MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems

Chunho Lee[1], Miodrag Potkonjak[1] and William H. Mangione-Smith[2]
The Departments of Computer Science[1] and Electrical Engineering[2]
The University of California at Los Angeles
{leec,miodrag}@cs.ucla.edu, billms@ee.ucla.edu

Abstract

Over the last decade, significant advances have been made in compilation technology for capitalizing on instruction-level parallelism (ILP). The vast majority of ILP compilation research has been conducted in the context of general-purpose computing, and more specifically the SPEC benchmark suite. At the same time, a number of microprocessor architectures have emerged which have VLIW and SIMD structures that are well matched to the needs of the ILP compilers. Most of these processors are targeted at embedded applications such as multimedia and communications, rather than general-purpose systems. Conventional wisdom, and a history of hand optimization of inner-loops, suggests that ILP compilation techniques are well suited to these applications. Unfortunately, there currently exists a gap between the compiler community and embedded applications developers. This paper presents MediaBench, a benchmark suite that has been designed to fill this gap. This suite has been constructed through a three-step process: intuition and market driven initial selection, experimental measurement to establish uniqueness, and integration with system synthesis algorithms to establish usefulness.

1 Introduction

Modern advances in compiler technology for instruction-level parallelism (ILP) have significantly increased the ability of compilers to capitalize on the opportunities for parallel execution that exist in many programs. Key technologies, such as trace scheduling [1], superblock scheduling [2], hyperblock scheduling [3], and software pipelining [4] are in the process of migrating from research labs to product groups. While a significant opportunity exists for further advances, the current state-of-the-art has been shown to be rather effective on a wide range of applications.

At the same time, a number of new microprocessor architectures have been introduced that present the hardware structures desired by most ILP compilers. For example, while Texas Instruments considers the new TMS320C6X ('C6X) to be a DSP [5], the architecture is similar to the Multiflow Trace [6]. Key features of the 'C6X include predicated instruction execution, a VLIW ISA, and a split register file. Example features from other architectures include multi-gauge arithmetic (or variable-width SIMD) found in the family of MPACT architectures from Chromatic [7] and the (now defunct) designs from MicroUnity [8]. The new class of mediaprocessor architectures [9] combines some of the features of DSP devices (e.g. saturating arithmetic and the ability to issue multiple operations concurrently) with those of more general purpose processors which make them good targets for compilation (e.g. large regular register files and modern models for condition codes). While these devices seem to be good targets for high-level compilation, the architectures have features that clearly are geared toward multimedia and communications systems.

Unfortunately, the vast majority of ILP research has focused on general-purpose computing, and in particular the integer SPEC benchmark [10]. While these applications have provided a good vehicle for directing existing research, they do not capture all of the essential elements of modern embedded multimedia and communications applications. We have developed the MediaBench suite to address this need. The initial goals of MediaBench are to:

1. Accurately represent the workload of emerging multimedia and communications systems.
2. Focus on portable applications written in high-level languages, as processor architectures and software developers are moving in this direction.
3. Precisely establish the benefits of MediaBench compared to existing alternatives, e.g. integer SPEC.
4. Develop a tool that is effective for system evaluation as well as system synthesis.

For the majority of these applications, current development involves hand optimization of assembly language routines for the most critical inner loops. This approach is also needed for the current class of ILP architectures with SIMD structures. The key to implementing these hand optimizations is identifying codes without loop carried dependencies, i.e. those that are vectorizable, and then applying software pipelining. These two issues are well understood by modern ILP compilers. Once these know techniques become part of the tool chain it will simply a matter of time and education before hand optimization becomes as outdated for these applications as it has for general-purpose computing.

The remainder of this paper is organized as follows. Section 2 discusses the relevant previous work. Section 3 presents the approach used for selecting components of MediaBench, and discusses the essential components of each application. Section 4 contrasts fundamental execution characteristics of MediaBench and the integer SPEC suite. Both of these benchmark suites are used to drive a synthesis experiment in section 5, which validates the need for and effectiveness of the overall approach. Finally, section 6 presents some concluding remarks.

2 Previous Work

Existing high-level benchmarks can generally be broken down into application domains. The SPEC benchmarks were specifically developed to assist in commercial evaluation and marketing of desktop computing systems [11]. Database systems are typically evaluated using the TPC benchmarks [12]. Similar examples exist for Windows applications, network performance and I/O. However, no effective high-level benchmark exists for either general embedded systems or applications that have a strong DSP component. The benchmark which is most commonly cited for embedded systems performance is Dhrystones [13], which is known to be almost completely unrepresentative of any actual workload. In fact, Dhrystones is of such little value that it has been abandoned for use in general-purpose systems, although that was the original target domain. Performance evaluation of embedded systems is further complicated by the common use of Dhrystones/mW. Power consumption measurements are strongly dependent on external memory traffic, bus loading and the computation workload. These problems are exacerbated by the fact that Dhrystones is a small benchmark and does not stress the memory system. Consequently, the metric Dhrystones/mW is the ratio of two unreliable metrics.

Most DSP system evaluation has been focused on assembly language or the use of small kernels, e.g. DSPstone [14]. Saghir et al. developed a mix of low level kernels and small applications with the goal of compiler evaluation and system synthesis [15]. Their suite emphasizes kernel codes and low-level filter operations much more than MediaBench. Furthermore, no clear comparison is made to an existing benchmark suite to establish the value of the new code base.

A number of researchers have used benchmark suites to drive automatic synthesis of embedded systems [16, 17]. We know of no study that evaluated the distinctiveness of a benchmark suite by comparing it to an existing option. Without this step, one cannot evaluate the unique benefits of a particular suite.

MediaBench is first suite focus on complete applications for multimedia and communications systems, as well as the first to use only high-level language in order to stress compilation technology. Furthermore, this paper presents a philosophy for benchmark design that is more rigorous than the ad hoc methods use previously.

3 MediaBench Components

MediaBench is composed of complete applications coded in high-level languages. All of the applications are publicly available, making the suite available to a wider user community. MediaBench 1.0 contains 19 applications culled from available image processing, communications and DSP applications. However, the MediaBench suite is an evolving tool that will be adjusted and augmented as more codes that are representative become available. The current components include:

JPEG: JPEG is a standardized compression method for full-color and gray-scale images. JPEG is lossy, meaning that the output image is not exactly identical to the input image. Two applications are derived from the JPEG source code; cjpeg does image compression and djpeg, which does decompression.

MPEG: MPEG2 is the current dominant standard for high-quality digital video transmission. The important computing kernel is a discrete cosine transform for coding and the inverse transform for decoding. The two applications used are mpeg2enc and mpeg2dec for encoding and decoding respectively.

GSM: European GSM 06.10 provisional standard for full-rate speech transcoding, prI-ETS 300 036, which uses residual pulse excitation/long term prediction coding at 13 kbit/s. GSM 06.10 compresses frames of 160 13-bit samples (8 kHz sampling rate, i.e. a frame rate of 50 Hz) into 260 bits.

G.721 Voice Compression: Reference implementations of the CCITT (International Telegraph and Telephone Consultative Committee) G.711, G.721 and G.723 voice compressions.

PGP: PGP uses "message digests" to form signatures. A message digest is a 128-bit cryptographically strong one-way hash function of the message (MD5). To encrypt data, it uses a block-cipher IDEA, RSA [18] for key management and digital signatures.

PEGWIT: A program for public key encryption and authentication. It uses an elliptic curve over GF(2^{255}), SHA1 for hashing, and the symmetric block cipher square [18].

Ghostscript: An interpreter for the PostScript language. The single application for Ghostscript is gs, which does file I/O but no graphical display.

Mesa: Mesa is a 3-D graphics library clone of OpenGL. All display output functions were removed from the library and demo programs included in the package. Three applications are used: *mipmap* [19] which executes fast texture mapping using precomputed filter results, *osdemo* which executes a standard rendering pipeline, and *texgen* which generates a texture mapped version of the Utah teapot.

RASTA: A program for speech recognition that supports the following techniques: PLP, RASTA, and Jah-RASTA. The technique handles additive noise and spectral distortion simultaneously, by filtering the temporal trajectories of a non-linearly transformed critical band spectrum.

EPIC: An experimental image compression utility. The compression algorithms are based on a bi-orthogonal critically sampled dyadic wavelet decomposition and a combined run-length/Huffman entropy coder. The filters have been designed to allow extremely fast decoding without floating-point hardware

ADPCM: Adaptive differential pulse code modulation is one of the simplest and oldest forms of audio coding.

A web page (accessed through http://www.icsl.ucla.edu/~billms) has been constructed that contains all of the information necessary for acquiring, understanding and using the MediaBench suite. A precise set of input test files has been selected for each application.

4 Performance Characteristics

The goal of this phase of the project is to empirically test whether the MediaBench suite is quantitatively different from SPECint for some set of metrics which architects generally agree are important. This is the first step in validating the intuition behind our hypothesis. SPECint 1995 was selected because it is the most credible alternative, though in principle this technique of testing for uniqueness can be applied to any other benchmark suite. The JPEG code is common to both benchmark suites.

Performance evaluation began by executing each application under two different execution environments on two different processor architectures. The IMPACT tool suite [20] was used to collect execution characteristics for the HPPA processor architecture. We used a single-issue processor along with direct-mapped 16KB caches with 32-byte lines. Impact provides cycle-level simulation of the processor.

It has been widely reported that SPECint does not stress instruction caches [21]. Nonetheless, we expected MediaBench to provide even less stress since the applications were generally intended for embedded execution. The raw performance numbers (Figure 1) indicate that while both suites achieve extremely high instruction cache hit rates, MediaBench is better able to capitalize on the available caches. This phenomenon is highlighted by sorting the instruction cache hit rates along with the mean hit rates. Partly because of the effectiveness of caching for both suites, the standard deviation intersect; in fact, the SPECint standard deviation covers the MediaBench mean value. None the less, the instruction cache hit rates are statistically different under the students T-test.

Data caches are more effective for reads on MediaBench than SPECint, while they are less effective for writes. However, the summary variance is high, as is the variance for writes. This is due in part to three applications: rawcaudio, rawdaudio and compress. On the one hand, all three of these applications generate streams of data so it is not surprising that write hit rates would be rather low. However, a number of the MediaBench applications also produce write streams, and yet achieve much higher hit rates; e.g. PGP and Pegwit. The data cache does not do write allocate, although there is a combining write buffer. This phenomenon can be seen in the bus utilization for the three applications, which is relatively low. Data cache read hits show a statistical difference between MediaBench and SPECint, though there is none for either writes or the summary performance.

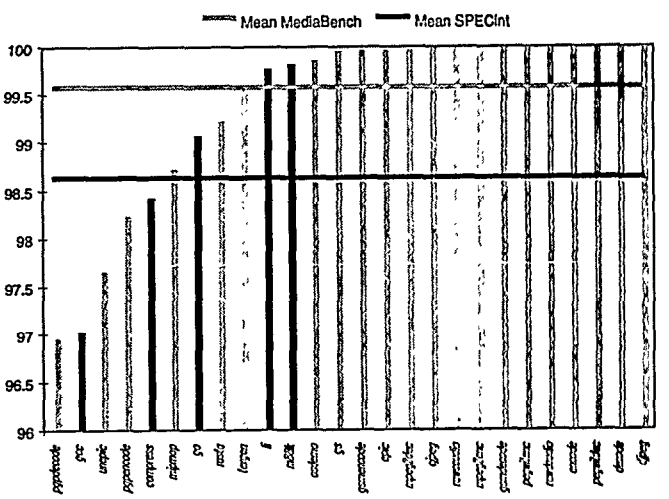

Figure 1: Instruction cache variation

Our initial intuition for MediaBench was that instruction caches would work well and data references would be cleanly divided into highly cacheable constants and uncachable data streams. The combined cache results seem to validate these expectations. We had no expectations for the trend in bus utilization, since the various cache effects serve to both increase (data streams) and decrease (instructions and data constants) memory traffic. The data presents a surprising result: the difference in mean bus utilization between MediaBench and SPECint is statistically significant. Figure 2 graphs the sorted values of bus

utilization along with the two mean values. SPECint requires almost 300% more bus bandwidth than MediaBench. Only pegwitdec surpasses the mean for SPECint; this traffic appears to be a direct consequence of relatively low instruction cache hit rates. None of the SPECint applications matches the mean bus utilization for MediaBench. This result matches the design of many embedded processor systems with relatively narrow memory buses and fewer memory devices.

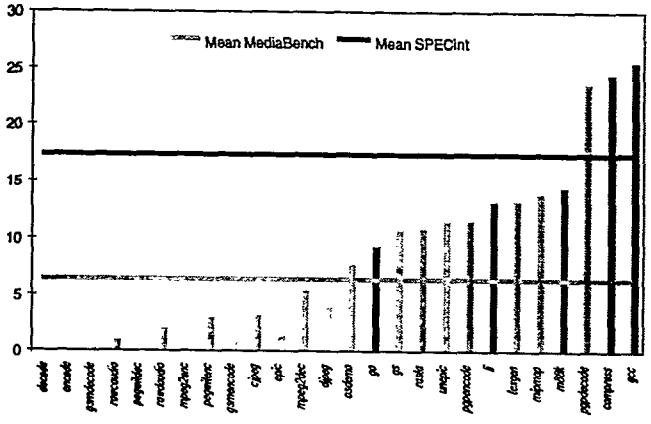

Figure 2: Bus utilization variation

DSP and communications codes are generally characterized by more time spent in loop-intensive structures, and thus one may expect a higher rate of branch instructions. In fact, the branching rates are approximately equal for the two suites, with large overlap when considering the variance. A similar result is seen with the utilization of integer ALUs. Neither branch nor integer ALU instruction rates shows a statistically significant difference in means.

Figure 3 graphs the sorted IPC rates and the two means. The data matches the traditional expectation of DSP applications, i.e. a large amount of ILP. While the means appear to be relatively similar, the variance is low and there is a strongly significant statistical difference between mean values.

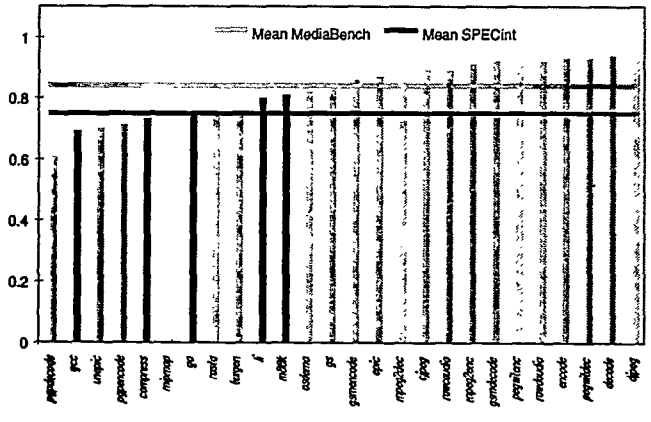

Figure 3: IPC variation

In summary, nine metrics were selected based on availability from the performance tools and intuition. Of these nine, four showed means that differed between MediaBench and SPECint to a degree that is statistically significant.

5 Synthesis Validation

We believe that MediaBench is a unique and useful suite from an important and underrepresented applications area. The runtime characteristics presented in the last section establish that there are clear statistical differences between MediaBench and the next most reasonable benchmark suite: SPECint. Our final component for evaluating MediaBench is to use it along with SPECint to drive a system-on-a-chip synthesis experiment. If the resulting systems are functionally different we will know that the suite adds 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 [22, 23]. The goal of this experiment is to evaluate the usefulness of the MediaBench suite, 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 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:

$$\frac{Performance}{Cost} = \frac{InstructionCount}{(CoreArea + CacheArea) * (InstructionCount + Misses * MissPenalty)}$$

By normalizing to an IPC, this expression avoids the problem of application weighting that would be introduced with a direct Delay*Cost measure. It should be noted that while application weighting is an appropriate technique for synthesizing a specific embedded system, MediaBench contains applications that likely would not be combined together (e.g. Pegwit and PGP). Additionally, because most of these codes operate on real-time data streams, the runtime is highly dependent on the size of the available test files.

The processor core is based on the IBM 40x PowerPC cores, with an estimated size of 8 mm^2 in 0.5um technology [24]. Cache area is calculated using the Cache Design Tools [25]. 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 is simply the main memory latency. For the experiments here we use a value of 10 clocks.

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 1KB and 16KB. 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. By itself, the inverse cost function produces a peak at 1KB and 1KB, with a drop down to 16KB and 16KB. At the same time, the Performance surface shows a peak at 16KB/16KB and is minimized at 1KB/1KB, though the function is more complicated than the inverse cost. Both of these functions are monotonic.

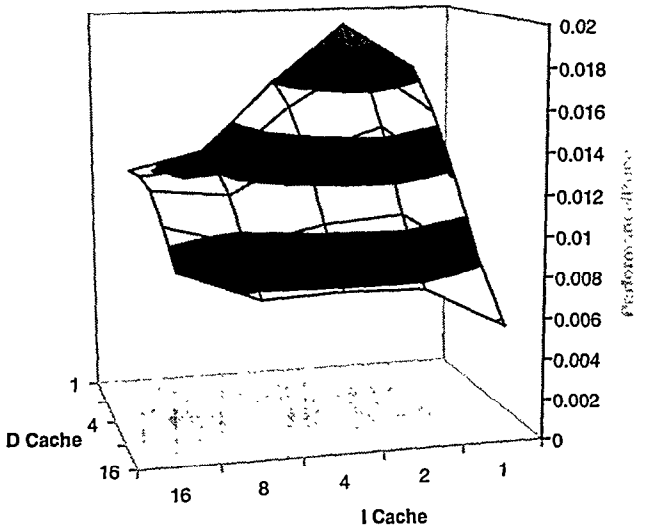

Figure 4: SPECint Performance/Cost vs. cache size

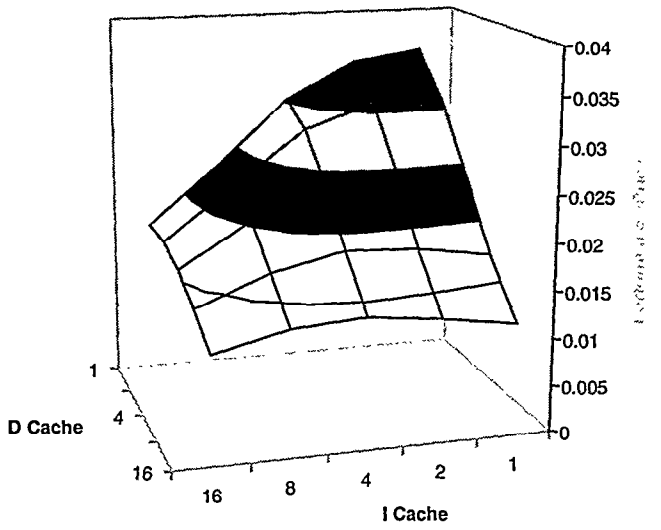

Figure 5: MediaBench Performance/Cost vs. cache size

The Performance/Cost numbers for each configuration were aggregated through the arithmetic mean. The results of this experiment are shown in Figure 4 for SPECint and Figure 5 for MediaBench. The Performance/Cost surface for SPECint reflects the benefits of increased instruction cache. The optimal design point is a 2KB instruction cache with a 1KB data cache. Additionally, the 2KB instruction cache is always a superior choice than the corresponding 1KB instruction cache. The surface also shows a slight upturn when the instruction cache is increased from 8KB to 16KB for several sizes of data caches, as a set of applications cache their working sets. None of these cache configurations approaches the optimal point, however. Clearly, for SPECint the performance benefits of large caches do not compensate for the increased cost. It is important to note that the Performance/Cost for SPECint is non-monotonic.

The Performance/Cost surface for MediaBench reflects the reduced stress on both instruction and data caches. While performance does climb for increased cache size, it is never able to compensate for the additional cost. Consequently, the smallest cache configuration (1KB and 1KB) proves to be optimal. Further experiments are under way to investigate the design space covered by smaller caches.

If SPECint were used to synthesize an embedded system for multimedia and communications applications, such as that represented by the MediaBench applications, the system designers would invest too much in instruction caches. The resulting chip would be 18% larger than the optimal design that is synthesized from MediaBench.

6 Conclusions

Significant advances in ILP compilation have resulted in tools that effectively identify and extract parallelism from within applications. At the same time, a number of new microprocessor architectures have been introduced that incorporate features which are well matched to these compilers. Most of these microprocessors are targeted to new-media applications, combining DSP and communications tasks with multimedia services. Unfortunately, most ILP compilers have been tested and developed in the context of general-purpose applications, particularly SPECint, rather than these embedded applications. A new benchmark suite is needed to capture the essential characteristics of these multimedia and communications applications.

The MediaBench suite fills this need. MediaBench is composed of full applications, not toy programs or code kernels. Each component of MediaBench is available through the Internet. Furthermore, each of these applications is coded in a high level language and has been compiled by multiple independent compilers for multiple processor architectures.

The resulting performance shows different characteristics from the most common alternative suite, i.e. SPECint. In particular, the performance difference is statistically significant in at least four important areas: achieved instructions-per-clock, instruction cache hit rate, data cache read hit rate, and memory bus utilization.

The final step in validating MediaBench involved using it and SPECint to drive a system synthesis experiment. A traditional experiment was conducted to optimize system Performance/Cost across a range of cache configurations.

The resulting optimization surface for SPECint and MediaBench showed significantly different characteristics, and resulted in different system configurations. In particular, the instruction cache is two times too large for the system synthesized by SPECint.

Acknowledgements: This work was supported by DARPA under contract F04701-97-C-0010. The authors also greatly appreciate the assistance of Professor W. M. Hwu's IMPACT team at UIUC.

References

[1] J. A. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," *IEEE Transactions on Computing*, vol. C-30, pp. 478-490, 1981.

[2] 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.

[3] 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," Proc. of Micro 25, 1992.

[4] P. Y. Hsu, "Highly Concurrent Scalar Processing," : Coordinated Science Laboratory Report #CSG-49, University of Illinois at Urbana-Champaign, 1986.

[5] J. Turley and H. Hakkarainen, "TI's New 'C6x DSP Screams at 1,600 MIPS," in *The Microprocessor Report*, vol. 11, 1997, pp. 14-17.

[6] 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," Proc. of Architectural Support of Programming Languages and Operating Systems, 1982.

[7] P. Kalapathy, "Hardware-Software Interactions on MPACT," in *IEEE Micro*, vol. 17, 1997, pp. 20-26.

[8] C. Hansen, "MicroUnity's MediaProcessor Architecture," in *IEEE Micro*, vol. 17, 1997, pp. 34-41.

[9] R. B. Lee and M. D. Smith, "Media Processing: A New Design Target," in *IEEE Micro*, vol. 17, 1997, pp. 6-9.

[10] B. Case, "SPEC95 Retires SPEC92," in *The Microprocessor Report*, vol. 9, 1995.

[11] SPEC, "SPEC Benchmark Suite Release 1.0," SPEC 1989.

[12] T. P. P. C. (TPC), "TPC Benchmark A," ITOM International Co., Los Altos 1989.

[13] R. P. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark," *Communications of the ACM*, vol. 27, pp. 1013-1030, 1984.

[14] V. Zivojnovic, J. M. Velarde, C. Schlager, and H. Meyr, "DSPstone: A DSP-Oriented Benchmarking Methodology," Proc. of Signal Processing Applications & Technology, Dallas, 1994.

[15] M. A. R. Saghir, P. Chow, and C. G. Lee, "Application-Driven Design of DSP Architectures and Compilers," Proc. of Acoustics, Speech, and Signal Processing, 1994.

[16] T. Conte and W. Mangione-Smith, "Determining Cost-Effective Multiple Issue Processor Designs," Proc. of International Conference on Computer Design, 1993.

[17] J. A. Fisher, P. Faraboschi, and G. Desoli, "Custom-Fit Processors: Letting Applications Define Architectures," Proc. of Micro 29, Paris, France, 1996.

[18] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York: John Wiley & Sons, 1996.

[19] A. Watt, *3D Computer Graphics*, 2 ed: Addison-Wesley, 1993.

[20] 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," Proc. of International Symposium on Computer Architecture, 1991.

[21] D. A. Patterson and J. L. Hennessy, "Large and Fast: Exploiting Memory Hierarchy," in *Computer Organization & Design The Hardware/Software Interface*: Morgan Kaufmann, 1994.

[22] D. Kirovski and M. Potkonjak, "System Level Synthesis of Low-Power Real-Time Systems," Design Automation Conference, Anaheim, CA, 1997.

[23] D. Kirovski, C. Lee, W. H. Mangione-Smith, and M. Potkonjak, "Application-Driven Synthesis of Core-Based Systems," Proc. of International Conference on Computer Aided Design, 1997.

[24] J. Turley, "401GF is Coolest, Cheapest PowerPC," in *Microprocessor Report*, vol. 10, 1996.

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