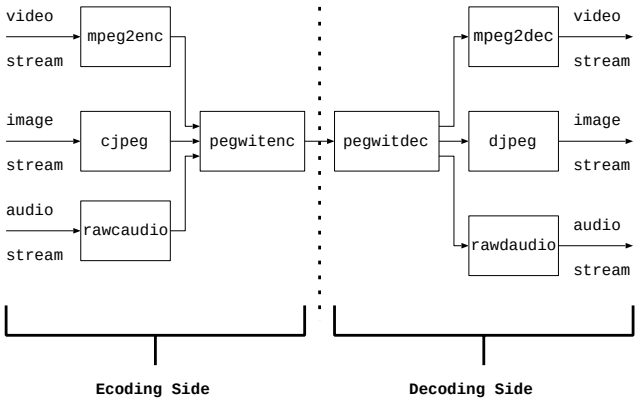

Figure 1. A hypermedia scenario

3.2. Target Architecture

The target architecture we use resembles a multiprocessor system with shared memory except that all the processors for a given usage scenario are laid out on a single die. A media task is assigned to its dedicated processor. More than one media application can be assigned to a processor if the given performance constraints are guaranteed to be met, i.e. all media tasks on the processor must be finished within a given time limit. Shared memory is used for data communication between tasks, with each processor maintaining its own cache. When multiple media applications are assigned to a single processor, flushing and refilling of the cache is accurately incorporated into the run time measurement platform.

As defined previously, we divide tasks into quanta. One of the benefits of using the notion of a quantum is that it simplifies synchronization of several applications running on multiple processors. Because most hypermedia applications have real-time performance characteristics, the user receives no benefit from performance that exceeds the specified goals. Hence, the use of quanta equivalent to the longest time frame with which tasks should be synchronized gives a convenient task assignment unit to allocate resources.

Figure 1 shows one hypermedia scenario that we will use to evaluate our framework. The scenario consists of 8 media applications, which are separated into two distinctive sets of media tasks: the encoder group and the decoder group. The figure illustrates synchronization boundaries between media tasks. For example, during the first quantum, video, audio or still images are encoded. At the next quantum, the encoded data sets are encrypted, while new video, audio and still images are encoded in pipeline fashion.

3.3. Area Model

We use the Intel StrongArm SA-110 as the baseline architecture for our analysis [36]. The device is a single-issue processor with the classic five stage pipeline. The SA-110 has an instruction issue unit, integer execution unit, integer multiplier, memory management unit for data and instructions, cache structures for data and instructions,

Configuration	Issue	IALU	Branch	Mem	Cache	Total
(1, 1, 1, 1, .5, .5)	1.25	2.5	1.25	5.0	1.53	14.01
(2, 2, 1, 2, 1, 1)	2.5	5.0	1.25	10.0	2.54	23.77
(4, 4, 1, 4, 2, 2)	5.0	10.0	1.25	20.0	4.55	43.28
(8, 8, 4, 8, 8, 8)	10.0	20.0	5.0	40.0	16.55	94.03

Table 1. Processor configuration examples and their area estimates (mm^2): a machine configuration consists of (issue width, number of ALUs, number of branch units, number of memory units, size of instruction cache(KB), size of data cache(KB))

and additional units such as phase locked loop (PLL). It is fabricated in a 0.35-μm three-metal CMOS process with 0.35V thresholds and 2V nominal supply voltage.

We develop a simple area model based on SA-110. The area of the chip is 49.92mm^2 (7.8mm $\times$ 6.4mm). Approximately 25% of the die area is devoted to the core (12.48mm^2). The issue unit and branch unit occupies approximately 5% of the die area (2.50mm^2). The integer ALU and load/store unit consume roughly 5% of the die area (2.50mm^2). The DMMU and IMMU (data and instruction MMU) occupies roughly 10% (5mm^2) of the area. The rest of the core area is used by other units such as the write buffers and bus interface controller. We assume that the area of miscellaneous units is relatively stable in the sense that it does not change as we increase the issue width or cache sizes. We further assume a VLIW issue unit area model which is generally of complexity $O(n)$.

The model is based on area models used in [13] and [2]. In particular, we use partitioned register files and multi-cluster machines to simplify the model. Consequently, the areas of components such as datapath and register files, that are generally of super-linear complexity, can be assumed to be of linear complexity. The chip area model is linear. Cycle speed can remain constant across various machine configurations when multi-cluster machines are used. The model may not be extremely accurate, but it may be good enough to demonstrate the framework without going into the details of building the entire machine. The area of an arbitrarily configured VLIW machine is given by

$$\begin{aligned} Area &= n_{issue}A_{issue} + n_{ALU}A_{ALU} + \\ &\quad n_{branch}A_{branch} + n_{mem}A_{mem} + A_{misc} \end{aligned} \tag{1}$$

The terms n_{issue}, A_{issue}, n_{ALU}, A_{ALU}, n_{branch}, A_{branch}, n_{mem}, A_{mem} and A_{misc} are the issue width, the baseline issue unit area, the number of ALUs, the area of a single ALU, the number of branch units, the branch unit area, the number of memory units, the area of single memory unit and miscellaneous area, respectively.

We did not include floating-point units in any machine configurations because the applications we used have only

```
$define ISSUE 1
$define IALU 1
$define BRANCH 1
$define MODEL superscalar
. . .
# Enumerate resources
(Resources declaration
  slot[0..$ISSUE$]
  ialu_0[0..$IALU$]
  mem_0[0..$ISSUE$]
  branch[0..$BRANCH$]
end)
. . .
```

Figure 2. An example High-Level Machine Description (HMDES)

integer operations. All functional units have multipliers. Cache area is calculated using the Cache Design Tools [14]. We use the same cache parameters for all cache configurations except their size and the number of read/write ports: 64 bytes per line and single bank. The external bus width is 64 bits and latency is 4 cycles. We assume that the number of read/write ports are the same as the number of functional units.

A set of example area estimates for superscalar machines with different cache and core configurations are shown in Table 1. In the rest of the paper we describe a machine configuration by a 6-tuple as shown in Table 1.

3.4. Media Applications

The set of media applications used in this experiment is composed of complete applications which are publically available and coded in a high-level language. We use 8 applications culled from available image processing, communications, cryptography and DSP applications. Brief summaries of applications and data used are shown in Table 3. More detailed descriptions of the applications can be found in [27].

3.5. Experiment Platform

We use the IMPACT tool suit [7] to measure run times of media applications on various machine configurations. The IMPACT C compiler is a retargetable compiler with code optimization phases especially developed for multiple-instruction-issue processors. The target machine for the IMPACT C can be described using the high-level machine description language (HMDES). A high-level machine description supplied by a user is compiled by the IMPACT machine description language compiler. Figure 2 shows an example HMDES file.

IMPACT provides cycle-level simulation of both the processor architecture and implementation. The optimized code is consumed by the Lsim simulator. At simulation time, Lsim takes cache structure information provided by a user. Figure 3 shows the flow of simulation using IMPACT tools.

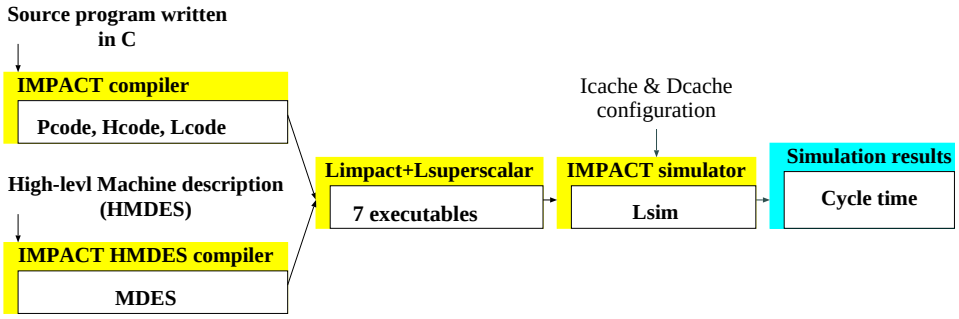

Figure 3. Performance measurement flow using IMPACT tools

Configuration	Area (mm^2)	A_1	A_2	A_3
M_1(1, 1, 1, 1, 0.5, 0.5)	14.01	15	20	20
M_2(2, 1, 1, 2, 2, 2)	25.76	10	10	15
M_3(4, 4, 1, 4, 2, 2)	43.28	5	10	5

Table 2. An illustrative run time examples: a machine configuration consists of (issue width, number of ALUs, number of branch units, number of memory units, size of instruction cache(KB), size of data cache(KB))

Application	Instr.[a]	Source	Description	Data file[b]	Data Description
JPEG encoder	13.9	Independent	JPEG image	101,484	PPM (bit map)
JPEG decoder	3.8	JPEG Group	encoding/decoding	5,756	JPEG compressed
MPEG encoder	1,121.3	MPEG Simul-	MPEG-2 movie	506,880	YUV 4 frames
MPEG decoder	175.5	ation Group	encoding/decoding	34,906	MPEG-2 (http://www.mpeg2.de/)
Pegwit encryption	34.0	George Barwood	encryption/	91,503	plain ASCII
Pegwit decryption	18.5		decryption	91,537	Pegwit encrypted
ADPCM encoder	6.8	Jack Jansen	speech compression	295,040	16 bit PCM
ADPCM decoder	5.9		and decompression	73,760	ADPCM encoded

Table 3. A brief description of applications and data used in the experiment

[a] Dynamic instruction count measured using SpixTools on SPARC5 (in millions)
[b] in bytes

4. Problem Formulation

Informally the problem can be stated as follows. For a given set of media applications and their performance constraints, synthesize an area optimal processor that guarantees the timing requirements of the applications.

Under the assumptions given in Section 3, run times of each media task on each architecture in consideration are measured. The mesaured run times can be organized in a table as shown in Table 2.

One obvious, albeit sub-optimal, solution can be obtained directly from the table by selecting best processors for each individual task. Subsequently, an individual task is assigned to a corresponding processor that guarantees run time constraints. For example, from Table 2 we can choose processors M_1, M_2 and M_2 for media applications A_1, A_2 and A_3, respectively. This simple-minded solution can be good enough when we cannot find much parallelism across all the tasks and the run times of applications on selected processors are similar. That is, if we cannot reduce run times of applications by increasing resources of a processor. If no significant parallelism is present in tasks, a multiple-instruction-issue processor is little better than a single-instruction-issue processor. In other words, it is not possible to reduce the run times of individual task and to assign more than one task to a multiple-instruction-issue processor, while satisfying timing requirements, just because it has more resources. This approach, however, will result in grossly inefficient solutions if run times are not similar across tasks. Assuming that M_4 is not available in the example, the above mentioned solution is the optimum solution.

Unfortunately, as alluded before, the solution for the problem is no longer trivial when there is a need to run more than one task on a processor. This situation results when we can find enough parallelism across tasks and performance requirements are such that we can justify the use of multiple-instruction-issue processors, thus requiring merging more than one tasks into a single processor. For example, from Table 2, we can choose M_1 and M_3 since A_2 and A_3 can run on M_3 in turn without violating timing requirements. The sum of areas of the two processors (57.29 mm^2) is smaller than that of the simple-minded solution (65.53 mm^2) discussed above. In this case, the problem size is much bigger than the simple-minded case. This is similar to the Bin Packing Problem (BPP) [11]. In fact, we can transform the BPP to this problem in polynomial time as will be shown next in this section. Under this scenario, we have $2^n - 1$ task subsets (power set of the task set) that should be considered to choose up to n processors, where n is the number of tasks.

We now define the problem using more formal Garey-Johnson format [15].

Selection Problem

Instance: Given a set T of n media applications, a_i, $i = 1, 2, ..., n$, a set of m processors, c_j, $j = 1, 2, ..., m$, the run times e_{ij} of the media applications a_i, $i = 1, 2, ..., n$ on the machines c_j, $j = 1, 2, ..., m$ and constants C and E,

Question: Is there a multisubset (subset in which more than one instance of a processor can be included) M of k

processors, c_p, $p = 1, 2, ..., k$, $1 \leq k \leq n$, such that $\sum_{j \in M} A_{c_j} \leq C$ and $\max_{j \in M} \sum_{i \in t_j} e_{ij} \leq E$? A_{c_j} is the area of the machine c_j and t_j is the set of tasks that are assigned to the machine j.

Theorem. The Selection Problem is NP-complete.

Proof. The Bin Packing Problem can be mapped to a special case of the Selection Problem. For a given task set T, $|T| = n$ and an integer $k \in \{e | e \in Z, 1 \leq e \leq n\}$, we map $2^n - 1$ objects to $2^n - 1$ subsets (excluding the empty set from the power set of T) of T. The k bins in the BPP are mapped to the k processors.

Note, however, that the simulation time to measure run times of each task on each processor takes well over two weeks depending on the experiment setup, it is well worth to try to devise an efficient algorithm to obtain the optimum solutions. In the following section, we explain an efficient optimal algorithm for the problem.

5. System Synthesis

In this section we informally explore the processor selection space and describe the framework of the approach. We elaborate an efficient algorithm for optimum solutions.

5.1. Global Design Flow

We collect run times (expressed as a number of cycles) of the media applications on 175 different machine configurations (25 cache configurations for 7 processor configurations). First we build executables of the media applications on seven different architectures. we did not modify application in any way to affect the compiler ability to find available ILP. In fact, we use the applications "out of the box" except that we eliminate graphical screen outputs. All the compilation was done with advanced scheduling features designed for multiple-issue-machines turned on.

The processors considered are machines with a single branch unit and one of the one-, two-, four-, and eight-issue units, machines with two branch units and one of the four- and eight-issue units, and machines with four branch units and a eight-issue unit. The IMPACT compiler generates aggressively optimized code to increase achieved ILP for each core. The optimized code is consumed by the Lsim simulator. We simulate the applications for a number of different cache configurations. For each executable of a benchmark, we simulate 25 combinations of instruction cache and data cache ranging from (512 bytes, 512 bytes) to (8 KB, 8 KB).

After all the simulations are completed, we run an efficient selection algorithm to select machine configuration sets under various performance constraints. Figure 4 shows the global flow of design process.

5.2. Synthesis Algorithm

In the following section we develop a branch and bound based algorithm for the selection of area minimal hyper-media processor configurations. From the run times measured as illustrated in Table 2, the algorithm examines all

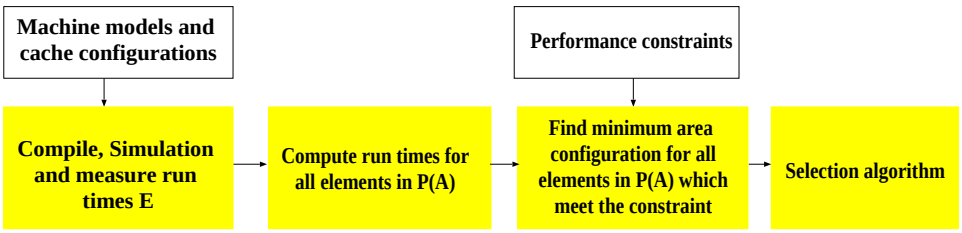

Figure 4. Global design flow. E is the set of run times and P(A) is the power set of a given media tasks

possible combinations of media task run times to find a best processor configurations.

Before entering the branch and bound loop, the algorithm finds the area optimal processor for each element of the power set of the media applications. This step eliminates all processors that are sub-optimal in terms of running partial task sets the size of which ranges from 1 to n, n being the number of media tasks. If, for a subset of tasks, we cannot find a processor that can satisfy a given timing constraint, then there is no feasible solution that run the particular subset of tasks on a processor. The algorithm is given in Figure 5 and 6.

The run time of the algorithm is dependent on two factors: the size of a given task set and the timing constraint. It is apparent that the size of a given task set affects the run time as the combination of tasks is exponential with respect to the number of tasks. As we lower the timing constraint, the number of feasible processors increases. This increase results in limited pruning of possible combinations of tasks in early stages of the algorithm. In our experiment, which is run on a SPARC4 machines and reported later, the run times of the algorithm ranges from less than one second for the strictest timing constraint to approximately 50 seconds for the least strict timing constraint.

6. Experimental Results

We evaluate the framework presented in this paper by conducting an experiment with a set of 8 media applications shown in Figure 1. The range of the performance constraints we examined is from 5.7×10^6 to 4.47×10^7 cycles, which is the maximum amount of time allotted to finish processing a *quantum* worth of computation. In our experiment, a quantum size is equal to 0.267 seconds, which implies that the speed constraint of the processors is ranging from 21.37MHz to 167.6 MHz.

In this experiment, for the tightest constraint (5.7 million cycles) six processor components are selected. As we allow more cycle times to finish a *quantum*, we need less processors and less cache memory. Note that processor configurations with same numbers (e.g., 1^{st} under the constraint 5.7 million cycles and 1^{st} processor under the constraint 16.2 millon cycles) are not necessarily the same in terms of the issue unit width, the number of branch units, and so forth.

Figure 7 and Table 4 shows changes in the number of selected processors and overall area as the performance

```
SET A = a set of media applications;
SET M = a set of processors;
SET C = a set of cache configurations;
SET E = Construct_Run_Time_Table( A, M, C );
SET P(A) = power set of A − {};
SET T = a set of timing constraints;
for each timing constraint t ∈ T
  SET S = Best_Processors(P(A), E, t);
  SET P = Branch_and_Bound(S, t);
endfor;
```

```
SET Construct_Run_Time_Table(A, M, C)
  SET E;
  for each a ∈ A
    for each m ∈ M
      generate an executable;
      for each c ∈ C
        e_ijk = run time of a_ij with k cache;
      endfor;
    endfor;
  endfor;
  return E;
```

```
SET Best_Processors(P(A), E, t)
  SET S;
  for each element s ∈ P(A)
    find the minimum area processor that can run
      all tasks in s in t
    include the processor to S;
  return S;
```

Figure 5. Synthesis of hypermedia processors

```
SET Branch_and_Bound(S, t)
  STACK stack = EMPTY;
  NODE best_node;
  NODE node = ROOT;   COST best = a big value;
  Generate_New_Nodes(S, t, stack, best, node);
  while( s is not empty )
    node = take a node out of stack;
    if( Complete(node) )
      best = Cost( node );
      best_node = node;
    else
      Generate_new_nodes(S, t, stack, best, node);
    endif;
  endwhile;
  return best_node;
```

```
Generate_New_Nodes(S, t, stack, best, node)
  for each s ∈ S
    if( No_Overlap(node, s) ) &&
     ( Performance(node, s) ≤ t ) &&
     ( Cost(node, s) ≤ best )
      insert s into stack;
    endif;
  endfor;
```

Figure 6. Synthesis of hypermedia processors (Continued)

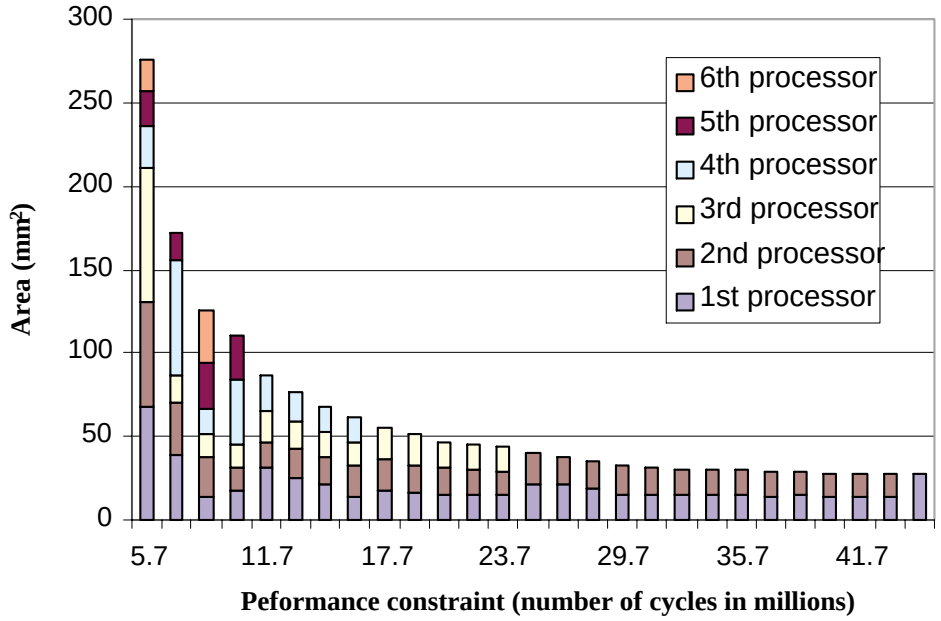

Figure 7. Minimum area configurations for a range of cycle time constrains

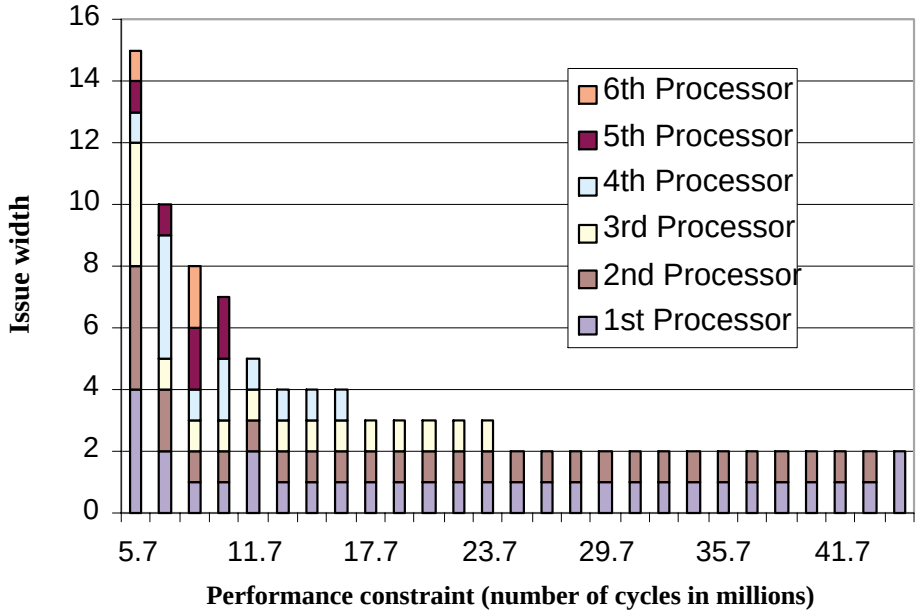

Figure 8. Issue widths of minimum area configurations for a range of cycle time constrains

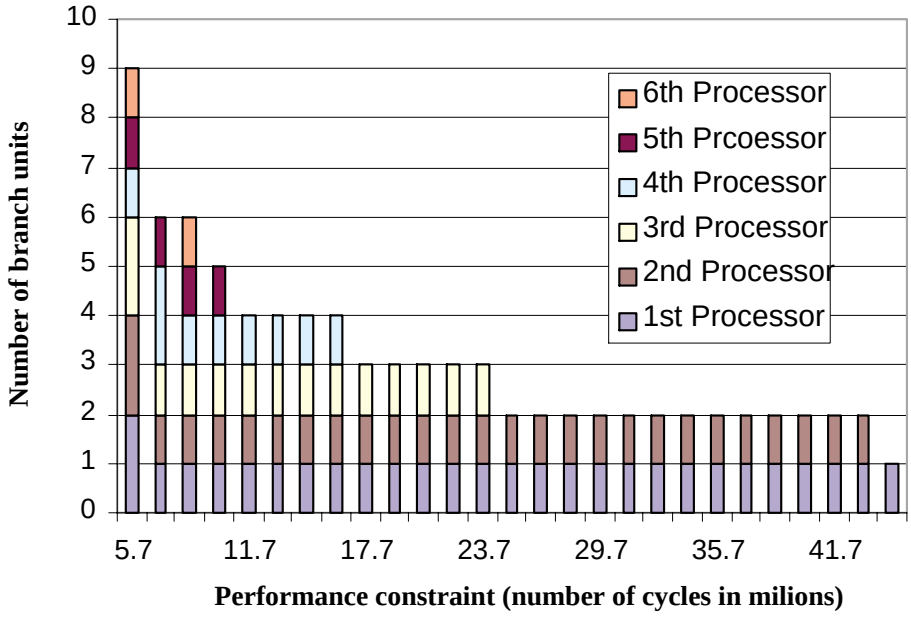

Figure 9. Branch widths of minimum area configurations for a range of cycle time constrains

constraint loosens. Looser timing results in reduction of the total number of processors used and also the total area of the processor. For a lower performance requirement more applications can be fit onto single small processor rather than separate multi-issue processors. This phenomenon can be also be seen on Figure 8 which shows the issue width of processors as the performance constraint varies. Single issue processors are dominant in the performance constraint range from 1.47×10^7 to 4.47×10^7 number of cycles. The reason is that single issue machines give the most performance per area than wider issue machines. The available ILP in the applications does not compensate the increase of the area of wider issue units. Therefore, wide issue machines only appear to be viable choices between 5.7×10^6 and 1.17×10^7 where speed is more critical.

The size of branch units also shows the similar characteristics. Figure 9 shows that only the extreme case where 5.70×10^6 cycles are available would require 2 branch units on a single processor. The main reason for this result is that the higher number of branch units only appear in wide issue machines and for those machines, the available ILP of the applications do not give the equal amount of speed up.

Figure 10 and 11 show the total size of instruction cache (I-cache) and data cache (D-cache) as the performance constraint varies. It seems that they are varying randomly. However, the general trend is that the cache requirement goes down as we loosen the constraint. Examining the results further reveals that all the bumps where the total size goes from local minimum to local maximum occur exactly when the number of processors is reduced. The area of a processor saved by reducing the issue width is transfered to a larger cache area so that the applications can be run faster and fit onto smaller number of processors. Table 5 summarizes instruction and data cache sizes

Cycles	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}	Sum
5.7	68.2657	61.8987	80.9638	25.0326	21.0357	19.0315	276.228
7.2	38.6433	31.4651	17.0273	68.2657	17.0273	-	172.4287
8.7	14.0115	23.0284	14.5148	14.5148	27.866	31.4651	125.4006
10.2	17.5236	14.0115	14.0115	38.6433	26.0623	-	110.2522
11.7	31.4651	15.5194	18.0268	21.0357	-	-	86.047
13.2	25.0326	18.0268	16.0226	17.0273	-	-	76.1093
14.7	21.0357	17.0273	14.5148	15.5194	-	-	68.0972
16.2	14.0115	18.0268	15.018	15.018	-	-	62.0743
17.7	17.5236	19.0315	19.0315	-	-	-	55.5866
19.2	16.0226	17.0273	18.0268	-	-	-	51.0767
20.7	15.018	16.0226	16.0226	-	-	-	47.0632
22.2	14.5148	15.5194	15.018	-	-	-	45.0522
23.7	14.5148	14.0115	15.018	-	-	-	43.5443
25.2	21.0357	19.0315	-	-	-	-	40.0672
26.7	21.0357	16.0226	-	-	-	-	37.0583
28.2	19.0315	16.0226	-	-	-	-	35.0541
29.7	15.018	17.0273	-	-	-	-	32.0453
31.2	15.018	16.0226	-	-	-	-	31.0406
32.7	14.5148	15.5194	-	-	-	-	30.0342
34.2	14.5148	15.018	-	-	-	-	29.5328
35.7	15.5194	14.0115	-	-	-	-	29.5309
37.2	14.0115	14.5148	-	-	-	-	28.5263
38.7	14.5148	14.0115	-	-	-	-	28.5263
40.2	14.0115	14.0115	-	-	-	-	28.023
41.7	14.0115	14.0115	-	-	-	-	28.023
43.2	14.0115	14.0115	-	-	-	-	28.023
44.7	27.866	-	-	-	-	-	27.866

Table 4. Required area (mm^2) to meet performance constraints

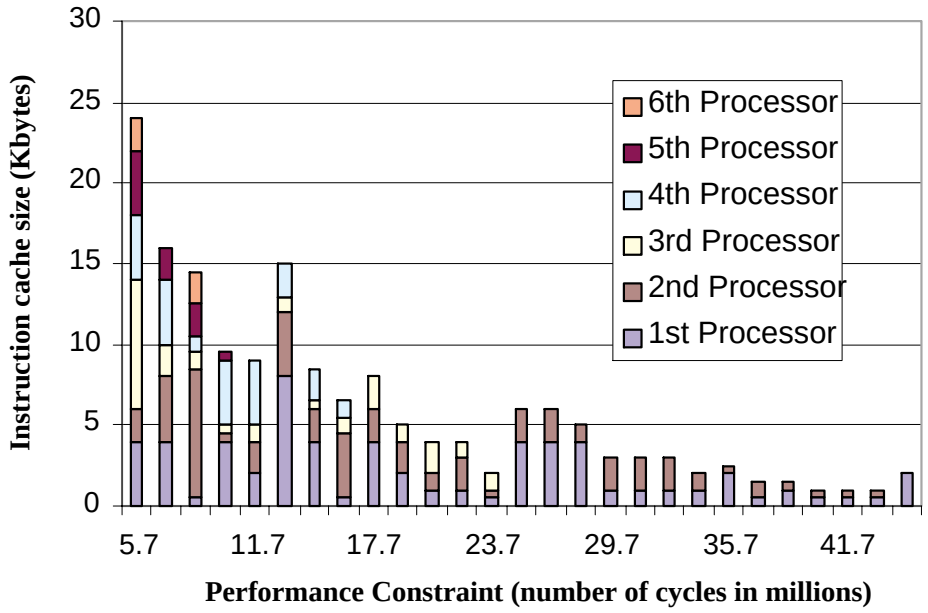

Figure 10. I-cache size of minimum area configurations for a range of cycle time constrains

required to meet a variety of cycle time constraints.

Figure 12 and 13 show the required cache amounts for the encoder group and decoder group (see Figure 1 for the explanation of the terms), respectively. We see that the D-cache size is roughly the same for encoding and decoding since the working set and the temporal locality is small. This is not the case for I-cache, since the encoding applications are more computationally intensive and have bigger inner loops. For example, *mpeg2enc* requires 50% more cycles to encode the same number of frames than *mpeg2dec* to decode on same processor configuration. Thus, higher I-cache and area usage is seen on encoding applications than decoding at the same performance constraint. Figure 14 shows the area used for encoding, decoding and the combined sets. Encoding set require slightly more area than the decoding set. Since the combined set includes all the applications and have more configuration combination to choose from, the total area used is less than the sum of the area of separated encoding and decoding sets.

7. Conclusion

A distributed hypermedia system that supports collaboration is an emerging tool for creation, discovery, management and delivery of information. Distributed hypermedia systems are becoming increasingly desired and practical as other areas of information technologies advance.

The advances in compiler technology and architectural enhancements found in commercial DSPs motivated this

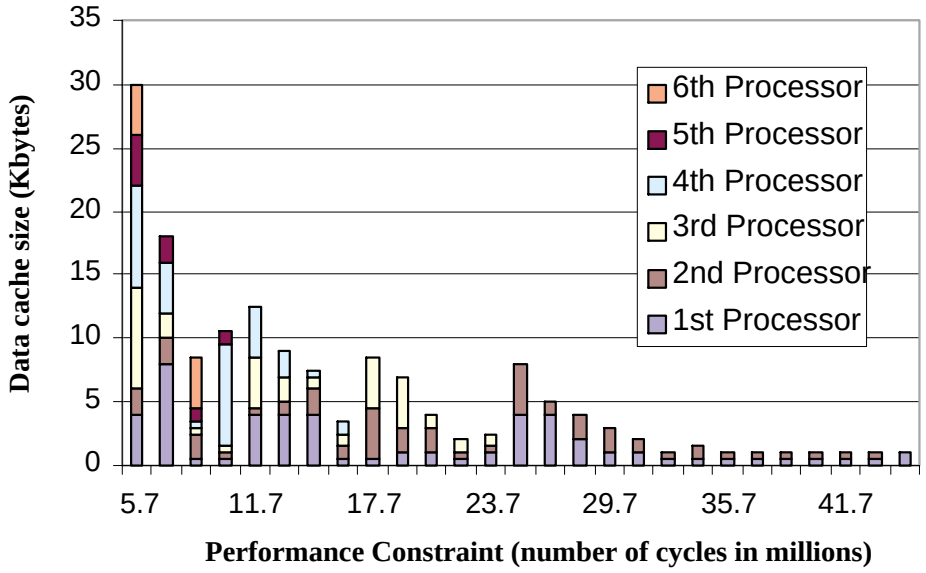

Figure 11. D-cache size of minimum area configurations for a range of cycle time constrains

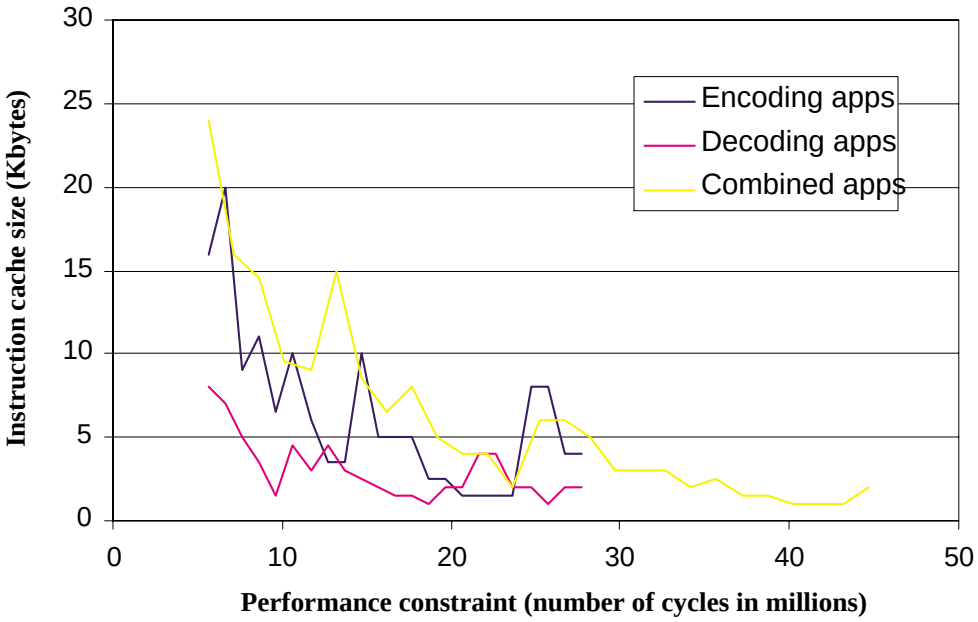

Figure 12. Required cache amount of minimum area configurations for encoding tasks

Cycles	1^{st}		2^{nd}		3^{rd}		4^{th}		5^{th}		6^{th}		Sum	
	I	D	I	D	I	D	I	D	I	D	I	D	I	D
5.7	4	4	2	2	8	8	4	8	4	4	2	4	24	30
7.2	4	8	4	2	2	2	4	4	2	2	-	-	16	18
8.7	0.5	0.5	8	2	1	0.5	1	0.5	2	1	2	4	14.5	8.5
10.2	4	0.5	0.5	0.5	0.5	0.5	4	8	0.5	1	-	-	9.5	10.5
11.7	2	4	2	0.5	1	4	4	4	-	-	-	-	9	12.5
13.2	8	4	4	1	1	2	2	2	-	-	-	-	15	9
14.7	4	4	2	2	0.5	1	2	0.5	-	-	-	-	8.5	7.5
16.2	0.5	0.5	4	1	1	1	1	1	-	-	-	-	6.5	3.5
17.7	4	0.5	2	4	2	4	-	-	-	-	-	-	8	8.5
19.2	2	1	2	2	1	4	-	-	-	-	-	-	5	7
20.7	1	1	1	2	2	1	-	-	-	-	-	-	4	4
22.2	1	0.5	2	0.5	1	1	-	-	-	-	-	-	4	2
23.7	0.5	1	0.5	0.5	1	1	-	-	-	-	-	-	2	2.5
25.2	4	4	2	4	-	-	-	-	-	-	-	-	6	8
26.7	4	4	2	1	-	-	-	-	-	-	-	-	6	5
28.2	4	2	1	2	-	-	-	-	-	-	-	-	5	4
29.7	1	1	2	2	-	-	-	-	-	-	-	-	3	3
31.2	1	1	2	1	-	-	-	-	-	-	-	-	3	2
32.7	1	0.5	2	0.5	-	-	-	-	-	-	-	-	3	1
34.2	1	0.5	1	1	-	-	-	-	-	-	-	-	2	1.5
35.7	2	0.5	0.5	0.5	-	-	-	-	-	-	-	-	2.5	1
37.2	0.5	0.5	1	0.5	-	-	-	-	-	-	-	-	1.5	1
38.7	1	0.5	0.5	0.5	-	-	-	-	-	-	-	-	1.5	1
40.2	0.5	0.5	0.5	0.5	-	-	-	-	-	-	-	-	1	1
41.7	0.5	0.5	0.5	0.5	-	-	-	-	-	-	-	-	1	1
43.2	0.5	0.5	0.5	0.5	-	-	-	-	-	-	-	-	1	1
44.7	2	1	-	-	-	-	-	-	-	-	-	-	2	1

Table 5. Cache variation

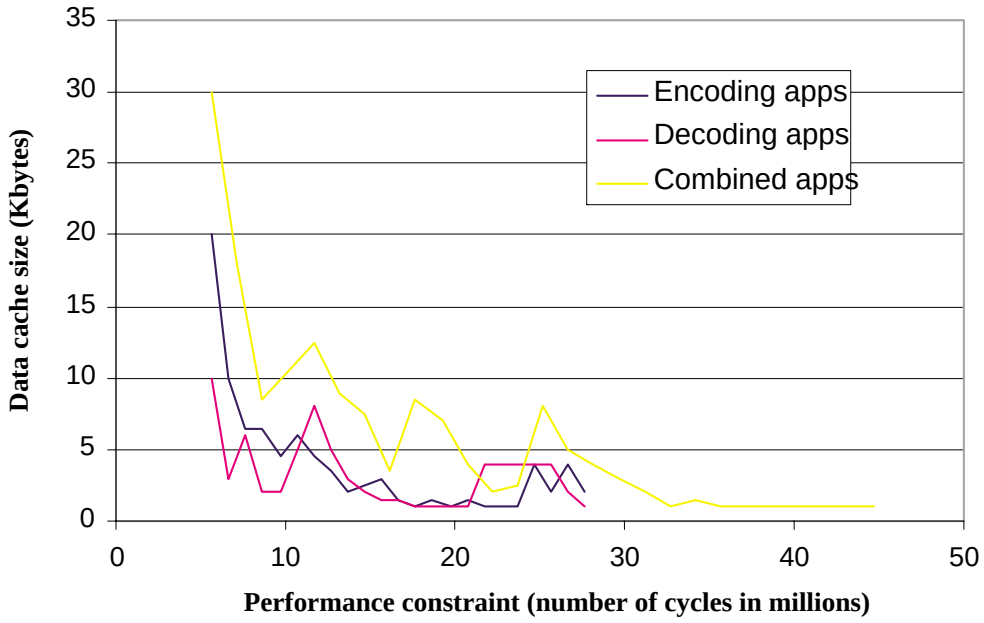

Figure 13. Required cache amount of minimum area configurations for decoding tasks

work. Thus, the approach presented in this paper makes use of a state-of-the-art ILP compiler, simulators, the notion of multiple-instruction-issue processors and an efficient algorithm to combine processors optimized for an individual task or a set of tasks to minimize the area of resulting hypermedia processor. Although the selection problem is shown to be NP-complete, we found that the optimal solutions can be obtained in reasonable run time for practically sized problems.

Using the developed framework we conduct an extensive exploration of area optimal system design space exploration for a hypermedia application. We found that there is enough ILP in the typical media and communication applications to achieve highly concurrent execution when throughput requirements are high. On the other hand, when throughput requirements are low, there is no need to use multiple-instruction-issue processors as they provide no desirable benefits. This phenomenon is due to the fact that increased area does not produce enough performance gains to justify the use of multiple-instruction-issue processors when throughput requirements are low.

The framework introduced in this paper is valuable in making early design decisions such as architectural configuration trade-offs including the cache and issue width trade-off under area constraint, and the number of branch units and issue width.

References

[1] G. Araujo, A. Sudarsanam, and S. Malik. Instruction set design and optimizations for address computation in

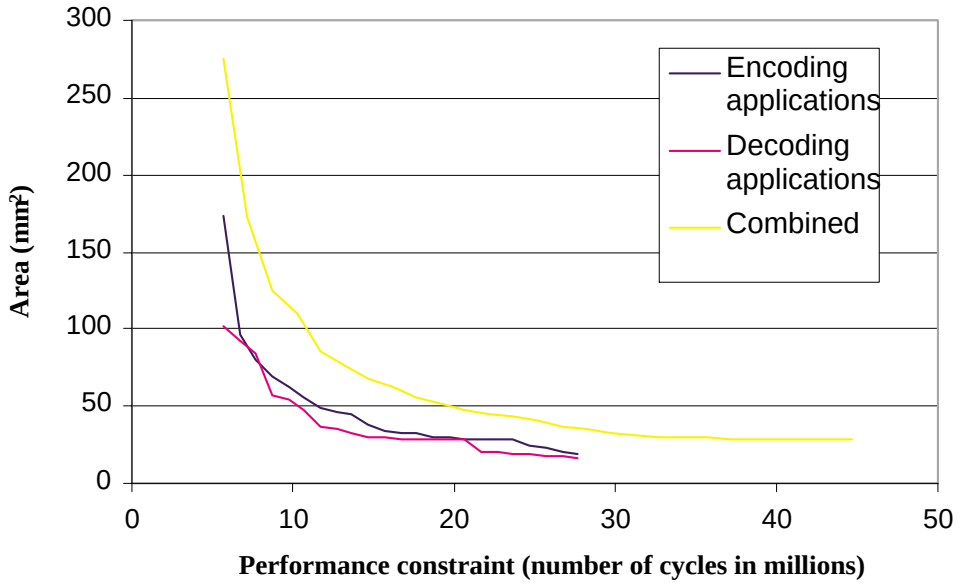

Figure 14. Required cache amount of minimum area configurations for all tasks

DSP architectures. In *Proceedings of International Symposium on System Synthesis*, pages 102–107, 1996.

[2] D. C. Argyres. Performance and cost analysis of the execution stage of superscalar microprocessors. Master's thesis, Department of Computer Science, University of Illinois, Urbana IL, May 1995.

[3] S. Banerjia, W. A. Havanki, and T. M. Conte. Treegion scheduling for highly parallel processors. In *Euro-Par*, pages 1074–1078, Passau, Germany, 1997.

[4] T. Berners-Lee, R. Cailliau, J. Groff, and B. Pollerman. World wide web: The information universe. *Internet Research: Electronic Networking Research Application and Policy*, 1(1):10–24, 1992.

[5] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world wide web. *Communications of ACM*, 37(8):76–82, 1994.

[6] V. Bush. As we may think. *The Atlantic Monthly*, July 1945.

[7] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. m. W. Hwu. IMPACT: An architectural framework for multiple-instruction-issue processors. In *International Symposium on Computer Architecture*, 1991.

[8] R. P. Colwell, R. P. Nix, J. J. O'Donnell, D. B. Papworth, and P. K. Rodman. A VLIW architecture for a trace scheduling compiler. In *Proceedings of ASPLOS-II*, pages 180–192, 1982.

[9] T. Conte and W. Mangione-Smith. Determining cost-effective multiple issue processor designs. In *International Conference on Computer Design*, 1993.

[10] T. M. Conte, K. N. P. Menezes, and S. W. Sathaye. A technique to determine power-efficient, high-performance superscalar processors. In *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, volume 1, pages 324–333, 1995.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[12] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computing*, C-30:478–490, 1981.

[13] J. A. Fisher, P. Faraboschi, and G. Desoli. Custom-fit processors: Letting applications define architectures. In *International Symposium on Microarchitectures*, Paris, France, 1996.

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

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, 1979.

[16] G. Goossens, J. Van Praet, D. Lanneer, W. Geurts, et al. Embedded software in real-time signal processing systems: design technologies. *Proceedings of the IEEE*, 85(3):436–454, March 1997.

[17] K. Gronbaek and R. Trigg. Design issues for a Dexter-based hypermedia system. *Communications of ACM*, 37(2):41–49, February 1994.

[18] F. Halasz. Reflections on note-cards: Seven issues for the next generation of hypermedia systems. *Communications of ACM*, 31(7):836–852, July 1988.

[19] C. Hansen. MicroUnity's MediaProcessor architecture. *IEEE Micro*, 17:34–41, 1997.

[20] L. Hardman, D. Bulterman, and G. van Rossum. The Amsterdam hypermedia model: Adding time and context to the dexter model. *Communications of ACM*, 37(2):50–62, February 1994.

[21] P. Y. Hsu. Highly concurrent scalar processing. Technical Report CSG-49, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1986.

[22] P. Kalapathy. Hardware-software interactions on MPACT. *IEEE Micro*, 17:20–26, 1997.

[23] F. Kappe, H. Maurer, and N. Sherboken. Hyper-G: A universal hypermedia systems. *Journal of Educational Multimedia and Hypermedia*, 2(1):39–66, 1993.

[24] M. Kessler. Distributed hypermedia. In *Proceedings of Southcon '95*, pages 190–195, 1995.

[25] K. Kim, R. Karri, and M. Potkonjak. Heterogeneous built-in resiliency of application specific programmable processors. In *International Conference on Computer-Aided Design*, pages 406–411, 1996.

[26] F. Kretz and F. Colaitis. Standardizing hypermedia information objects. *IEEE Communications Magazine*, pages 60–70, May 1992.

[27] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitectures*, 1997.

[28] R.B. Lee and M.D. Smith. Media processing: A new design target. *IEEE Micro*, 17:6–9, 1997.

[29] J. Leggett and J. Schnase. Viewing dexter with open eyes. *Communications of ACM*, 37(2):77–86, February 1994.

[30] R. Leupers and P. Marwedel. Retargetable generation of code selectors from HDL processor models. In *European Design and Test Conference*, pages 140–144, 1997.

[31] S. Liao, S. Devadas, K. Keutzer, and S. Tjiang. Instruction selection using binate covering for code size optimization. In *International Conference on Computer-Aided Design*, pages 393–399, 1995.

[32] C. Liem, T. May, and P. Paulin. Instruction-set matching and selection for DSP and ASIP code generation. In *The European Design and Test Conference*, pages 31–37, 1994.

[33] W. m. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm, and D. M. Lavery. The superblock: An effective technique for VLIW and superscalar compilation. *Journal of Supercomputing*, 1993.

[34] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann. Effective compiler support for predicated execution using the Hyperblock. In *International Symposium on Microarchitecture*, 1992.

[35] P. Marwedel. Processor-core based design and test. In *Asia and South Pacific Design Automation Conference*, pages 499–502, 1997.

[36] J. Montanaro et al. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 31(11):1703–1714, November 1996.

[37] T. Nelson. A filestructure for the complex, the changing and the intermediate. *ACM 20th National Conference*, 1965.

[38] P. G. Paulin, C. Liem, M. Cornero, F. Nacabal, et al. Embedded software in real-time signal processing systems: application and architecture trends. *Proceedings of the IEEE*, 85(3):419–435, March 1997.

[39] A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Micro*, 16(4):42–50, August 1996.

[40] M. Potkonjak and W. H. Wolf. Cost optimization in ASIC implementation of periodic hard real-time systems using behavioral synthesis techniques. In *ICCAD95*, pages 446–451. International Conference on Computer-Aided Design, 1995.

[41] M. Potkonjak and W. H. Wolf. Heuristic techniques for synthesis of hard real-time DSP application specific systems. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, pages 1240–3, 1996.

[42] A. Sudarsanam and S. Malik. Memory bank and register allocation in software synthesis for ASIPs. In *International Conference on Computer-Aided Design*, pages 388–392, 1995.

[43] J. Turley and H. Hakkarainen. TI's new 'C6x DSP screams at 1,600 MIPS. *The Microprocessor Report*, 11:14–17, 1997.

[44] W. Zhao and C.A. Papachristou. An evolution programming approach on multiple behaviors for the design of application specific programmable processors. In *European Design and Test Conference ED&TC 96*, pages 144–150, 1996.