Transactions Briefs

Exploring the Diversity of Multimedia Systems

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

***Abstract*—We evaluate the validity of the fundamental assumption behind application-specific programmable processors: that applications differ from each other in key parameters which are exploitable, such as the available instruction-level parallelism (ILP), demand on various hardware resources, and the desired mix of function units. Following the tradition of the CAD community, we develop an accurate chip area estimate and a set of aggressive hardware optimization algorithms. We follow the tradition of the architecture community by using comprehensive real-life benchmarks and production quality tools. This combination enables us to build a unique framework for system-level synthesis and to gain valuable insights about design and use of application-specific programmable processors for modern applications. We explore the application-specific programmable processor (ASSP) design space to understand the relationship between performance and area. The architecture model we used is the Hewlett Packard PA-RISC [1] with single level caches. The system, including all memory and bus latencies, is simulated and no other specialized ALU or memory structures are being used. The experimental results reveal a number of important characteristics of the ASSP design space. For example, we found that in most cases a single programmable architecture performs similarly to a set of architectures that are tuned to individual application. A notable exception is highly cost sensitive designs, which we observe need a small number of specialized architectures that require smaller areas. Also, it is clear that there is enough parallelism in the typical media and communication applications to justify use of high number of function units. We found that the framework introduced in this paper can be very valuable in making early design decisions such as area and architectural configuration tradeoff, cache and issue width tradeoff under area constraint, and the number of branch units and issue width.**

***Index Terms*—Application-specific programmable processor, instruction level parallelism, mediabench, mediaprocessor, system-level synthesis.**

I. Introduction

It has been predicted that the "micro-brain boom" (*sic*) will greatly increase demand for application-specific microprocessors for media applications [2]. Sales of handheld computers and personal digital assistants grew almost sixfold from 1994's total, to 5.6 million units in 1999. The market for programmable DSP chips increased 20% in 1998 to the $3.9 billion level. The new DSP markets, which are beginning to emerge, including digital cameras, satellite phones, smart antennas, voice over IP, ac motor control, and even digital TV, is forecast to grow at a 33% compound rate to the $13.4 billion level in 2002 [3].

This market growth coincides with an interesting technological advance that will change both the semiconductor business and microprocessor design. Since 1992, microprocessors account for 23% of total semiconductor sales. In 1998, these chips accounted for 30% of total value of the semiconductor production. The increasing share of microprocessors in semiconductor market is due to a new phase of silicon integration enabled by deep submicron fabrication technology.

Manuscript received July 16, 1998; revised October 22, 1999.

J. Kin and W.-H. Mangione-Smith are with the Electrical Engineering Department, University of California, Los Angeles, CA 90095 USA (e-mail: johnsonk@icsl.ucla.edu; billms@icsl.ucla.edu).

C. Lee and M. Potkonjak are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: leec@cs.ucla.edu; miodrag@cs.ucla.edu).

Publisher Item Identifier S 1063-8210(01)00703-X.

For example, SA-1100 from Intel [4] incorporates many functions such as a memory controller, color LCD driver, PCMCIA interface, IrDA and USB communication channels, and extensive power management into a single chip along with its core logic, previously available only through "glue logic" chips. One implication of this technology is that almost all semiconductor manufacturers are entering the microprocessor business.

As a consequence of this trend, the market will be more crowded and competitive in spite of increasing demand. This pressure will force manufacturers to focus on microprocessors that are cheaper and more aggressively optimized for specific applications. A challenge to microprocessor designers will be to design a microprocessor that executes a targeted application very well yet can achieve economy-of-scale. For example, video-game players such as PlayStation from Sony and Nintendos64 from Nintendo need to employ ever-more powerful processors for the application and yet remain cheap enough to sell for under $300.

On the technical side, recent advances in compiler technology and microprocessor architecture 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. Key ILP compiler technologies, such as trace scheduling [4], superblock scheduling [5], treegion-scheduling [6], hyperblock scheduling [8], and software pipelining [9] are in the process of migrating from research labs to product groups.

At the same time, a number of new microprocessor architectures have been introduced. These designs present 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. One of the best examples that has these features is TMS320C6X from Texas Instruments [9]. Although TI considers the TMS320C6X to be a DSP, the architecture is almost a copy of the Multiflow Trace [10]. Multi-gauge arithmetic (or variable-width SIMD) is found in the family of MPACT architectures from Chromatic [11] and the designs from MicroUnity [12]. Most of the multimedia extensions of programmable processors also adopt this architectural enhancement [14].

The arrival of production quality ILP compilers and commercial DSPs with VLIW and SIMD architectures stimulated the idea of custom-fit processors [15]. The premise of such an approach is that applications differ from each other in exploitable measures, for example the available ILP, demand on various hardware components (e.g., cache memory units, register files) and the number of function units. The presumption is that a microprocessor can be designed by adding hardware components tailored to a specific application so that it can execute the single application extremely well. Of course, an obvious drawback of this approach is that it provides no guarantee that other applications will run as well as the targeted application. While the current microprocessors for media applications (mediaprocessors) are claimed to target general applications in a domain [13], a custom-fit processor targets a single application (although they remain programmable).

We report on a method of system-level synthesis of single or multiple application programmable processors. We use a benchmark suite consisting of complete applications written in a high level language [16]. We use the IMPACT tool suit [18] to collect performance measurements of benchmarks on various machine configurations. The IMPACT

TABLE I
MACHINE 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)

Configuration	Issue unit	IALU	Branch unit	Memory unit	Other	Cache memory	Total
(1, 1, 1, 1, 0.5, 0.5)	0.74	3.67	6.13	5.52	13.98	1.53	31.56
(2, 2, 1, 2, 1, 1)	2.96	7.33	6.13	11.03	13.98	2.54	43.95
(4, 4, 1, 4, 2, 2)	11.76	14.66	6.13	22.07	13.98	4.55	73.15
(8, 8, 1, 8, 4, 4)	47.04	29.32	6.13	44.14	13.98	8.56	149.17
(4, 4, 2, 4, 8, 8)	11.76	14.66	12.26	22.07	13.98	16.55	91.28
(8, 8, 2, 8, 4, 4)	47.04	29.32	12.26	44.14	13.98	8.56	155.30
(8, 8, 4, 8, 8, 8)	47.04	29.32	24.52	44.14	13.98	16.55	175.55

TABLE II
APPLICATIONS USED IN THE EXPERIMENT. DYNAMIC INSTRUCTION COUNTS WERE MEASURED ON A SPARC

No.	Benchmark	Dynamic Instructions	Source	Description
1	JPEG encoder	13.9 million	Independent JPEG Group	JPEG image encoding/decoding
2	JPEG decoder	3.8 million		
3	MPEG encoder	1121.3 million	MPEG Simulation Group	MPEG-2 movie encoding/decoding
4	MPEG decoder	175.5 million		
5	GSM encoder	184.2 million	Technische Universitat, Berlin	European wireless voice coding standard
6	GSM decoder	73 million		
7	G.721 encoder	274.1 million	Sun Microsystems, Inc.	CCITT voice coding standard
8	G.721 decoder	511.7 million		
9	PGP encryption	169.9 million	Massachusetts Institute of Technology	Encryption/ decryption
10	PGP decryption	155.3 million		
11	Pegwit encryption	34.0 million	George Barwood	Encryption/ decryption
12	Pegwit decryption	18.5 million		
13	Mipmap	47.6 million	University of Wisconsin	3-D rendering examples using MESA graphics library
14	OS-demo	9.0 million		
15	Texgen	83.8 million		
16	Rasta	24.4 million	ICSI at UC Berkeley	Voice recognition
17	EPIC encoder	50.3 million	University of Pennsylvania	Wavelet image encoding/decoding
18	EPIC decoder	7.2 million		
19	ADPCM encoder	6.8 million	Jack Jansen	Speech compression and decompress
20	ADPCM decoder	5.9 million		

C compiler is a retargetable compiler with code optimization components supporting multiple-instruction-issue processors. The target machine is described using the high-level machine description language. A high-level machine description supplied by a user is compiled by the IMPACT machine description language compiler. IMPACT provides cycle-level simulation tools.

This paper is organized as follows. The next section briefly surveys related works and summarizes the contributions of this work. Section III presents the background materials including machine model, benchmarks, experiment platform (such as tools), and an example set of results obtained using the tools. Our approach in this project is explained in Section IV in detail. Section V formulates the search problem defined in the previous section in formal terms. The solution space exploration strategy and algorithm is described in Section VI. Extensive experimental results are reported in Section VII. Finally, Section VIII draws conclusions.

II. RELATED WORKS AND OUR CONTRIBUTIONS

The work on synthesis and evaluation of application-specific programmable processors has been conducted independently in two research communities, computer-aided design and architecture. There is, however, a strong converging trend of the two areas due to recent technological advances and application trends. In this section we survey the related works in these two fields.

There have 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 have been conducted by Goosens [18], Paulin [19], and Marwedel [20]. In particular, a great deal of effort has been made in combining retargetable compilation technologies and design of instruction sets [22]–[26]. Several research groups have published results on the topic of selecting and designing instruction set and processor architecture for a particular application domains [27], [28].

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 [29] presented one of the first efforts that focused on large application codes (i.e., SPEC) written in a high-level language. While they had a similar goal to ours, i.e., evaluating performance efficiency by including hardware cost, their evaluation approach was substantially different. Conte *et al.* [29] 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

TABLE III
DATA SET USED IN THE EXPERIMENT

No.	Benchmark	File size	Format	Description
1	JPEG encoding	101,484 bytes	PPM	an uncompressed bit map
2	JPEG decoding	5,756 bytes	JPG	a JPEG compressed
3	MPEG encoding	506,880 bytes	YUV components	4 frames
4	MPEG decoding	34,906 bytes	MPEG-2	MPEG-2 archive (http://www.mpeg2.dec/)
5	GSM encoding	295,040 bytes	16 bit PCM	Clinton speech
6	GSM decoding	30,426 bytes	GSM encoded	Clinton speech
7	G.721 encoding	295,040 bytes	16 bit PCM	Clinton speech
8	G.721 decoding	147,520 bytes	G.721 encoded	Clinton speech
9	PGP encryption	91,503 bytes	plain ASCII	text file
10	PGP decryption	20,163 bytes	PGP encrypted	text file
11	Pegwit encryption	91,503 bytes	plain ASCII	text file
12	Pegwit decryption	91,537 bytes	Pegwit encrypted	text file
13	Mipmap	N/A		mipmap texture mapping example
14	OS-demo	N/A		3-D rendering pipeline example
15	Texgen	N/A		texture mapped Utah teapot
16	Rasta	17,024 bytes	SHPERE format	package with SPHERE
17	EPIC encoding	65,595 bytes	PGM	Gray scale 256 x 256
18	EPIC decoding	7,432 bytes	EPIC encoded	Gray scale 256 x 256
19	ADPCM encoding	295,040 bytes	16 bit PCM	Clinton speech
20	ADPCM encoding	73,760 bytes	ADPCM encoded	Clinton speech

TABLE IV
RUN-TIME CHARACTERISTICS MEASURED ON HPPA-7100 USING IMPACT SUITES

No.	Benchmark	IPC	BTB	I hit	D read	D write	D hit	Bus	Branch	IALU
1	JPEG encoder	0.89	91.96	99.97	98.30	83.42	94.52	3.14	15.25	53.74
2	JPEG decoder	0.95	91.67	99.97	98.69	53.10	84.25	5.48	5.06	66.83
3	MPEG encoder	0.87	80.38	99.99	97.30	81.91	96.33	5.35	19.11	41.10
4	MPEG decoder	0.91	89.83	99.96	99.41	87.13	95.81	2.01	11.73	39.59
5	GSM encoder	0.86	86.80	99.58	99.98	99.89	99.94	2.93	8.67	43.66
6	GSM decoder	0.92	97.15	99.97	99.99	99.75	99.90	0.27	15.82	67.63
7	G.721 encoder	0.93	89.74	100.00	100.00	99.50	99.85	0.03	20.23	55.91
8	G.721 decoder	0.94	91.90	100.00	100.00	99.99	100.00	0.01	20.64	56.01
9	PGP encryption	0.93	79.93	100.00	98.69	99.44	98.94	1.32	13.51	63.42
10	PGP decryption	0.92	79.53	99.99	96.93	97.30	97.05	2.89	13.21	63.16
11	Pegwit	0.71	95.77	99.07	98.12	79.32	88.72	11.54	14.24	21.62
12	Pegwit	0.61	86.15	97.02	95.06	77.78	86.42	23.64	12.36	19.27
13	Mipmap	0.74	99.24	98.23	98.55	82.98	91.89	13.85	11.71	24.92
14	OS-demo	0.82	94.82	99.86	96.21	45.58	80.47	7.64	13.64	46.43
15	Texgen	0.76	89.47	98.42	98.31	88.16	94.40	13.24	9.06	29.39
16	Rasta	0.75	91.28	99.77	94.99	81.46	90.45	10.82	12.31	20.64
17	EPIC encoder	0.87	95.55	100.00	97.44	42.32	90.48	3.93	16.20	37.15
18	EPIC decoder	0.70	92.03	100.00	93.51	44.88	71.72	11.51	13.36	32.63
19	ADPCM	0.89	82.67	100.00	99.98	0.40	85.72	0.92	27.26	55.11
20	ADPCM	0.92	83.49	100.00	99.99	0.20	71.46	1.93	23.20	62.39

machine for execution on superscalar implementations. Fisher *et al.* [15] studied the variability of application-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 among attractive architectures across a set of applications.

We adopt a methodology of system synthesis combining the key paradigms of both communities. Following the tradition of the CAD community, we develop an accurate area estimate and aggressive optimization algorithms. We follow the tradition of the architecture community by using comprehensive real-life benchmarks and production quality compilation and simulation tools. This combination enables us to build a unique framework of system-level synthesis and to gain valuable insights about design and use of application-specific programmable processors for modern applications.

Unlike previous works, we use a set of complete applications written in a high-level language as benchmarks. We incorporate the role of cache memory units in machine performance into the machine model, which is essential for producing meaningful results.

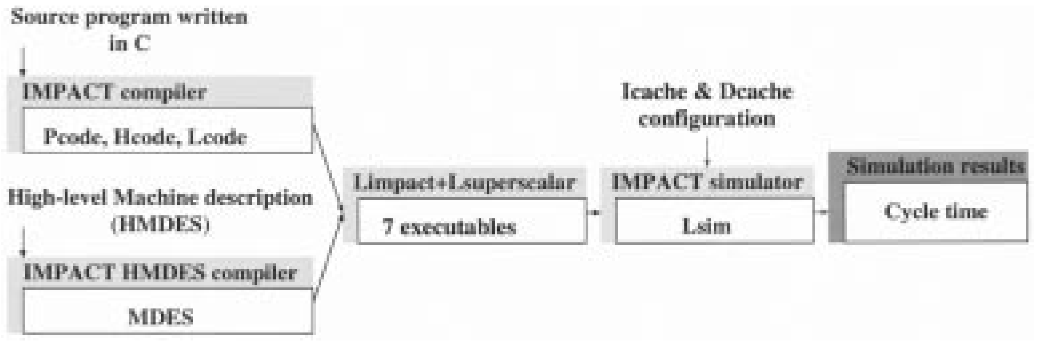


Fig. 1. Performance measurement flow using IMPACT tools.

TABLE V
AN EXAMPLE SET OF RESULTS

No.	Benchmark	m1	m2	m3	m4	m5	m6	m7
1	JPEG encoding	3.64594	2.82103	2.99120	3.84738	2.20971	2.63822	2.55203
3	JPEG decoding	4.23373	3.27552	2.98682	4.26518	2.30956	2.62019	2.71783
10	MPEG encoding	2.47749	1.42038	1.92456	2.56567	1.09295	1.37050	1.77085
9	MPEG decoding	1.93998	1.96280	1.60252	3.67821	3.44580	3.61907	3.35929
7	GSM encoding	2.25379	2.27273	2.29321	2.30589	1.99094	2.28716	2.28159
6	GSM decoding	2.93745	2.93340	2.97611	3.00388	1.67255	3.12961	3.14519
4	G.721 encoding	3.07744	2.02068	2.47258	3.37837	1.66267	2.02378	2.53573
2	G.721 decoding	3.11276	1.96270	2.36477	3.44207	1.67080	2.08472	2.48196
15	PGP encryption	3.14815	2.73464	2.97603	3.20877	2.46403	2.71223	2.93360
14	PGP decryption	3.28457	2.80132	2.93317	3.37517	2.50223	2.7617	2.84765
13	Pegwit encryption	2.78988	2.11644	2.17627	2.82412	1.98262	1.91087	2.02120
12	Pegwit decryption	2.82299	2.25366	2.23790	2.84193	2.03881	1.93049	2.03430
8	Mipmap	2.94320	2.90037	2.91227	2.78862	2.28356	2.75903	2.69301
11	OS-demo	2.42271	2.19599	2.28499	2.46821	2.01700	2.21351	2.30880
19	Texgen	2.40340	1.94311	2.04235	2.39706	1.76382	1.86625	1.95847
16	Rasta	2.26343	2012603	2.14634	2.27230	2.00360	2.08559	2.09685
5	EPIC encoding	2.06328	1.78115	1.69603	2.17036	1.85368	1.77016	1.72495
20	EPIC decoding	1.69781	1.73186	1.67798	1.76916	1.78700	1.73074	1.64389
17	ADPCM encoding	2.08063	2.10912	2.07475	2.30649	2.42662	2.38234	2.30367
18	ADPCM encoding	2.20944	2.22268	2.2268	2.41923	2.54383	2.51963	2.54963

We focus on the number of machine configurations that should be developed in order to maximize performance for all of the benchmarks given an area constraint. We understand that it is in the best interest of a processor designer to understand which architecture and how many functional units or cache size is best for one particular application. However, our first goal is to develop a framework for managers to understand how big the chips portfolio should be in one particular domain. It is not intended for a single designer to find his best application-specific system. The objective function of the optimizer is minimization of selected machine configurations, thereby maximizing the number of benchmarks that can be run on a processor as though it is optimized for each individual benchmark. In one extreme case, we end up with as many machine configurations as the number of individual benchmarks. On the other extreme, we need only one machine. Clearly, the most interesting solutions lie somewhere in the middle.

Power consumption evaluation and optimization is very often an important aspect in multimedia processors; however, it is beyond the scope of this paper. We have published a thorough investigation of power consumption using similar framework and tools in another paper [30].

III. PRELIMINARY DISCUSSION

In this section we discuss the experimental environment that has been adapted and developed for the investigation. First, we describe the machine model used to estimate the area of a machine configuration. The benchmark suite is introduced along with the characteristics of its components. Finally, we explain the experimental platform, including tools and their example outputs.

A. Machine Model

To estimate the cost of a machine configuration, we adopt a simple model developed by Argyres [31]. Given the area of the issue unit, the cost of any scalar machine configuration is a linear function of the numbers of branch, memory, and arithmetic units. A machine may include any number of each function unit. For a superscalar machine, the issue unit area cannot be estimated using a simple linear model since it requires more complex logic for runtime code scheduling. We assume that the issue unit area will take $O(n^2)$ space since the complexity of dependency checking algorithm is $O(n^2)$. When a VLIW machine is considered, the issue unit area is known to be of complexity $O(n)$ or

TABLE VI
CONFIGURATIONS OF THE MACHINES USED IN TABLE V

	m1	m2	m3	m4	m5	m6	M7
Area	94.58	155.45	155.45	100.71	159.19	156.82	159.19
Issue width	4	8	8	4	8	8	8
No. ALU	4	8	8	4	8	8	8
No. branch unit	1	1	1	2	2	2	2
No. memory unit	4	8	8	4	8	8	8
Icache (KB)	8	2	4	8	1	2	4
Dcache (KB)	8	4	2	8	4	2	1

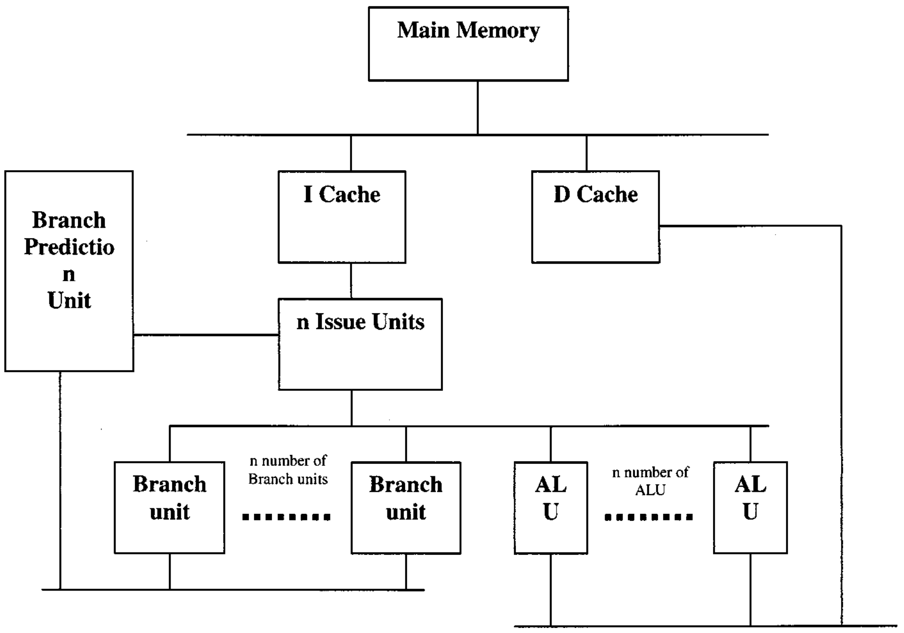


Fig. 2. System model being simulated.

sublinear. The cost function of arbitrarily configured superscalar machines is given by

$$\begin{aligned}\text{Area} = &(A_B - A_E - A_C - A_U) + A_u \\ &+ n_b A_b + n_m A_m + n_i A_i + A_{dc} + A_{\text{ic}} \qquad (1)\end{aligned}$$

where

A_B	area of a baseline machine;
A_E	area of a baseline machine execution unit;
A_C	baseline machine cache area;
A_U	baseline machine issue unit area;
A_u	issue unit area;
n_b	number of branch units;
A_b	area of branch units;
n_m	number of memory units;
A_m	area of memory units;
n_i	number of ALU;
A_i	area of an ALU;
A_{dc}	data cache area;
A_{ic}	instruction cache area.

The baseline architecture chosen for the analysis is the PowerPC 604 [32], a four-issue processor. The 604 has two simple integer ALUs and one complex integer ALU, one floating-point unit, one branch unit, and one memory unit. We assume that machine configurations that have an issue unit smaller than the baseline machine have at least one complex integer ALU. The area of the complex integer unit is assumed to be half of the baseline integer unit (two simple integer units and one complex integer unit). The area of issue unit is scaled based on the area complexity $(O(n^2))$. We did not include floating-point units in any machine configurations because the benchmarks we used have mostly integer (or fixed-point) operations. Finally, we scaled the area for 0.35μm technology rather than the original 0.5μ technology used

```
$define ISSUE 1
$define IALU 1
$define FALU 1
$define FMUL 1
$define FDIV 1
$define BRANCH 1
$define MODEL superscalar
# Enumerate resources
(Resources declaration
    slot[0..$ISSUE$]
    ialu_0[0..$IALU$]
    falu_0[0..$FALU$]
    fmul_0[0..$FMUL$]
    fdiv_0[0..$FDIV$]
    mem_0[0..$ISSUE$]
    branch[0..$BRANCH$]
end)
```

Fig. 3. An example high-level machine description (HMDES).

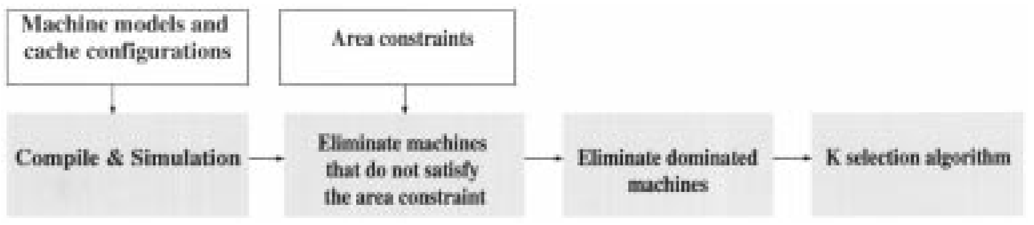

Machine models and cache configurations
Area constraints
Compile & Simulation
Eliminate machines that do not satisfy the area constraint
Eliminate dominated machines
K selection algorithm

Fig. 4. Global design flow.

by Argyres. A set of example machine configurations and their respective area estimates are shown in Table I.

B. Benchmarks

The set of benchmarks used in this work is composed of complete applications which are publically available and coded in a high-level language. The collection is composed of 21 applications culled from available image processing, communications, cryptography, and DSP applications. Brief summaries of benchmarks and data used are shown in Tables II and III, respectively. More detailed descriptions of the benchmarks can be found in a previous publication [16].

As discussed in the Introduction, the idea that a programmable processor can be tuned to a target application is based on the assumption that applications differ from each other in exploitable features. As an illustration, Table IV shows measured characteristics of the benchmarks used in the experiment. Note that the combination of the instructions per cycle (IPC), bus utilization, branch issue, and ALU issue exhibit distinctive characteristics for each benchmark. Although the target was a single-issue machine, we found that there was strong evidence that performance tuning for an individual application could be beneficial. Note that in order to reduce the effect of memory operations on other measurements, the target machine has 32 KB instruction cache and 32 KB data cache, resulting in high cache hit rates.

C. Experimental Tools

We use the IMPACT tool suit [18] to automatically tune application codes and collect performance measurements of benchmarks on various machine configurations. The IMPACT C compiler is a retargetable compiler with code optimization components developed for multiple-instruction-issue processors. It incorporates code improving techniques such as function inline expansion, instruction placement, loop unrolling, loop peeling, memory disambiguation, register renaming, branch prediction, critical path depth reduction, and an integrated register allocation and code scheduling algorithm for both VLIW and Superscalar architectures. The target machine for IMPACT C is described using a high-level machine description (HMDES) (see Section IV for an example) supplied by a user. IMPACT provides cycle-level simulation of the processor implementation. Fig. 1 shows the flow of simulation using the IMPACT tools.

We collect run-times (expressed as a number of cycles) of the benchmarks on 175 different machine configurations. First we build executables of the benchmarks for seven different processor configurations. They 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 ILP for each configuration. All the machine configurations have the same number of ALU and memory units as the issue width. The optimized code is consumed by the Lsim simulator. We simulate the benchmarks for a number of different cache configurations. For each executable of a benchmark, we simulate 25 combina-

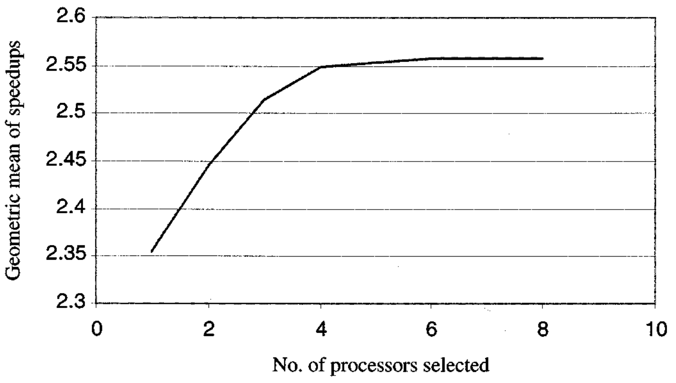


Fig. 5. Performance versus number of selected processors: Area constraint 84 mm^2.

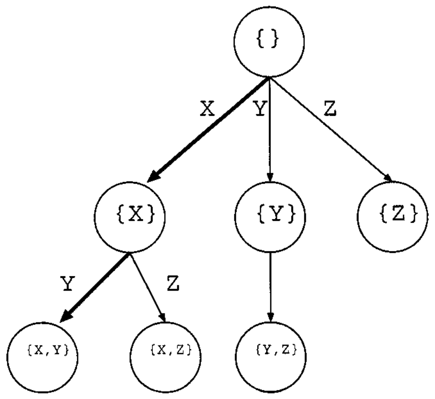


Fig. 6. A search space and a selected path (X, Y) indicated by think lines.

tions of instruction cache and data cache ranging from (512 bytes, 512 bytes) to (8 KB, 8 KB).

Measured run-times of benchmarks through simulations are normalized with respect to a baseline machine. We selected as a baseline configuration a machine with one branch unit, one-issue unit, 512 bytes of instruction cache, and 512 bytes of data cache. An example set of results is shown in Table V. There are 128 different machine configurations that satisfy the illustrated area constraint, 16 mm^2. After run-times are measured, we eliminate machines that are dominated by at least one other machine. By dominated, we mean a machine runs slower than or equal to the speed of another machine for *all* benchmarks. In this particular example, there are seven machines left after dominated machine configurations are eliminated. The areas of the machine configurations are shown in Table VI.

IV. Approach

Our hypothesis is that a set of machine configurations that run a given set of benchmarks equally well with respect to a baseline machine can be found. In other words, there is at least one machine from the set that

Machine	Application 1	Application 2	Arithmetic mean	Geometric mean
m1	1	6	3.5	2.65
m2	3	4	3.5	3.46

Fig. 7. An example of speed-up numbers and machine selection.

can be used to build an application-specific system. Effectively, we can say that the machine is optimized to run the specific application. In this section, we describe our approach to the selection problem. First we show the global flow of the design process. We describe the combinatorial nature of the search space by showing an example search tree.

A. Global Design Flow

The experiment is carried out by first selecting a set of machine models. A portion of an example high-level machine description file (HMDES) used by the IMPACT tool suite is shown in Fig. 3. Machines are described using HMDES. The HMDES files are compiled by the HMDES compiler [34], [35]. Detailed and precise descriptions of the execution constraints for the HP PA7100, PLAYDOH, Intel Pentium, EPIC, and Sun SPARC have been ported and widely used by IMPACT C compiler and Lsim simulator. The benchmarks are compiled by the IMPACT C compiler for the machines described in HMDES. The architecture model we used in the experiment is the Hewlett Packard PA-RISC (HPPA) [1]. All the executables are simulated using the IMPACT simulator, Lsim. At simulation time, we specify cache configurations for the simulator. Lsim simulates everything from different branch prediction scheme, reorder buffer, caches and memory. Memory latency, misprediction penalty and ALU latency are specified as Lsim parameters (Fig. 2) in the system model being simulated. We did not incorporate a second level cache or specialized ALU; however, adding one and changing ALU latency is possible in Lsim. Through simulations we measure run-times of all the benchmarks. The run-times are normalized with respect to a baseline machine run-time for each benchmark to obtain speed-up numbers. After all the simulations are completed, we begin searching for the best machine configuration sets for specific area constraints. For each area limit, we eliminate all machines that do not satisfy the area bound. From the machines that satisfy the area bound, we eliminate all the dominated machines (refer to Section III). Finally, we apply the K-selection algorithm (see Section V), to select a set of machine configurations that run the benchmark set best. Fig. 4 shows the global flow of design process.

B. Selection Problem Search Space

The search space is relatively small due to the area constraint and the number of dominated implementations. Nevertheless, there is a possiblity that the search space explodes due to its combinatorial nature. The likelihood of this phenomenon occuring appears to be a strong function of the area models used. Fig. 6 shows an example search space with three machine configurations. The search starts with an empty set (indicated by the root node in the diagram) and follows one of the possible path in the search tree. Each node in the search tree is an instance of selection completed.

Fig. 5 shows an example measurement on performance versus number of selected processors. We are looking for the break point of diminishing returns for adding configurations to a set. This goal will be elaborated in more detail in the next section.

V. Selection Problem Formulation

In this section we formulate the problem of finding the minimum number of machine configurations for a given set of benchmarks in such a way that all the benchmarks execute well on at least one of the selected machines.

A. Informal Description of Problem

Informally, the problem can be stated as follows: Given an area constraint and speed-up numbers of benchmarks on machines that fit into the given area, we want to select a subset of the machine configurations in such a way that the geometric mean of speed-ups across all the benchmark is maximized and the subset size is kept small.

We normalize the run time with respect to a baseline since we are not interested in the sum of run-times [36]. The sum of run-times does not reflect the performance effect of shorter benchmarks in the presence of longer benchmarks. In some cases, a benchmark that takes a long time to complete due to large data sets dominates the sum of run-times.

We use the geometric mean to summarize the selected machines since we normalize the measurements [36]. In general, the geometric mean is not a good method of summarizing performance numbers [37] as it does not show the nature of the workload. For example, consider a workload consisting of two applications. On a baseline machine one application takes 5000 s to complete and the other 2 s. We compare two machines based on a set of normalized numbers. Assume that one machine improves the performance of the first application by a factor of two and the other the second application by the same factor. Then the geometric mean indicates that the performance of the two machines is the same although the first machine cuts the running time by 2500 s while the second machine by only 1 s. This is a problem when we summarize normalized performance numbers for a *mix* of workload. As indicated earlier, however, we are not interested in the machine performance on a *mix* of workload. Instead, we are interested in predicting the performance of one of the selected machines on each individual benchmarks. As an illustration, consider the speed-up numbers given in Fig. 7. While the arithmetic mean suggests that there is no difference between m1 and m2, the geometric mean provides more useful insight.

We want to see how many machine configurations are necessary in order to achieve high performance for all the benchmarks. The objective function of the optimization problem is minimization of the number of selected machine configurations, thereby, on average, maximizing the number of benchmarks that can be run on a processor as though it is optimized for each individual benchmark. In one extreme case, we might end up with machine configurations for each individual benchmark. On the other extreme, we might need only one processor solution for all applications.

B. Formal Description of Problem

We now define the problem using more formal Garey–Johnson format [39].

Selection problem:

Instance: Given a set of n benchmarks, $a_i, i = 1, 2, \ldots, n, k$ machine configurations, $m_j, j = 1, 2, \ldots, k$, the speed-up factors E_{ij} of the benchmarks $a_i, i = 1, 2, \ldots, n$ on the machines $m_j, j = 1, 2, \ldots, k$ with respect to a baseline machine and constants K and C.

Question: Is there a set M of K machine configurations, $c_p, p = 1, 2, \ldots, K$, such that $\sqrt[n]{\prod_{i=1}^{n} \max_{j \in M} E_{ij}} \leq C$?

To determine the constant K we divide the problem into two subproblems, namely, a ω-selection problem and K-selection problem. Starting from $\omega = 1$ we iteratively increase ω until the benefit of increasing ω is less than a given threshold ρ. Formally the subproblems are stated as follows.

```
for a set of machine configurations {
    for a set of benchmarks {
        generate an executable of a benchmark for the target machine;
        measure speed-up factors with respect to the baseline machine;
    }
}
eliminate all machines that don¡¦t satisfy area constraint
eliminate all machines that are dominated by at least one other machine;
$R = 1$;
$i = 1$;
$D_{i-1} = 1$;
while ( ( $R \geq \rho$ ) && ( $i \leq k$ ) ) {
    $D_i$ = branch-and-bound( $D_{i-1}$, $i$ );
    $R = \frac{D_i - D_{i-1}}{D_i}$;
    $D_{i-1} = D_i$;
    i++;
}
```

```
branch-and-bound( $D_{i-1}$, $w$ ) {
Stack s;
Node c;
    $D_{best} = D_{i-1}$ ;
    Generate_new_nodes( c, $D_{best}$ , s );
    while( s is not empty ) {
        c = take a node out of s;
        If( bound() == FALSE ) then continue;
        else {
            if the number of selected machines is less than w then
                Generate_new_nodes( c, s );
            else {
                $D = \sqrt[n]{\prod_{i=1}^{n} \max_{j \in P} E_{ij}}$ ;
                If( D > $D_{best}$ ) then $D_{best}$ = D;
            }
        }
    }
    return $D_{best}$ ;
}
```

```
Generate_new_nodes( c, s ) {
    for a set of possible choices {
        est = estimate( c, choice );
        insert the new choice and sort s based on est;
    }
}
```

```
estimate( c, choice ) {
    for all benchmarks I {
        $speedup_i = \max_{j \in c \cup \{choice\}} E_{ij}$;
    }
    $n = |allbenchmarks| - |c \cup \{choice\}|$;
    for all the machines not selected j
        $improve_i = \sum_{i=1}^{n} speedup_i - E_{ij}$;
    Sort( improve );
    greedy = n best machines based on improve
    for all benchmarks I {
        $speedup_i = \max_{j \in \{c \cup \{choice\} \cup greedy\}} E_{ij}$;
    return ( geometric mean of speedup );
}
```

Fig. 8. System-level synthesis of application-specific programmable machines.

ω-selection problem: Given a set of n benchmarks, $a_i, i = 1, 2, \ldots, n, k$ machine configurations, $m_j, j = 1, 2, \ldots, k$, the speed-up factors E_{ij} of the benchmarks $a_i, i = 1, 2, \ldots, n$ on the machines $m_j, j = 1, 2, \ldots, k$ with respect to a baseline machine and constants ω

$$\textbf{Maximize:} \quad D_\omega = \sqrt[n]{\prod_{i=1}^{n} \max_{j \in p} E_{ij}} \tag{2}$$

where P is the selected machine set of size ω.

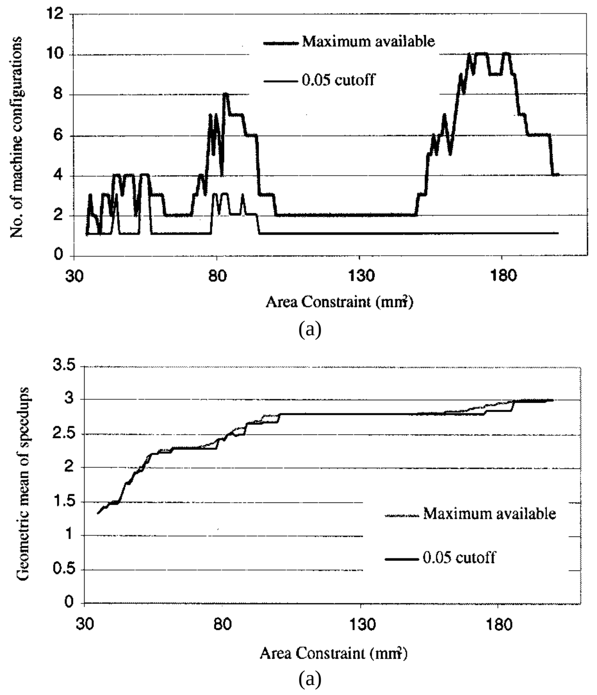


Fig. 9. Quadratic complexity issue unit area model. Selected processor configurations (cutoff value: 0.05): (a) processor configurations and (b) performance.

The size of the machine set is determined by an iterative test of comparing D_ω and $D_{\omega+1}$. Since the D is monotonic, we continue to evaluate D and compare them using (3) until we reach a point where the benefit of the set size increase drops below a certain limit.

K-selection problem:

$$\min\left\{\omega \middle| \rho \leq \frac{D_{\omega+1} - D_\omega}{D_\omega}, \quad \omega = 1, 2, \ldots, k-1\right\} \tag{3}$$

where D_ω is given by (2) and ρ is a cutoff ratio.

VI. Solution Space Exploration: Strategy and Algorithms

The algorithm for system-level synthesis of application-specific programmable processor is given in Fig. 8. Considering that the run-time of simulations for 20 benchmarks on 175 machine configurations is about a week, we can tolerate a longer search time to find the optimal result. Generally, the size of the search problem is dramatically reduced by eliminating machine configurations that do not satisfy a given area constraint and those that are dominated by at least one other machine. Consequently, a smaller number of machines needs to be considered. The machine configuration A dominates the configuration B if no benchmarks have longer execution times on the machine A than on machine B.

The search for an optimum solution is organized using an implicit enumeration method. In particular, we adopt a branch-and-bound algorithm shown in Fig. 8 to speed up the selection.

The branch-and-bound algorithm consists of two major components: branching and evaluation. The branching step takes the current state of selection (a node in the search tree) and generates a number of new nodes by adding an available (still not considered in particular search path of the search tree) machine to the current state of selection (refer to Fig. 6). As shown in Fig. 8, it examines to see if adding a machine to the current state of selection can result in a better solution than the current best solution found. Initially, the current best solution is set to the previous best solution. The previous best solution is the best solution found for the machine set size less than the current search size by

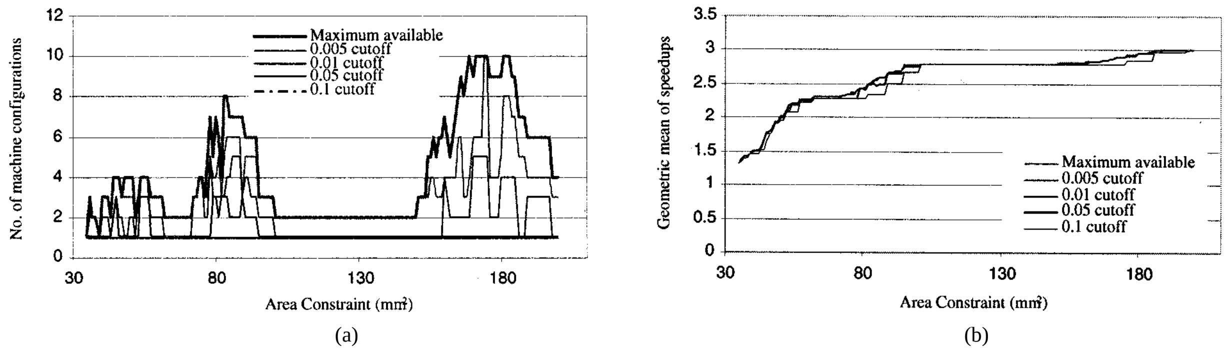


Fig. 10. Quadratic complexity issue unit area model: Selected processor configurations: (a) processor configurations and (b) performance.

TABLE VII
SNAPSHOTS OF SPEED-UPS OF BENCHMARKS AT VARIOUS CUT-OFF VALUES FOR THE GIVEN AREA CONSTRAINT (84 mm^2, 100 mm^2, 169 mm^2) (NOTE: ACTUAL AREA UNDER THE AREA CONSTRAINTS ARE GIVEN IN TABLE VIII)

No.	Benchmark	cut-off:0.05			cut-off:0.01			cut-off:0.005		
		84mm^2	100mm^2	169mm^2	84mm^2	100mm^2	169mm^2	84mm^2	100mm^2	169mm^2
1	JPEG encoder	3.02	3.51	3.87	3.02	3.65	3.87	3.02	3.65	3.87
2	JPEG decoder	3.56	3.91	4.27	3.56	4.23	4.27	3.56	4.23	4.27
3	MPEG encoder	2.45	2.56	2.57	2.45	2.56	2.57	2.45	2.42	2.57
4	MPEG decoder	3.05	3.55	3.68	3.05	3.55	3.95	3.05	2.56	3.95
5	GSM encoder	2.22	2.30	2.31	2.22	2.30	2.32	2.24	2.94	2.32
6	GSM decoder	2.89	2.99	3.00	2.89	2.99	3.19	2.91	2.99	3.19
7	G.721 encoder	2.68	3.38	3.38	2.88	3.38	3.38	2.88	3.38	3.38
8	G.721 decoder	2.71	3.44	3.44	3.04	3.44	3.44	3.04	3.44	3.44
9	PGP encryption	2.78	3.08	3.21	2.83	3.15	3.21	2.83	2.26	3.21
10	PGP decryption	2.88	3.22	3.38	3.02	3.28	3.38	3.02	2.15	3.46
11	Pegwit encryption	2.64	2.66	2.82	2.64	2.79	2.82	2.64	3.28	2.82
12	Pegwit decryption	2.63	2.68	2.84	2.63	2.82	2.84	2.62	2.79	2.84
13	Mipmap	2.81	2.69	2.79	2.81	2.94	3.10	2.81	3.56	3.10
14	OS-demo	2.22	2.40	2.47	2.25	2.42	2.47	2.25	2.82	2.47
15	Texgen	2.11	2.31	2.40	2.11	2.40	2.40	2.11	2.42	2.40
16	Rasta	2.13	2.22	2.27	2.12	2.26	2.31	2.13	2.31	2.31
17	EPIC encoder	1.93	1.78	2.17	1.93	2.06	2.17	2.04	2.17	2.30
18	EPIC decoder	1.70	1.76	1.77	1.70	1.76	1.83	1.70	1.76	1.83
19	ADPCM encoder	2.30	2.31	2.31	2.30	2.31	2.43	2.30	2.42	2.44
20	ADPCM encoder	2.33	2.34	2.42	2.33	2.34	2.55	2.34	2.40	2.65

one. The branching is bounded by the bounding function. The bounding function compares the current node and a candidate processor with the best node of the same size found. The node size is the number of processors. If the current node and the candidate are dominated by the best node, then we cut the path off from search. We compute the lower bound of the geometric mean of the maximum speed-up factors of each benchmark. The lower bound is obtained by using a steepest descent algorithm. The steepest descent algorithm selects machines in the order that the biggest improvement can be achieved. If the estimate is greater or equal to the current best solution, we have an opportunity to find a better solution than the current best solution by exploring the search path. Otherwise, there is less of a chance of obtaining a better solution. We sort the search order based on the lower bound so to increase the bounding rate.

VII. EXPERIMENTAL RESULTS

We evaluated the tools and algorithms by running extensive experiments ranging from the area constraint of 30 to 200 mm^2. The implementation technology is assumed to be 0.35 μm. For each area constraint, we obtain an optimum set of machine configurations for cutoff values 0.1, 0.05, 0.01, and 0.005. The cutoff values are ρ's as defined in (3) (see Section V).

Fig. 9(a) shows an experimental result using the cutoff value of 0.05. The thicker line shows the number of machines that are left after eliminations. The thinner line in the figure indicates the number of selected machines to cover all the benchmarks under area constraints. We clearly see that we need more machine configurations when less area is available. On the other hand, the more area we have, the more general the processor we can design. The results suggest that when more than 100 mm^2 of area is available, there is no advantage in having more than one architecture to be able to build application-specific systems for all the benchmarks. Moreover, for the given compiler technology and benchmarks, there is no need to have more than 100 mm^2 of area since the speed-up increase achieved by machines greater than 100 mm^2 are minimal.

The overall performance comparison between all configurations and selected configurations are shown in Fig. 8(b). There are three distinctive points where the speed-up increase rate changes. Up to the area 57 mm^2, we see rapid performance increase, which is mainly due to increased amount of cache memories. From 57 to 101 mm^2 the measurement shows modest increase of performance. The performance increase

TABLE VIII
SELECTED PROCESSORS THAT RUN BENCHMARK BEST FOR THE SNAPSHOT CASES IN TABLE VII: THE LAST COLUMN SHOWS BENCHMARKS THAT RUN BEST ON PROCESSORS GIVEN IN COLUMN 3. COLUMN 4 SHOWS MACHINE CONFIGURATIONS IN THE FORM OF: ISSUE WIDTH, NUMBER OF ALUs, NUMBER OF BRANCH UNITS, NUMBER OF MEMORY UNITS, SIZE OF INSTRUCTION CHACHE (KB), SIZE OF DATA CACHE (KB). ACTUAL AREAS ARE GIVEN IN THE FIFTH COLUMN

Cut-off	Max area	No.	Configuration	Actual area	Benchmarks that run best
0.05	84	*m1*	(2, 2, 1, 2, 8, 8)	61.11	decode, encode, epic, mpeg2enc, pegwitdec, pegwitenc
		m2	(4, 4, 1, 4, 4, 4)	81.89	cjpeg, djpeg, gsmdecode, gsmencode, mipmap, osdemo, pgpdecode, pgpencode, rasta, texgen
		m3	(4, 4, 2, 4, 1, 4)	83.24	mpeg2dec rawcaudio rawdaudio unepic
	100	*m4*	(4, 4, 2, 4, 8, 4)	94.36	all applications
	169	*m5*	(4, 4, 2, 4, 8, 8)	100.71	all applications
0.01	84	*m6*	(2, 2, 1, 2, 8, 8)	61.11	epic, mpeg2enc, pegwitdec pegwitenc
		m7	(4, 4, 1, 4, 4, 4)	81.89	cjpeg djpeg gsmdecode gsmencode mipmap rasta texgen
		m8	(4, 4, 1, 4, 8, 1)	83.46	decode, encode, osdemo, pgpdecode, pgpencode
		m9	(4, 4, 2, 4, 1, 4)	83.24	mpeg2dec, rawcaudio, rawdaudio, unepic
	100	*m10*	(4, 4, 1, 4, 8, 8)	94.58	cjpeg, djpeg, epic, mipmap, osdemo, pegwitdec, pegwitenc, pgpdecode, pgpencode, rasta, texgen
		m11	(4, 4, 2, 4, 8, 4)	94.36	decode, encode, gsmdecode, gsmencode, mpeg2dec, mpeg2enc, rawcaudio, rawdaudio, unepic
	169	*m12*	(4, 4, 2, 4, 8, 8)	100.71	cjpeg, decode, djpeg, encode, epic, mpeg2enc, osdemo, pegwitdec, pegwitenc, pgpdecode, pgpencode, texgen
		m13	(8, 8, 2, 8, 4, 4)	166.33	gsmdecode, gsmencode, mipmap, mpeg2dec, rasta, rawcaudio, rawdaudio, unepic
0.005	84	*m14*	(2, 2, 1, 2, 8, 8)	61.11	mpeg2enc, pegwitdec, pegwitenc
		m15	(4, 4, 1, 4, 1, 8)	83.46	epic
		m16	(4, 4, 1, 4, 4, 4)	81.89	cjpeg, djpeg, mipmap, rasta, texgen
		m17	(4, 4, 1, 4, 8, 1)	83.46	decode, encode, osdemo, pgpdecode, pgpencode
		m18	(4, 4, 2, 4, 1, 4)	83.24	mpeg2dec, rawcaudio, unepic
		m19	(4, 4, 2, 4, 4, 1)	83.24	gsmdecode, gsmencode, rawdaudio
	100	*m20*	(4, 4, 1, 4, 8, 8)	94.58	cjpeg, djpeg, mipmap, osdemo, pegwitdec, pegwitenc, pgpdecode, pgpencode, rasta, texgen
		m21	(4, 4, 2, 4, 4, 8)	94.36	epic, mpeg2dec, rawcaudio, rawdaudio
		m22	(4, 4, 2, 4, 8, 4)	94.36	decode, encode, gsmdecode, gsmencode, mpeg2enc, unepic
	169	*m23*	(4, 4, 2, 4, 8, 8)	100.71	cjpeg, decode, djpeg, encode, mpeg2enc, osdemo, pegwitdec, pegwitenc, pgpdecode, texgen
		m24	(8, 8, 2, 8, 1, 8)	168.68	epic, rawdaudio
		m25	(8, 8, 2, 8, 4, 4)	166.33	gsmdecode, gsmencode, mipmap, mpeg2dec, unepic
		m26	(8, 8, 4, 8, 1, 2)	166.70	pgpdecode, rawcaudio

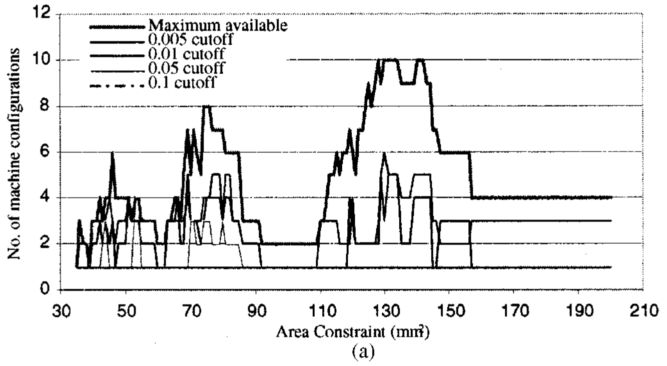
shown in this interval is mainly due to increased issue width. For the processors larger than 101 mm^2, the performance increase is minimal. One of the underlying reasons that causes the phenomenon is that the ILP found by the compiler and hardware scheduler is fully exploited by having a certain amount of hardware, thereby performance increase possibility is exhausted. The limitation of performance increase in the face of increased area illustrates either the limitation of the current compiler technology or the inherent lack of ILP in the benchmarks. Note, however, that the measurement is not for a single processor. Smaller area cases tend to have more than one architectures which are more application specific.

Experimental results for the cutoff values 0.1, 0.05, 0.01, and 0.005 are given in Fig. 10. Smaller cutoff values result in machine configuration sets that are more tuned to each application. In general, however, a smaller cutoff value does not result in dramatic performance increase. In most cases, the cutoff value of 0.05 appears to give a good tradeoff between the number of machine configurations and performance.

Speed-up factors of each benchmark are shown in Table VII. They are snapshots of experimental results summarized by the line graphs in Fig. 10. The table contains maximum speed-up factors for three cutoff values (0.05, 0.01, and 0.005) and three area constraints (85, 100, and 169 mm^2). Note that the area constraints are not actual areas but rather bounds. We consider machines under the given area constraints. Table VIII gives the number of machine configurations selected and the best performing machine configuration for each benchmark. The actual areas of the selected machines are given in column 5 of the table. The combinations of components for the selected machines are shown in column 4.

Fig. 11 shows the results when the liner complexity issue unit area model is assumed. The results suggest that the machine configuration selection problem has no strong dependence to an issue area model used. Although we observe that there is shift to smaller areas, essentially the results are identical to the results based on the quadratic complexity issue unit area model.

In summary, we found that under the machine models and machine configuration choices described in this paper, when more than 100 mm^2 of area is available, there is little advantage in having more than one architecture to be able to build application-specific systems for all the benchmarks. Moreover, for the given compiler technology and benchmarks, there is little need to have more than 100 mm^2 of area since



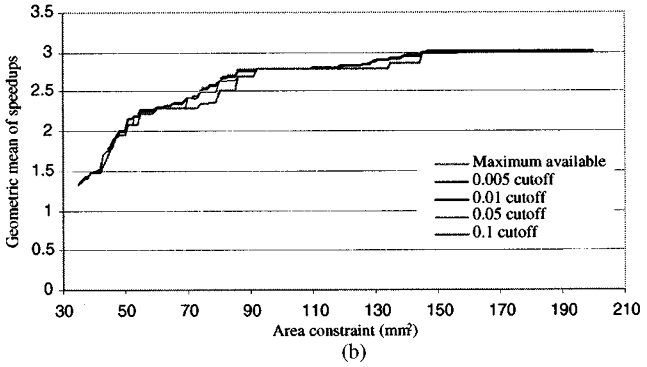


Fig. 11. Linear complexity issue unit area model: selected processor configurations: (a) processor configurations and (b) performance.

the speed-up increase achieved by machines greater than 100 mm^2 are minimal. One notable exception is that for highly cost sensitive designs we observe a need for small number of specialized architectures which achieve smaller areas.

VIII. Conclusion

The arrival of production quality ILP compilers and commercial DSPs with VLIW architecture stimulated the idea of programmable processors that are aggressively tuned to specific applications. The assumption behind the idea is that there are ways of designing programmable processors that can exploit the run-time characteristics of specific applications. The run-time characteristics include the available ILP, demand on various hardware components such as cache memory units, register files, and the number of function units. It is assumed that a microprocessor can be designed by adding hardware components tailored to a specific application so that it can execute the single application extremely well. We ran extensive experiments on a framework based on the key paradigms of CAD and architecture communities. This combination enabled us to gain valuable insights about design and use of application-specific programmable processors for modern applications. We evaluated 175 machine configurations on 20 benchmarks under the area constraint ranging from 30 to 200 mm^2. For each area constraint, we obtain an optimum set of machine configurations for a number of cutoff values. The run time of the entire synthesis process was about a week. It is well known that when the area constraint is tight, more machine configurations are needed for application specific designs. In the figures, we found out that even with more area, there still exists a fair number of different configurations due to the introduction of different functional units (branch unit and ALU) with the tradeoff of cache size.

In the system level integration market, we believe a standard design solution means a quick time to market and guaranteed functionality. We develop this framework to ease design managers finding their chips portfolio in their particular interested domains.

We have found that the framework introduced in this paper can be very valuable in making early design decisions such as area and architectural configuration tradeoff, cache and issue width tradeoff under area constraint, and the number of branch units and issue width.

References

[1] M. J. Mahon *et al.*, “Hewlett-packard precision architecture: The processor,” *Hewlett-Packard J.*, vol. 37, no. 8, 1996.
[2] R. D. Hof and O. Port, “Silicon dreams,” in *Business Week Int. Ed.*, May 13, 1996.
[3] W. Strauss, *DSP Strategies 2002. Forward Concepts*, 1998.
[4] J. Turley, “SA-1100 puts PDA on a chip,” *Microprocessor Rep.*, vol. 11, no. 12, Sept. 1997.
[5] J. A. Fisher, “Trace scheduling: A technique for global microcode compaction,” *IEEE Trans. Computing*, vol. 30, pp. 478–490, 1981.
[6] W. M. W. Hwu *et al.*, “The superblock: An effective technique for VLIW and superscalar compilation,” *J. Supercomputing*, 1993.
[7] S. Banerjia, W. A. Havanki, and T. M. Conte, “Treegion scheduling for highly parallel processors,” in *Euro-Par*, Passau, Germany, 1997.
[8] S. A. Mahlke, “Effective compiler support for predicated execution using the hyperblock,” in *Proc. Int. Symp. Microarchitecture*, 1992.
[9] P. Y. Hsu, “Highly concurrent scalar processing,” in *Tech. Rep. CSG-49*, Coordinated Science Lab., Univ. Illinois at Urbana-Champaign, 1986.
[10] J. Turley and H. Hakkarainen, “TI’s new ’C6x DSP screams at 1600 MIPS,” *Microprocessor Rep.*, vol. 11, pp. 14–17, 1997.
[11] R. P. Colwell *et al.*, “A VLIW architecture for a trace scheduling compiler,” in *Proc. ASPLOS-II*, 1982.
[12] P. Kalapathy, “Hardware-software interaction on MPACT,” *IEEE Micro.*, vol. 17, pp. 20–26, 1997.
[13] C. Hansen, “MicroUnity’s mediaprocessor architecture,” *IEEE Micro.*, vol. 17, pp. 34–41, 1997.
[14] R. B. Lee and M. D. Smith, “Media processing: A new design target,” *IEEE Micro.*, vol. 17, pp. 6–9, 1997.
[15] A. Peleg and U. Weiser, “MMX technology extension to the Intel architecture,” *IEEE Micro.*, vol. 16, no. 4, pp. 42–50, Aug. 1996.
[16] J. A. Fisher, P. Faraboschi, and G. Desoli, “Custom-fit processors: Letting applications define architectures,” in *Int. Symp. Microarchitectures*, Paris, France, 1996.
[17] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “Mediabench: A tool for evaluating and synthesizing multimedia and communications systems,” in *Proc. Int. Symp. Microarchitectures*, 1997.
[18] P. P. Chang *et al.*, “IMPACT: An architectural framework for multiple-instruction-issue processors,” in *Int. Symp. Computer Architecture*, 1991.
[19] G. Goossens *et al.*, “Embedded software in real-time signal processing systems: Design technologies,” *Proc. IEEE*, vol. 85, pp. 436–454, Mar. 1997.
[20] P. G. Paulin *et al.*, “Embedded software in real-time signal processing systems: Application and architecture trends,” *Proc. IEEE*, vol. 85, pp. 419–435, Mar. 1997.
[21] P. Marwedel, “Processor-core based design and test,” in *Proc. Asia and South Pacific Design Automation Conf.*, 1997.
[22] G. Araujo, A. Sudarsanam, and S. Malik, “Instruction set design and optimizations for address computation in DSP architectures,” in *Proc. Int. Symp. Syst. Synthesis*, 1996.
[23] C. Liem, T. May, and P. Paulin, “Instruction-set matching and selection for dsp and asip code generation,” in *Proc. European Design Test Conf.*, 1994.
[24] A. Sudarsanam and S. Malik, “Memory bank and register allocation in software synthesis for ASIPs,” in *Proc. Int. Conf. Computer-Aided Design*, 1995.
[25] S. Liao *et al.*, “Instruction selection using binate covering for code size optimization,” in *Proc. Int. Conf. Computer-Aided Design*, 1995.
[26] R. Leupers and P. Marwedel, “Retargetable generation of code selectors from HDL processor models,” in *Proc. European Design Test Conf.*, 1997.
[27] W. Zhao and C. A. Papachristou, “An evolution programming approach on multiple behaviors for the design of application specific programmable processors,” in *Proc. European Design Test Conf. ED&TC 96*.

[28] K. Kim, R. Karri, and M. Potkonjak, "Heterogeneous built-in resiliency of application specific programmable processors," in *Proc. Int. Conf. Computer-Aided Design*, 1996.

[29] T. Conte and W. Mangione-Smith, "Determining cost-effective multiple issue processor designs," in *Proc. Int. Conf. Computer Design*, 1993.

[30] T. M. Conte, K. N. P. Menezes, and S. W. Sathaye, "A technique to detemine power-efficient, high-performance superscalar processors," in *Proc. 28th Hawaii Int. Conf. System Sciences*, 1995.

[31] J. Kin *et al.*, "Power efficient mediaprocessors: Design space exploration," in *Proc. 36th Design Automation Conf.*, 1999.

[32] 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, 1995.

[33] S. P. Song and M. Denman, "The powerPC 604 RISC microprocessor," in *Tech. Rep.*, Motorola Inc., and IBM, 1994.

[34] J. C. Gyllenhaal, W. W. Hwu, and B. R. Rau, "Optimization of machine descriptions for efficient use," in *Proc. 29th Int. Symp. Microarchitecture*, Dec. 1996.

[35] ——, "HMDES version 2.0 specification," in *IMPACT Tech. Rep.*, IMPACT-96-03, University of Illinois, Urbana, IL, 1996.

[36] R. Jain, *The Art of Computer Systems Performance Analysis*. New York: Wiley, 1991.

[37] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. San Francisco, CA: Morgan Kaufman, 1993.

[38] J. E. Smith, "Characterizing computer performance with a single number," *Communications ACM*, vol. 31, no. 10, pp. 1202–1206, 1998.

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