Quantitative Selection of Media Benchmarks

Chunho Lee and Miodrag Potkonjak
Computer Science Department, University of California
Los Angeles, CA 90095-1596, USA
e-mail: leec@cs.ucla.edu, miodrag@cs.ucla.edu

Abstract— Over the last decade, significant advances have been made in compilation technology and microprocessor architectures for capitalizing on instruction-level parallelism (ILP). While most of the processors containing superscalar, VLIW and SIMD structures that are well matched to the needs and capabilities of the ILP compilers are targeted at embedded applications, the majority of all ILP compilation work has been conducted in the context of general-purpose computing. As a consequence, there currently exists a gap between the compiler community and embedded application developers. We present a quantitative benchmark selection and validation technique to close the gap. It is based on more quantitatively rigorous metrics to accurately measure usefulness and effectiveness of benchmarks. We assemble a collection of DSP and communication applications written in a high-level language. An experimental selection of benchmarks from the collected applications is conducted and its applicability and usefulness are verified through an application specific system synthesis.

I. Introduction

As computer hardware and software systems become increasingly more complex, testing and simulation tend to replace verification and validation. Since developers have to address the issues of correctness and performance at the same time, benchmarks play a more critical role than ever in complex system development processes. From the system synthesis point of view, optimizing synthesis efforts to poorly designed benchmarks can be very costly due to the resource-restricted nature of embedded systems.

Most benchmarks designed to evaluate DSP systems have mainly been composed of a mix of low level kernels or small hand-written applications in assembly language [13, 15, 16]. Many general-purpose processors with so-called multimedia extensions were mainly motivated and developed by benchmarks based on the kernels. Two major problems have been raised concerning the validity and usefulness of the existing benchmarks. First, quality and performance of the existing benchmarks in system development or system synthesis have never been reported. Second, the existing benchmarks does not adequately reflect major advances in architecture and compiler technology.

Modern advances in compiler technology for instruction-level parallelism (ILP) have significantly increased the ability of a compiler to capitalize on the opportunities for parallel execution that exist in various programs [9]. At the same time, a number of new microprocessor architectures have been introduced. They present the hardware structures that nicely matches with most ILP compilers. Architectural enhancements found in commercial products include predicated instruction execution, VLIW execution and a split register file. Multi-gauge arithmetic (or variable-width SIMD) is found in many mediaprocessors. Most of the multimedia extensions of general-purpose processors also adopt this architectural enhancement. The majority of ILP research, however, has focused on general-purpose computing, and in particular the integer SPEC benchmarks. As a consequence, there currently exists a gap between the compiler community and embedded application developers.

We present a quantitative benchmark selection and validation technique to close the gap. It is based on more quantitatively rigorous metrics to accurately measure usefulness and effectiveness of benchmarks. We assemble a collection of DSP and communication applications written in a high-level language. First, we characterize collected applications by measuring a number of run-time characteristics of the program. And then we develop an algorithm to select a subset of collected applications. An experimental selection of benchmarks from the collected applications is conducted and its applicability and usefulness are verified through experimental application specific system synthesis.

The remainder of this paper is organized as follows. Section II discusses the relevant previous works and our contributions in the area. Preliminary materials including applications of a benchmark suite, quantitative benchmark selection criteria, and tools used in this work are described in Section III. Section IV briefly discusses the global design flow of a media benchmark suite. Section V presents the approach used for collecting media applications and explains key components of each application. After metrics measured and used to characterize an application are briefly discussed, run-time characteristics of all the applications are given in the section. Section VI

0-7803-4425-1/98/$10.00 1998 IEEE

presents a benchmark selection approach. Section VII presents an experimental media benchmark selection and an experimental core-based system synthesis based on the selected benchmark suite is discussed. Finally, Section VIII draws conclusions.

II. Previous Works and Our Contributions

Most DSP system evaluation has been focused on assembly language or the use of very small kernels [16]. Saghir et al. developed a mix of low level kernels and small applications with the goal of compiler evaluation and system synthesis [14]. This suite emphasizes kernel codes and low-level filter operations. Furthermore, no clear measurement is made to establish the value of the new code base.

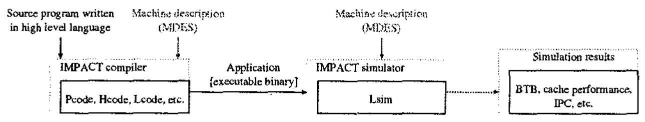
A number of researchers have used benchmark suites to drive automatic synthesis of embedded systems [4, 6]. However, we know of no study that quantitatively evaluated the quality and performance of a benchmark suite. Without this step, it is unclear what the benefit of the benchmark suite is.

The quantitative media benchmark design effort presented in this paper is the first to focus on complete applications for multimedia and communications systems, as well as the first to use only applications written in a high-level language in order to stress compilation technology. Furthermore, this paper presents a philosophy for benchmark design that is more rigorous, and consequently well tested, than *ad hoc* methods used previously.

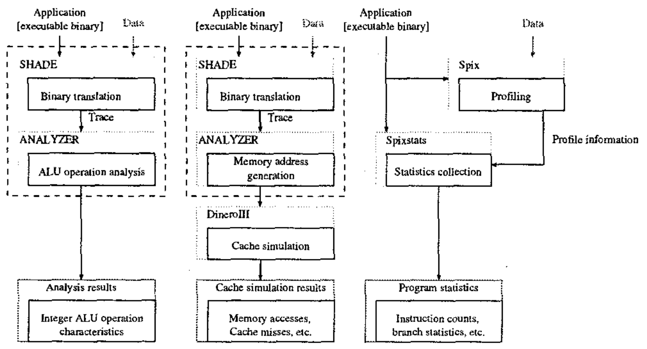
III. Preliminaries

This section presents the reasoning and goals of the quantitative approach to benchmark selection. Conventional wisdom, and a history of benchmarking, suggests that benchmarks serve a number of purposes throughout design, simulation, implementation, and testing phase of computer systems development. The primary objective in designing benchmark suite is to select a set of applications in a domain in such a way that it can satisfy all the prescribed purposes.

Through rigorous analysis of objectives, we identified quality criteria that guide benchmark design: relevancy, compactness, comprehensiveness, and resolvability. A benchmark suite should be based on real applications representing their full diversity and complexity. As it is prohibitively expensive to include a large number of applications in a benchmark suite, the size of a benchmark suite should be as small as possible while providing a certain degree of assurance that it covers applications in a domain comprehensively. A good benchmark set strikes balance between its size and comprehensiveness based on real life applications. Finally, a benchmark should result in a fair comparison among designs and approaches with good resolutions. In other words, we do not want all the benchmarks too easy or too difficult, which may result in no differentiation among the various systems that may or may not equal in their quality.


Source program written in high level language
Machine description (MDES)
IMPACT compiler
Pcode, Hcode, Lcode, etc.
Application [executable binary]
Machine description (MDES)
IMPACT simulator
Lsim
Simulation results
BTB, cache performance, IPC, etc.

(a) Using IMPACT suite



(b) Using SpixTools, SHADE and DineroIII

Fig. 1. Run-time characteristic measurement flows

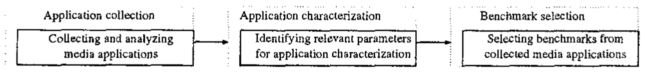

Application collection
Collecting and analyzing media applications
Application characterization
Identifying relevant parameters for application characterization
Benchmark selection
Selecting benchmarks from collected media applications

Fig. 2. The global design flow of quantitative benchmark selection

We use a set of profiling, tracing, and simulation tools as well as compilers to conduct the application characterization experiments. We measure a set of metrics which computer architects generally agree are important.

Application characterization begins by executing each application under two different execution environments on two different processor architectures. The IMPACT tool suite [1] was used to collect execution characteristics for the HPPA processor architecture. We used a single-issue processor core along with a pair of direct-mapped 16KB caches. IMPACT provides cycle-level simulation of both the processor architecture and the implementation (see Figure 1(a) for the simulation flow).

Run-time characteristics of programs were collected on the SPARC architecture using the SpixTools suite [2], SHADE tracing environment [3], and DineroIII [8]. Figure 1(b) shows the overall measurement flow of run-time characteristics. SpixTools is a collection of programs that allow instruction-level profiling of application programs. *spix* instruments application programs. As it runs, an instrumented program generates the program execution profile. *spixstats* analyzes the profile information and generates a summary of run-time statistics of the program.

SHADE is a library that provides uniform interfaces to tracing facilities. It allows users to define custom trace analyzers and thus collect rich information on runtime

TABLE I

A BRIEF DESCRIPTION OF APPLICATIONS AND DATA USED IN THE EXPERIMENT

Application	Instr.[a]	Source	Description	Data file[b]	Data Description
JPEG encoder	13.9	Independent JPEG Group	JPEG image encoding/decoding	101,484	PPM (bit map)
JPEG decoder	3.8			5,756	JPEG compressed
MPEG encoder	1,121.3	MPEG Simulation Group	MPEG-2 movie encoding/decoding	506,880	YUV 4 frames
MPEG decoder	175.5			34,906	MPEG-2 (http://www.mpeg2.de/)
GSM encoder	184.2	Technische Universität, Berlin	European wireless voice coding standard	295,040	16 bit PCM
GSM decoder	73			30,426	GSM encoded
G.721 encoder	274.1	Sun Microsystems, Inc.	CCITT voice coding standard	295,040	16 bit PCM
G.721 decoder	511.7			14,7520	G.721 encoded
PGP encryption	169.9	MIT	encryption/ decryption	91,503	plain ASCII
PGP decryption	155.3			20,163	PGP encrypted
Pegwit encryption	34.0	George Barwood	encryption/ decryption	91,503	plain ASCII
Pegwit decryption	18.5			91,537	Pegwit encrypted
Ghostscript	708.9	Aladdin Software	postscript interpreter	78,519	PS color picture
Mipmap	47.6	University of Wisconsin	3-D renderers using Mesa graphics library	N/A	mipmap texture mapping example
OS-demo	9.0			N/A	3-D rendering pipeline example
Texgen	83.9			N/A	texture mapped Utah teapot
Rasta	24.4	ICSI at UCB	rasta-plp processing	17,024	SPHERE format
EPIC encoder	50.3	University of Pennsylvania	Wavelet image encoding/decoding	65,595	PGM Gray scale 256 × 256
EPIC decoder	7.2			7,432	EPIC encoded Gray scale 256 × 256
ADPCM encoder	6.8	Jack Jansen	speech compression and decompression	295,040	16 bit PCM
ADPCM decoder	5.9			73,760	ADPCM encoded

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

events. We wrote two custom analyzers composed of approximately 500 lines of C code to collect run-time statistics of applications such as ALU operation counts, ALU operation width counts (i.e. 8-bit, 16-bit, and 32-bit), and percentage of ALU operations that are associated with memory operations.

Memory reference characteristic of the programs are measured by DineroIII. A small SHADE program is used to generate memory reference traces and they are consumed by DineroIII. DineroIII simulates cache models specified by a user and measures a set of cache performance metrics such as miss rates, write-back counts, bus traffic amount, etc.

IV. Design Flow of Media Benchmark Suite

The global design flow of a media benchmark suite is shown in Figure 2. The first step in benchmark design is to collect applications in a domain as diverse as possible. Although we do not claim that we have exhaustively collected all the media applications, we believe that the set of application used in this experiment represents reasonably good coverage of applications in the domain. The collected applications are instrumented, profiled, simulated and analyzed. We select a subset of measured metrics using a statistical method as relevant ones for use in benchmark selection. Finally, a subset of collected applications is selected as a media benchmark suite using a benchmark selection algorithm.

V. Media Application Collection

The set of media applications used in this experiment is composed of complete applications coded in C. The collection is composed of 21 applications culled from available image processing, communications and DSP applications. A brief summary of the collected applications and the set of data used are given in Table I.

We use the raw measurement numbers reported by the IMPACT suite to identify to relevant parameters characterizing applications. The IMPACT Lsim simulator provides a number of measurements such as IPC (instructions per cycle), BTB (branch target buffer) hit rate, cache hit rate, bus utilization, branch issue rate, and ALU issue rate. We expected media applications to provide less stress on instruction caches than other applications (e.g. SPECint) since the applications were generally intended for embedded execution. The expectation is verified by the simulation results.

The measurements made by SpixTools, DineroIII, and SHADE analyzers on SPARC are summarized in Table II and III. Note that IPC numbers reported by the IMPACT Lsim and those of Table II are not the same in nature. The former is reported by Lsim accounting all the architectural details such as BTB, cache, etc. The latter is computed based on raw performance numbers reported by SpixTools assuming perfect cache. The formula is given by

$$IPC = \frac{IC - Nop}{IC + branch\_penalty \times annulled}, \qquad (1)$$

where IC stands for total instruction count, Nop *nop* count, and annulled squashed instruction count.

Table III shows an analysis of ALU operations. *ALU ops* column displays the percentage of integer ALU operations. The percentage of ALU operations involving an immediate value are in the next column. In the final four columns, integer ALU operations associated with memory operation (i.e. store) and their breakdown in terms of their operation width. By "integer ALU operations asso-

TABLE II
RUN-TIME CHARACTERISTICS BY SPIXTOOLS AND DINEROIII

Application	IPC	I-cache[a]		D-cache[b]	
		1 KB	2 KB	2 KB	4 KB
JPEG encoder	0.966	2.19	1.32	17.22	11.36
JPEG decoder	0.995	2.39	1.15	21.67	7.15
MPEG encoder	0.896	0.74	0.12	5.36	4.23
MPEG decoder	0.971	1.93	0.47	4.53	2.95
GSM encoder	0.991	3.60	1.02	0.51	0.14
GSM decoder	0.966	0.89	0.68	1.40	1.09
G.721 encoder	0.977	11.33	7.01	0.37	0.24
G.721 decoder	0.978	10.31	7.30	1.98	0.19
PGP encryption	0.980	3.67	2.71	10.16	6.97
PGP decryption	0.980	3.63	2.79	9.50	6.63
Pegwit encryption	0.994	6.12	3.32	23.89	18.04
Pegwit decryption	0.995	6.47	2.55	25.41	17.07
Ghostscript	0.952	21.89	16.47	9.40	6.20
Mipmap	0.933	15.60	8.98	13.84	9.19
Osdemo	0.969	7.39	3.94	8.92	7.59
Texgen	0.967	15.61	13.15	12.70	8.27
Rasta	0.961	13.20	9.58	10.30	7.46
EPIC encoder	0.988	0.05	0.04	21.31	16.68
EPIC decoder	0.942	0.52	0.12	16.09	13.04
ADPCM encoder	0.856	0.05	0.03	4.16	1.88
ADPCM decoder	0.852	0.04	0.01	4.12	1.80

[a]miss rate (%)
[b]miss rate (%)

TABLE III
RUN-TIME CHARACTERISTICS MEASURED BY A SHADE ANALYZER

Application	ALU[a] (%)	Immed[b] (%)	Mem[c] (%)	ops Width (%)		
				8	16	32
JPEG encoder	59	64	27	22	12	66
JPEG decoder	60	52	36	55	08	37
MPEG encoder	51	31	08	06	19	75
MPEG decoder	45	76	21	12	02	85
GSM encoder	73	50	41	00	94	05
GSM decoder	72	53	75	00	99	01
G.721 encoder	60	72	30	01	27	72
G.721 decoder	59	71	32	04	35	62
PGP encryption	50	38	27	00	08	92
PGP decryption	50	37	28	00	04	96
Pegwit encryption	64	53	22	00	34	66
Pegwit decryption	61	56	17	00	51	49
Ghostscript	46	53	57	53	00	46
Mipmap	32	55	29	18	03	70
Osdemo	37	60	45	21	10	66
Texgen	40	53	32	25	01	50
EPIC encoder	51	54	03	14	19	66
EPIC decoder	50	63	24	00	17	83
ADPCM encoder	67	59	47	99	00	01
ADPCM decoder	66	77	15	02	96	02

[a]number of ALU operations
[b]ALU operations with immediate
[c]iALU ops that produce results shipped out to memory

ciated with memory operation," we mean ALU operations involved in lifetime of a value that is eventually stored in memory.

A correlation test among measurements made by the IMPACT suite indicates that bus utilization, branch issue rate, and iALU issue rate are closely correlated to IPC. The observation is intuitively understandable since a program can execute faster if it does not have to wait for memory access and can issue more ALU instructions. It turns out that other factors does not have great impact on IPC. Measurements on SPARC shows a similar results.

VI. QUANTITATIVE BENCHMARK SELECTION

The goal of benchmark selection is achieving the same or similar quality and performance in benchmarking as using all the applications in a targeted domain. The selection problem is formulated as an optimization problem and the validity of the formulation and its solution is verified by experimental results.

The basic idea of quantitative benchmark selection methodology is finding a minimum subset of applications in an application domain space using the notion of optimization. The quality of the selected benchmarks should be measured quantitatively and by maximizing the measured quality, we can find a subset which best characterizes the application domain. As an example, Figure 3 shows a conceptual domain space. Each application is numerically characterized and plotted in the application space. The example shows five candidate benchmarks indicated by shaded dots.

The benchmark selection problem is informally described as follows:

"A sufficiently large set of N applications is given and each application is *numerically* characterized by a set of relevant properties. The task is to select k applications which according to some quantitative criteria are the best representatives of all applications."

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

Quantitative benchmark selection problem

Instance: A sufficiently large set X of N applications and a constant M are given. Each application x_j is *numerically* characterized by a set of m relevant properties, x_{jl}, $l = 1, ..., m$.

Question: Is there a set Y of k applications y_i, $i = 1, ..., k$ such that $\sum_{r \in R} D(x_r) \leq M$, where $D(x_r) = \min_{\forall i} Dist(x_r, y_i)$ and $R = X - Y$?

$Dist(x_r, y_i)$ can be any distance measure, e.g. geometric $\sqrt{\sum_{l=1}^{m}(x_{rl} - y_{il})^2}$, or Manhattan $\sum_{l=1}^{m} |x_{rl} - y_{il}|$.

Determination of the constant M is not a trivial task. For practical reasons the problem is divided into two subproblems: benchmark set size determination and benchmark selection. The benchmark selection subproblem given a benchmark set size is defined as follows:

"Given the size of a benchmark set k and a set of N applications $X = \{x_i | i = 1, 2, ..., N\}$, find a set of applications $Y = \{y_j | j = 1, 2, ..., k\}$ such that

Minimize :

$$D_k = \sum_{i=1}^{|R|} d_i, \qquad (2)$$

where $R = X - Y$, $d_i = \min\{d(r_i, y_j) | j = 1, 2, ..., k\}$, and $d(r_i, y_j)$ refers to a distance function from r_i to y_j."

The size of a benchmark set is determined by an iterative test that examines the effect of selecting a certain size by measuring total distances from each application to selected applications as given by the expression (2). We continue the set size determination process until we reach a point where the set size strikes balance between the economy of benchmarks and its effectiveness. We compute and compare the total distances incrementing the

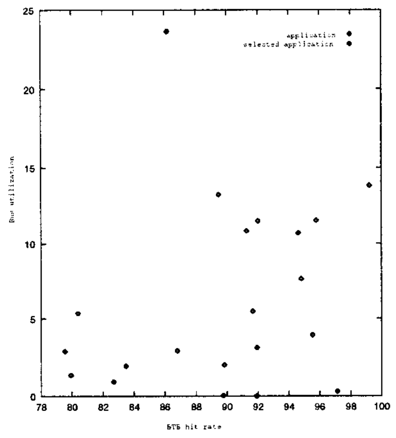


Fig. 3. An example of an application domain space and its members: Parameterized characteristics in this example are BTB hit rate and bus utilization from the IMPACT simulation results.

size one at a time until we get no significant improvement by adding more applications to the benchmark set.

The benchmark set size problem is formally given by "Given a set of N applications $X = \{x_i | i = 1, 2, ..., N\}$, select a benchmark set size k such that

$$\min\{k | \rho \leq \frac{D_{k+1}}{D_k}, k = 1, 2, ...N - 1\}, \qquad (3)$$

where D_k is given by the expression (2) and ρ refers to a target ratio."

We proved that the quantitative benchmark selection problem is NP-complete by using polynomial "local replacement" transformation from a special instance of the MIN-MAX MULTICENTER problem where the selected points are from the set of the available points.

The overall procedure for the quantitative development of benchmarks which includes a general combinatorial optimization technique known as simulated annealing [10], is given by the pseudo-code in Figure 4. We use the standard geometric cooling schedule.

```
Benchmark Selection()
  Determine relevant properties by correlation test;
  Normalize the values;
  Generate an initial solution;
  while not done do
    if equilibrium criterion is satisfied
      exit;
    if system is frozen
      exit;
    obtain the next solution;
    if the new solution is better than older one
      Accept the new solution;
    else
      Accept the new solution with standard probability;
  enddo;
endSelection();
```

Fig. 4. A pseudo-code of simulated annealing-based benchmark selection algorithm

The benchmark set is validated using the resubstitution technique [5]. A number of randomly chosen applications are eliminated from the application collection from which the original benchmark set is selected. And then a new benchmark set is selected. By comparing the new benchmark set with the original by the weighted bipartite matching technique, we can statistically validate the selected benchmark set. We use *the Hungarian Method* [12] for the matching problem.

VII. Selected Applications and Experimental Evaluation of Benchmarks

We select a subset of applications using parameters obtained on SPARC. Instruction count, IPC, miss rates, and bus traffic used to characterize applications. Out of 21 applications, 6 applications are selected: G.721 encoder, Pegwit decryption, PGP encryption, Texgen, EPIC encoder, and MPEG decoder.

Evaluation of the selected media benchmarks is conducted by using them to drive a system-on-a-chip synthesis experiment. If the resulting systems are the same then we know that the suite demonstrates value to the practice of designing embedded systems.

The synthesis process adopted here is similar to those currently being pursued in the CAD research community [11]. The fundamental goal of this experiment is to evaluate the usefulness of the selected benchmarks, not to present a fundamental new approach to system synthesis.

One of the most pressing demands on embedded system designers is to reduce cost, in large part through reducing die size. Because of this concern, we have decided to focus on a system with a simple single-issue RISC processor core and on-chip cache memories. The synthesis experiment will optimize the cache architecture in order to maximize the ratio of performance to cost. The equation used to evaluate performance/cost is:

$$\frac{Performace}{Cost} = \frac{IC}{(Core + Cache) \times (IC + M \times P)}, \qquad (4)$$

where IC is the instruction count, Core the area of processor core, Cache the cache area, M the cache miss count, and P the cache miss penalty.

By effectively normalizing to an IPC, this expression avoids the problem of application weighting that would be introduced with a direct Delay*Cost measure. The SpixTools are used to measure the raw instruction count for each application. The core is based on the IBM 40x PowerPC cores, with an estimated size of $8mm^2$ in $0.5um$ technology. Cache area is calculated using the Cache Design Tools [7]. Two models exist for miss penalty: critical-word first and full-line blocking. We assume that the external memory bus is wide enough to satisfy a fundamental read operation with a single transaction. Thus, the penalty for a miss under the critical word first model is simply the main memory latency. For the experiments here we use a value of 10 clocks.

TABLE IV
ACHIEVABLE IPC ESTIMATES FOR EACH SYNTHESIZED SYSTEM WHEN ALL COLLECTED APPLICATIONS ARE USED

D-cache	I-cache				
	1 KB	2 KB	4 KB	8 KB	16 KB
1 KB	0.468654	0.484169	0.496126	0.508322	0.517997
2 KB	0.53417	0.554503	0.570247	0.586422	0.599339
4 KB	0.615828	0.643011	0.664278	0.686332	0.704092
8 KB	0.688917	0.723113	0.750119	0.778361	0.801282
16 KB	0.721115	0.758672	0.788456	0.81972	0.845183

TABLE V
PERFORMANCE/COST ESTIMATES FOR EACH SYNTHESIZED SYSTEM WHEN ALL COLLECTED APPLICATIONS ARE USED

D-cache	I-cache				
	1 KB	2 KB	4 KB	8 KB	16 KB
1 KB	0.0352662	0.0309254	0.0243599	0.0170932	0.0119559
2 KB	0.0341191	0.0307664	0.025084	0.0182657	0.0131168
4 KB	0.0302374	0.0282848	0.0242049	0.0186424	0.0139693
8 KB	0.0231661	0.0225232	0.020375	0.0168523	0.013405
16 KB	0.0166441	0.0166039	0.0156431	0.0137135	0.0115207

This experiment is focused on sizing the cache memory. Each cache is direct mapped with 16-byte lines. The instruction and data caches were independently sized between 1 KB and 16 KB. The area cost for each cache configuration can be fixed statically, while each application has a different performance level. Performance measures were calculated for each application on each of the 25 cache configurations, using the base *spix* instruction counts and cache performance numbers generated by DineroIII. The Performance/Cost numbers for each configuration were aggregated through the arithmetic mean. The results of this experiment are shown in Table IV and V for all applications and Table VI and VII for the selected benchmarks. The discrepancies between the two cases are less than 4% for both IPC numbers and Performance/Cost numbers. The benefits of having a smaller set of quantitatively selected applications to drive the synthesis effort include greatly reduced simulation time and a higher level of confidence that the synthesis result is reasonably good.

VIII. CONCLUSION

A benchmark selection algorithm is introduced and an experimental benchmark set is selected. Evaluation of the selected media benchmarks is conducted by using them to drive a system-on-a-chip synthesis experiment. A traditional experiment was conducted to optimize system Performance/Cost across a range of cache configurations. The resulting optimization surfaces for all application and the selected benchmarks implies the benchmark set is very effective and efficient in driving the synthesis effort, thus demonstrating the applicability and usefulness.

TABLE VI
ACHIEVABLE IPC ESTIMATES FOR EACH SYNTHESIZED SYSTEM WHEN ALL SELECTED BENCHMARKS ARE USED

D-cache	I-cache				
	1 KB	2 KB	4 KB	8 KB	16 KB
1 KB	0.456335	0.470323	0.492735	0.497136	0.502535
2 KB	0.529407	0.549042	0.566279	0.566482	0.584131
4 KB	0.612913	0.641565	0.667093	0.697473	0.709106
8 KB	0.704523	0.733764	0.769245	0.792122	0.818732
16 KB	0.737968	0.769901	0.797947	0.824088	0.852012

TABLE VII
PERFORMANCE/COST ESTIMATES FOR EACH SYNTHESIZED SYSTEM WHEN SELECTED BENCHMARKS ARE USED

D-cache	I-cache				
	1 KB	2 KB	4 KB	8 KB	16 KB
1 KB	0.0328342	0.0287636	0.0227204	0.0160445	0.0111374
2 KB	0.0325374	0.0293537	0.0240297	0.0176446	0.0125651
4 KB	0.0300942	0.0282212	0.0243075	0.018945	0.0140687
8 KB	0.0240271	0.0234779	0.0214378	0.0180162	0.0141989
16 KB	0.0170331	0.0170685	0.0162282	0.0144558	0.0120227

REFERENCES

[1] 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.

[2] B. F. Cmelik. SpixTools: Introduction and user's manual. Technical Report TR-93-6, Sun Microsystems Laboratories, 1993.

[3] R. F. Cmelik and D. Keppel. Shade: A fast instruction-set simulator for execution profiling. Technical Report SMLI 93-12, UWCSE 93-06-06, Computer Science and Engineering, University of Washington, 1993.

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

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

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

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

[8] M. D. Hill. Test driving your next cache. *Magazine of Intelligent Personal Systems (MIPS)*, pages 84–92, August 1989.

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

[10] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[11] D. Kirovski, C. Lee, W. H. Mangione-Smith, and M. M. Potkonjak. Application-driven synthesis of core-based systems. In *Proceedings of ICCAD*, 1998.

[12] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[13] P. Lapsley and J. Bier. DSP benchmarks: Methodology and results. In *Proceedings of the 5th International Conference on Signal Processing Applications and Technology*, volume 1, pages 871–876, Dallas, TX, USA, October 1994. DSP Associates.

[14] M. A. R. Saghir, P. Chow, and C. G. Lee. Application-driven design of DSP architectures and compilers. In *International Conference on Acoustics, Speech, and Signal Processing*, 1994.

[15] D. Shear. EDN's DSP benchmarks. *EDN*, pages 126–148, September 1988.

[16] V. Zivojnovic, J. Martinez Velarde, C. Schlager, and M. Meyr. DSPstone: A DSP-oriented benchmarking methodology. In *Proceedings of the 5th International Conference on Signal Processing Applications and Technology*, volume 1, pages 715–720, Dallas, TX, USA, October 1994.