ABSTRACT

DASARATHAN, DINESH. Benchmark Characterization of Embedded Processors. (Under the direction of Professor Thomas M. Conte).

The design of a processor is an iterative process, with many cycles of simulation, performance analysis and subsequent changes. The inputs to these cycles of simulations are generally a selected subset of standard benchmarks. To aid in reducing the number of cycles involved in design, one can characterize these selected benchmarks and use those characteristics to hit at a good initial design that will converge faster. Methods and systems to characterize benchmarks for normal processors are designed and implemented. This thesis extends these approaches and defines an abstract system to characterize benchmarks for embedded processors, taking into consideration the architectural requirements, power constraints and code compressibility. To demonstrate this method, around 25 benchmarks are characterized (10 from SPEC, and 15 from standard embedded benchmark suites - Mediabench and Netbench), and compared. Moreover, the similarities between these benchmarks are also analyzed and presented.

Benchmark Characterization of Embedded Processors

by

Dinesh Dasarathan

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2005

Approved By:

Dr. Edward F. Gehringer	Dr. Eric Rotenberg

Dr. Thomas M. Conte
Chair of Advisory Committee

To my parents and advisor ...

Biography

Dinesh Dasarathan was born on 29th August, 1981, in Chennai, India. He graduated from the College of Engineering - Guindy, Anna University, with a Bachelor of Engineering in Computer Science in May 2003. He then enrolled in the graduate program in Computer Science at North Carolina State University in Fall 2003. With the defense of this thesis, he will receive the Master of Science in Computer Science degree.

Acknowledgements

I am deeply indebted to my advisor, Dr. Thomas M. Conte, for his support and invaluable guidance throughout my thesis. It was great fun to be part of the research group that is led by one of the most energetic and witty persons I have ever met.

Thanks to Dr. Edward F. Gehringer and Dr. Eric Rotenberg for agreeing to be on my thesis committee in spite of their tight schedule and for the valuable feedback regarding this thesis document. I would also thank Dr. Rotenberg for providing me with his simulators and benchmark executables, that helped me a lot in my research.

Thanks to Saurabh Sharma for giving valuable pointers that helped me on my experiments, Paul Bryan for helping me proofread this document and Aravindh Anantaraman for helping me integrate the WATTCH simulator with Dr. Rotenberg's version of SimpleScalar.

I would like to thank my friends Chithi, Vurumaani, Baawa, Kaakka, Jaggu and Kriii for making my graduate life pleasant and enjoyable. Finally, and most importantly, I would like to thank my parents and Jonty for their love and affection.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Processor design is a complex matter with a great number of design choices. Consider the design of a system in which the addition of new hardware might improve overall performance by executing certain instructions faster, or degrade performance by reducing the clock speed. How is it possible to know whether or not this hardware will finally contribute to a better design of the system? The answer is simple: the usefulness of the new hardware depends on the programs that the system will run. In order to maintain a general standard, designers often use benchmark programs that are representative of the programs that the system finally runs. Hence, the system's performance while running the benchmarks determines the design of the system.

The performance of a system can be simulated before it is ever built and the simulation results can be used to improve the design. Simulation involves selecting an initial design, simulating the design for each benchmark, often a lengthy process and then adjusting the design, and reiterating. If the initial design is far from what is required, the number of iterations will be large, wasting large amounts of effort and increasing the time to market. Therefore, in order to reduce the number of design iterations, it is very important to find an *initial design* that is closest to the final requirement. This is where *benchmark characterization* comes into the picture.

The premise of this thesis is that it is possible to have a reference work that contains the cost-effective designs for each benchmark, and to use these designs to determine the initial design. The most attractive method would be to define an abstract system that is general enough to include many system designs as special cases and then measure a benchmark's performance in terms of this abstract system. We call this the *benchmark's characteristics* and the process itself, *benchmark characterization* [5].

Until now, we have considered how benchmark characterization is relevant to processor design in general-purpose systems. With the surge of the embedded systems market, it is important to extend benchmark characterization to meet the design requirements of embedded processors. Embedded processor design is quite different from general processor design, with design issues such as power utilization and code compression for smaller memory requirements being important in addition to the architectural requirements of normal processors. The remaining portion of this thesis will deal with redefining the abstract model of benchmark characterization for embedded processors and also characterizing some common benchmark suites such as SPEC2000 [7], Mediabench [9] and Netbench [10] on architectural as well as embedded aspects. This thesis would also focus upon how these characteristics will affect the design of an embedded system. In addition, we shall also be studying whether these benchmark suites are similar and also decide whether they are suitable for embedded design.

Chapter 2

The Method of Benchmark Characterization

The abstract system implemented to characterize the benchmarks involves a set of simulators that take as input the run-time traces of the benchmarks, generate results and then use these results to output their characteristics. The main constituents of this abstract system are a single-pass memory simulation [3] that uses the recurrence-conflict method, a modified form of the Simplescalar simulator [1], the WATTCH toolkit [2] for Simplescalar and a byte-based Huffman encoder [8]. The traces were generated using a modified form of the functional Simplescalar simulator, and fed into the abstract system. To generate traces, the selected benchmarks were compiled with gcc at the lowest level of optimization. All but the SPEC benchmarks were run to completion. We could run only the first few billions of events of the SPEC benchmarks for their reference inputs.

All of the processor characteristics mentioned below (IPC, intermediate instruction frequency and branch characteristics) were output directly by the modified Simplescalar simulator. Of these the intermediate instruction frequency was generated by modifying the functional simulator, while IPC and branch characteristics were output by the cycle-accurate simulator. The memory system characteristics were generated using the single-pass memory simulator, while the power characteristics for each benchmark were produced using the

WATTCH toolkit coupled with the modified Simplescalar simulator. The code compression characteristics were generated using the byte-based Huffman encoder. Figure 2.1 illustrates the abstract system model.

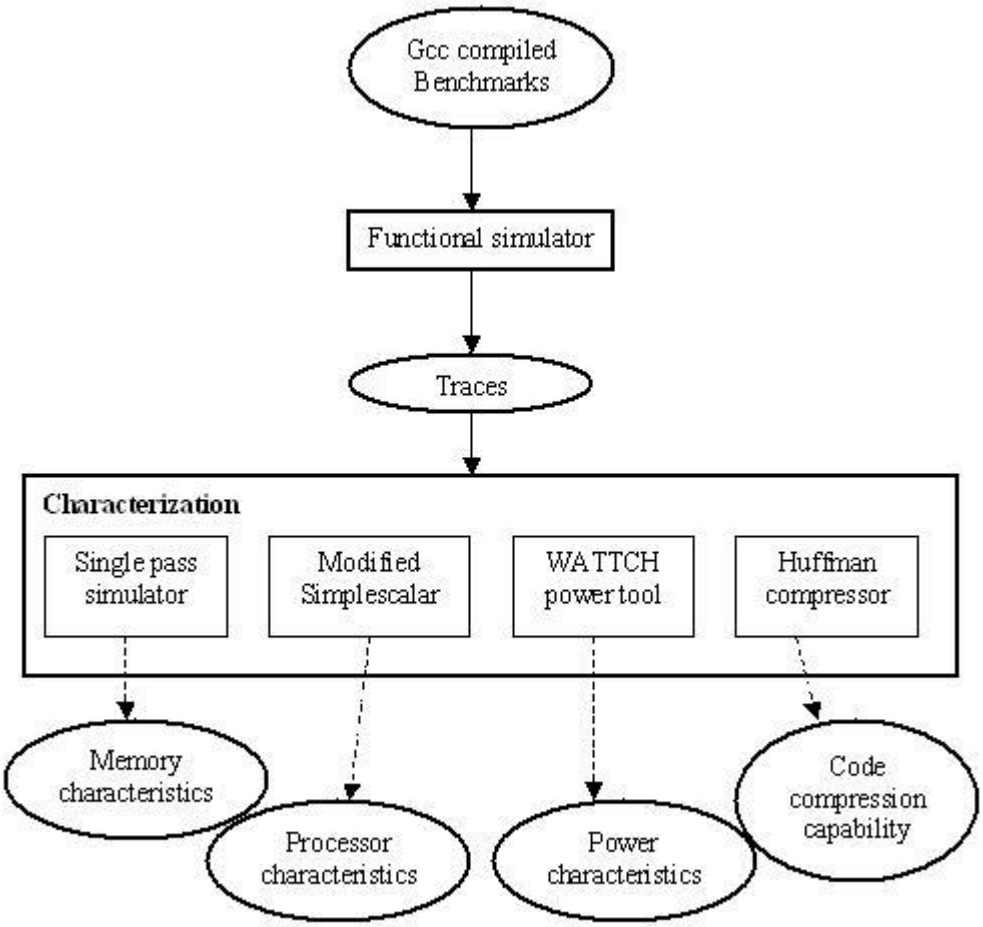

Figure 2.1: The Abstract System Model

Chapter 3

Power Characteristics

Power is a very important factor in embedded processor design. Today's embedded processors are used in applications where battery power is critical, such as cell phones, cars and other mobile devices. In order to prolong battery life, it is important to design embedded processors that satisfy low-power constraints. Embedded systems nowadays have at least two distinct modes of operation: the high-power mode and the sleep mode. (This is a simplified view, with many more intermediate modes that progressively consume less power, while executing the program less rapidly.) Techniques such as dynamic voltage or frequency scaling have also come into widespread use to reduce the overall power consumption.

The power requirements of a benchmark used during design can indicate whether different power modes are necessary in a system, and whether dynamic voltage or frequency scaling need to be applied. We have analyzed all benchmarks using the WATTCH simulator, and obtained the power consumed for each benchmark on the modified simplescalar simulator, whose configuration is discussed in section 5.2. Figure 3.1 shows the power characteristics for the cc3 model of the pipeline operating at 1000 MHz. The cc3 model implements the idea that even dormant portions of the pipeline contribute to the total power consumed. (In the case of WATTCH, dormant regions consume 10% of the power consumed by active regions.)

While designing an embedded processor the designer has to take into account the power requirements of the benchmark. Techniques such as dynamic voltage or frequency scaling must be applied for designs where power is a constraint.

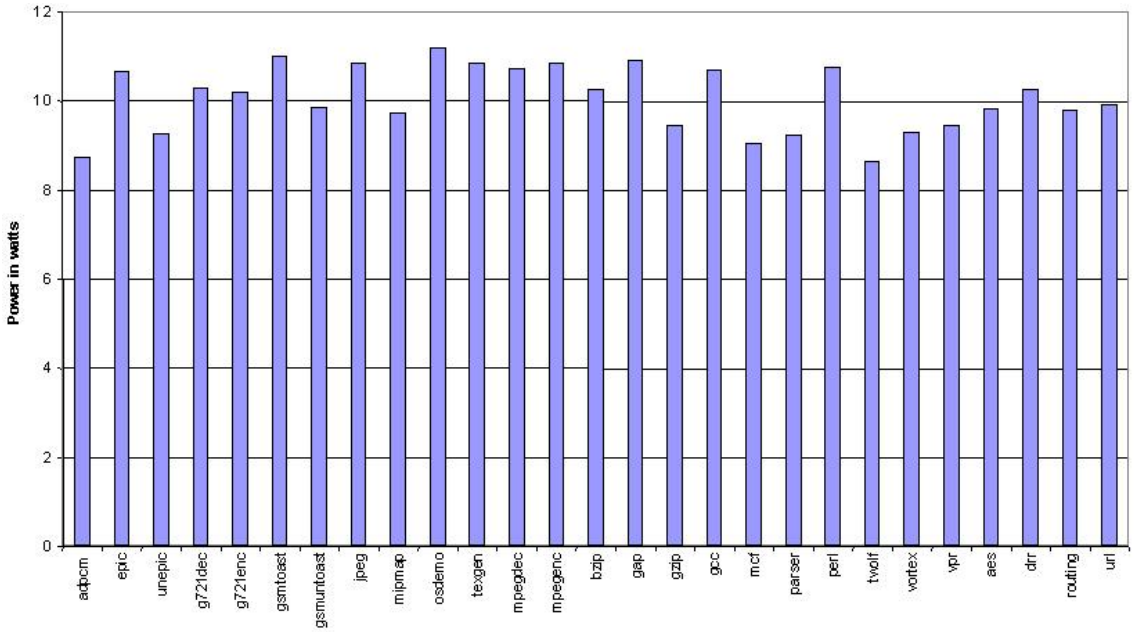

Figure 3.1: Total Power Consumed by the Benchmarks

Chapter 4

Code Compressibility

Since on-chip space and memory area are important constraints involved in embedded processor design, prior work has been done on compressing the executable image of the benchmark, so that more of it fits into the main memory. The compression ratio of a benchmark can affect main memory design. If a benchmark is highly compressible, then the designer has the freedom to manipulate the memory requirements accordingly.

There are many compression techniques available [6]. For this study the standard Huffman encoding scheme was selected. This is a byte-based method, where bytes of executable image are encoded on a per byte basis. The compression ratios of all benchmarks are indicated by Figure 4.1.

Most of the benchmarks show a compression ratio of around 45%. This implies that the designer has the advantage of reducing the main memory size by the same amount during design. But, of special importance are the benchmarks DRR and URL. The compression ratio for these benchmarks is almost negligible, since their corresponding executable images have symbols occurring at the same frequency. Therefore, the Huffman encoder that works at giving smaller codes to high frequent symbols shows a low compression ratio. For such benchmarks, the designer has to take care not to manipulate memory requirements if high performance is desired.

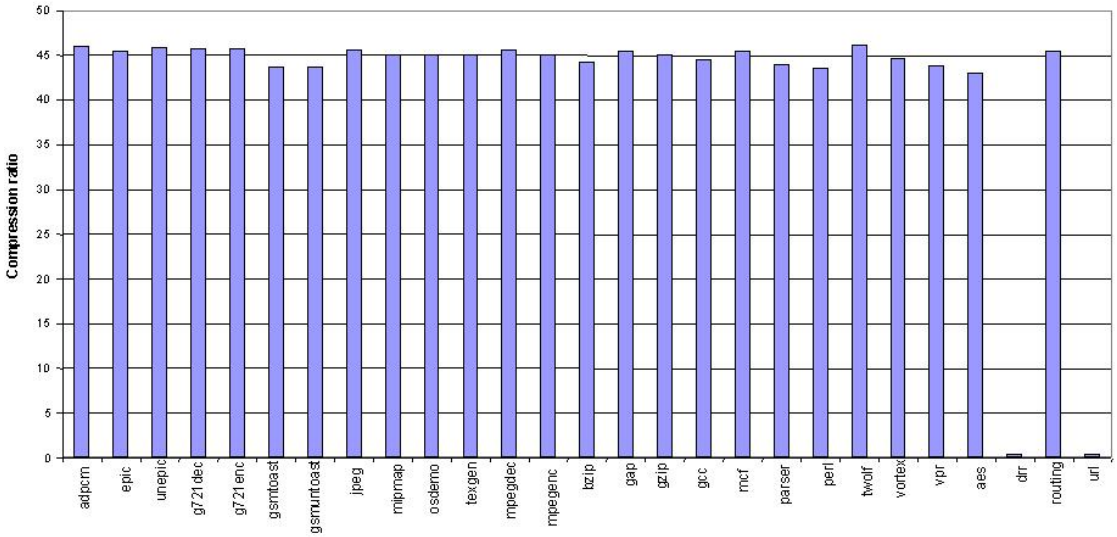

Figure 4.1: Compression Ratios for Benchmarks

Chapter 5

Processor Characteristics

5.1 Intermediate Instruction Frequencies

Processor design involves providing execution resources (e.g. registers, function units, and supporting logic) to achieve high performance. Knowing the relative importance of various operations can be extremely useful when designing a processor. Consider our example from the introduction of whether to add hardware or not: information on the relative use of different instruction classes in benchmarks could help answer this question. Such relative frequencies could also help answer how important floating-point hardware is, or whether implementing a multiply/divide unit is worthwhile. To gather the intermediate code instruction frequencies, we chose 11 categories of important instruction types (see Tables 5.1, 5.2 and 5.3) and separated intermediate instructions from the traces we obtained into these categories. (Note that "*" indicates a frequency less than 1%, and a "-" indicates no occurrence.)

Table 5.1: Mediabench Intermediate Instruction Frequencies (in %)

Benchmark	Branches	Loads	Stores	itAlu	itMul	itDiv	fpCvt	fpAlu	fpMul	fpDiv	fpSqrt
adpcm	27	7	1	65	*	*	-	-	-	-	-
epic	15	13	2	54	*	*	6	5	5	*	-
unepic	20	13	11	49	1	*	2	3	*	1	-
G721decode	23	14	4	58	1	*	-	-	-	-	-
G721encode	23	13	4	59	1	*	-	-	-	-	-
gsm toast	5	17	5	66	7	*	-	-	-	-	-
gsm untoast	18	7	4	67	4	*	-	-	-	-	-
jpeg	16	21	7	56	*	*	-	-	-	-	-
mipmap	16	22	12	38	*	*	2	7	3	*	-
osdemo	19	22	10	47	*	*	*	1	1	*	*
texgen	13	22	12	40	1	*	2	5	5	*	*
mpegdecode	12	15	4	53	*	*	3	8	5	*	*
mpegencode	17	27	2	51	*	*	1	1	1	*	*

Table 5.2: SPEC 2000 Intermediate Instruction Frequencies (in %)

Benchmark	Branches	Loads	Stores	itAlu	itMul	itDiv	fpCvt	fpAlu	fpMul	fpDiv	fpSqrt
bzip	15	21	9	55	*	*	-	-	-	-	-
gap	8	21	11	54	6	*	-	-	-	-	-
gcc	18	23	19	40	*	*	*	-	*	*	-
gzip	21	20	5	54	*	*	-	-	-	-	-
mcf	20	23	19	38	*	*	*	*	*	-	-
parser	20	28	10	41	*	1	-	-	-	-	-
perl	17	27	13	43	*	*	*	*	*	*	-
twolf	18	27	9	45	*	1	*	*	*	*	*
vortex	17	12	34	37	*	*	-	-	-	-	-
vpr	19	24	8	45	*	*	1	2	1	*	*

Table 5.3: Netbench Intermediate Instruction Frequencies (in %)

Benchmark	Branches	Loads	Stores	itAlu	itMul	itDiv	fpCvt	fpAlu	fpMul	fpDiv	fpSqrt
aes	18	17	9	54	1	1	-	-	-	-	-
drr	35	27	9	29	*	*	-	-	-	-	-
routing	21	26	10	43	*	*	-	-	-	-	-
url	14	31	12	43	*	*	-	-	-	-	-

The value of high-performance integer hardware cannot be argued, as its use represents at least 40% of the instructions for all benchmarks. Integer division, floating-point division, and floating-point conversions are hardly used by either SPEC2000 and Netbench and could safely be implemented by software for designs based upon these benchmarks. But, for design based on Mediabench, floating-point hardware becomes important for high performance. Integer multiplication is also employed by Gsm and Gap, which suggests that well-designed hardware should support integer multiplication, although perhaps not a full-scale integer multiplier. The frequency of control transfers for Gsm and Gap shows that these programs have longer basic blocks. Therefore, if the system will be running applications similar to Gsm and Gap, branch-prediction hardware is less critical.

5.2 Instruction Level Parallelism

The number of parallel instructions that are available to be executed in one cycle is another important characteristic of a benchmark. This can influence the degree of superscalar issue and also the parameters that will be used during concurrent execution, such as the number of register read and write ports. A design based on a highly parallel benchmark can have a higher degree of ports, while design based on lesser parallel benchmarks can manage with a smaller degree of superscalar issue and fewer ports.

The available parallelism in a benchmark can be best measured by running the benchmark on a simulator such as Simplescalar, and finding the IPC. A modified form of Simplescalar was used for this purpose. This version had the following configuration:

Rob size 64
IQ size 64
Fetch width 8
Dispatch width 8
Issue width 8
Issue-mem width 4
Retire width 4
I$: 128KB, 4-way set-associative, 128B line size
D$: 64KB, 4-way set-associative, 64B line size

This configuration was used as a standard in this thesis for generating all characteristics involving the Simplescalar simulator.

Figure 5.1 indicates the IPC calculated using the above configuration for all 3 benchmark suites. It can be found that Mediabench and Netbench are highly parallel due to more loops within the program, while SPEC2K comparatively has a lower number of parallel instructions. Hence, designing a highly parallel machine can be cost-viable for workloads similar to Gsm, Mpeg2, Texgen, Routing and Osdemo, but is rather wasteful if the workload is similar to benchmarks like MCF and PERL.

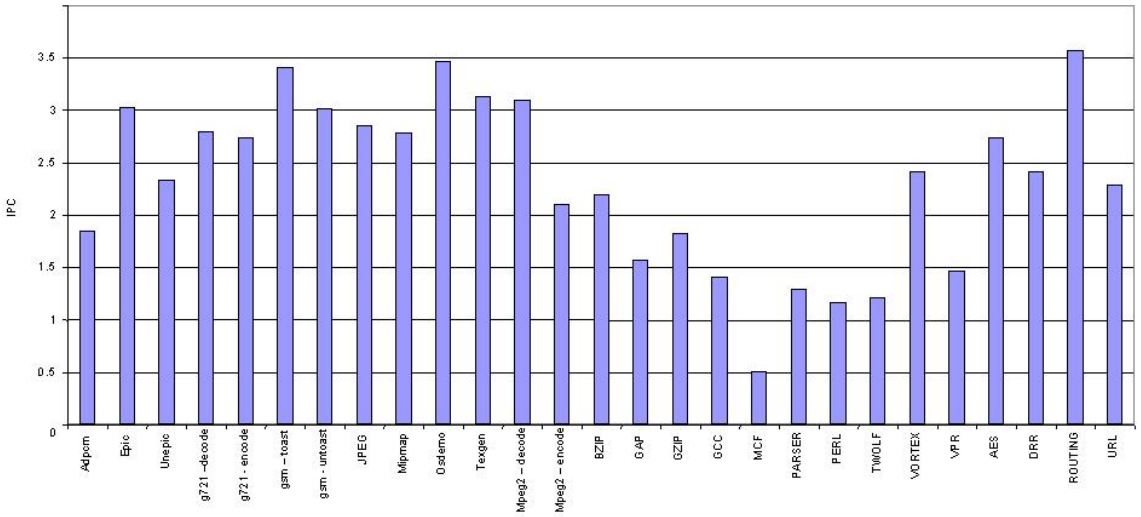

Figure 5.1: IPC of Benchmarks

5.3 Branch Characteristics

The branch characteristics of a benchmark can determine if branch-prediction hardware is critical for the design; and if so, it can indicate whether a highly-accurate, high-cost branch predictor must be used, or whether a less-accurate, lower-cost branch predictor will suffice. The branch characteristics of benchmarks can be best represented by the misprediction rates when there is no interference between branches (branch interference occurs if more than one branch maps to the same row in the branch-prediction table). To achieve zero interference between branches, very large prediction tables were used, sized so that they could accommodate one row for each value of the PC. The global branch history of the branches was collected using a GShare predictor with tables of size of 16MB, to

prevent interference.

Figure 5.2 shows the branch misprediction rates calculated using this method. It can be noted that benchmarks Mpeg Encoder and adpcm require highly accurate branch predictors, whereas benchmarks Mipmap, Aes and routing may or may not need elaborate branch predictors.

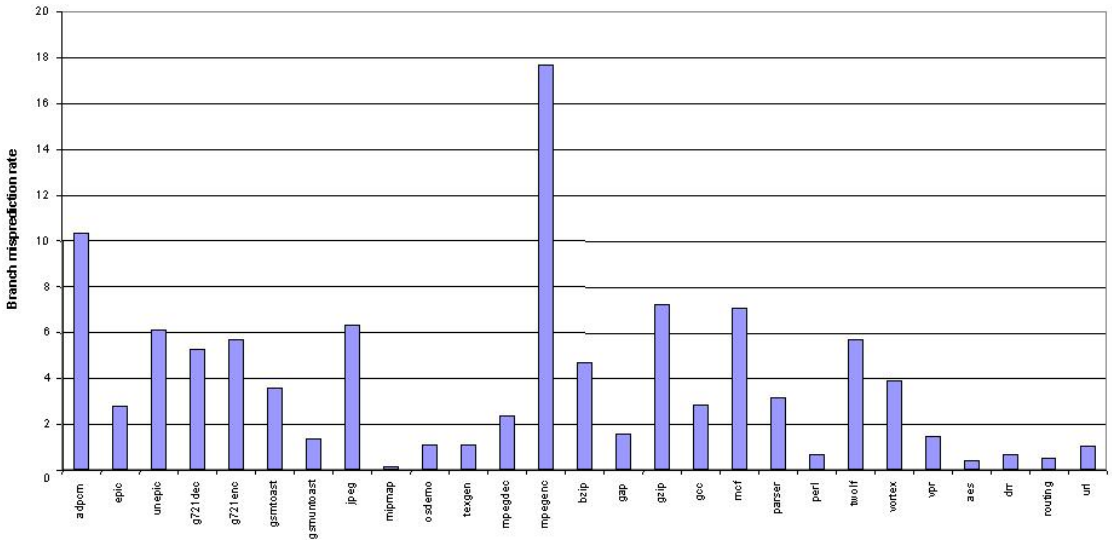

Figure 5.2: Branch Misprediction Rates for Benchmarks

Chapter 6

Memory System Characteristics

Modern memory systems consist of one or more levels of caching above a virtual memory system. The entire set of miss ratios for the benchmarks constitutes a vast amount of data. Instead of presenting that data, we present several cost-effective memory system designs for the benchmarks. We describe these designs first by discussing the top-level cache and second-level instruction and data cache requirements, then several good TLB designs, and then we close by discussing main-memory design requirements. Throughout this section we exclude the results for 4-way set-associative caches because we discovered that the required cache sizes were the same as higher associativities.

The abstract system model employs a modified single-pass algorithm [3] to record the number of recurrences and organizational misses for all cache dimensions in a design space. This algorithm takes as input the miss ratio and the block size, and gives the least expensive cache design for all associativities. If the given miss ratio cannot be met by the benchmark, then the algorithm outputs the least expensive cache design that meets the intrinsic miss ratio of the benchmark for that block size (The intrinsic miss ratio corresponds to the miss ratio for a cache with an infinite number of blocks.)

Since some of the benchmarks had very long traces, statistical trace sampling [4] was used to reduce run time, without unduly affecting the accuracy of the results.

6.1 Cache Requirements

Figures 6.1 - 6.6 show the minimum dimensions for all benchmark suites, required for top-level instruction and data caches, which are backed up by a unified second-level cache. The top-level caches use a block size of 32 bytes while the unified L2 cache uses a block size of 64 bytes. The design criterion we used for L1 caches was to select the smallest (i.e., least expensive) cache that provides a 10% miss ratio or better. Similarly, the design criterion we used for the unified L2 cache was the smallest cache that provides a maximum 1% or intrinsic miss ratio.

For L1 data caches, the required cache size ranges from 256 bytes to 16MB (Mcf). Mcf has a high miss rate since it is a classic example of pointer-chasing algorithms with huge tree structures, where reference locality is far less than other benchmarks. L1 Instruction caches have a range between 256 bytes and 16KB (gcc). The range is less here since instruction streams follow the trend of spatial locality more closely than data streams. For L2 unified caches, the required cache size ranges from 512 bytes to 4MB. It is important to note that for a few benchmarks such as mcf and epic, the L2 cache requirement is less than the L1 Data cache requirement. This is because for the unified cache, instruction streams dominate the cache characteristics and yield a lower number of misses.

Most benchmarks show a decrease in cache-size requirement as the set associativity increases. This is because misses due to cache dimension limitations account for a significant portion of the overall misses. The reduced number of reloading misses causes misses due to cache dimensions to dominate the miss ratio, thus making the benefit of increased set associativity more obvious.

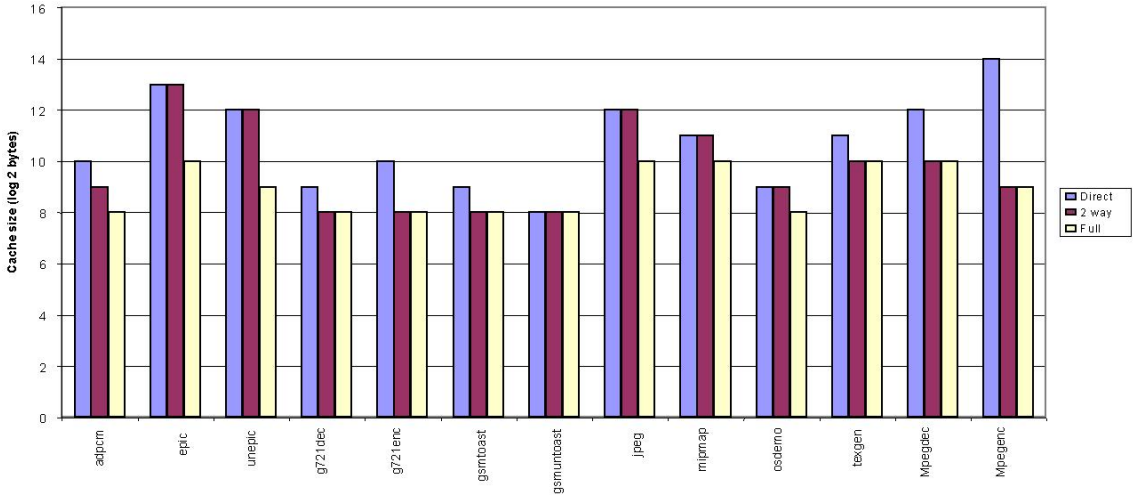

Figure 6.1: Mediabench - L1 Data Cache Design (32 byte block) for Max.10% Miss Ratio

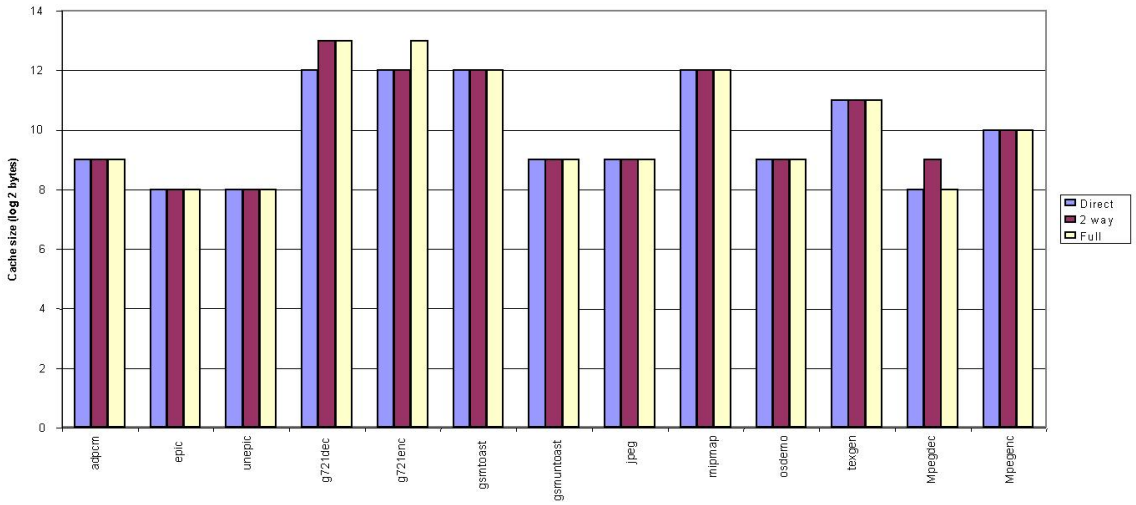

Figure 6.2: Mediabench - L1 I-Cache Design (32 byte block) for Max. 10% Miss Ratio

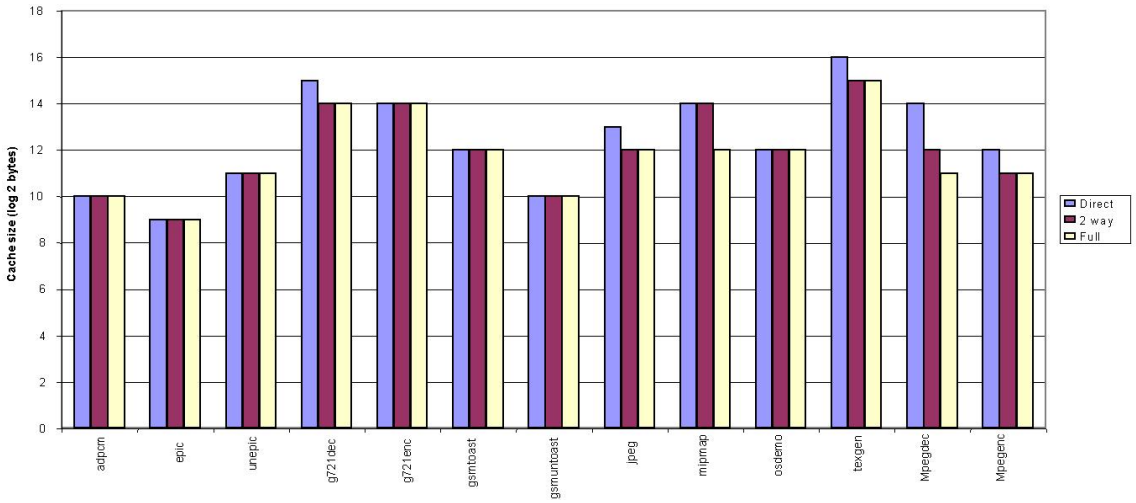

Figure 6.3: Mediabench - L2 Unified Cache Design (64 byte block) for Max 1% or Intrinsic Miss Ratio

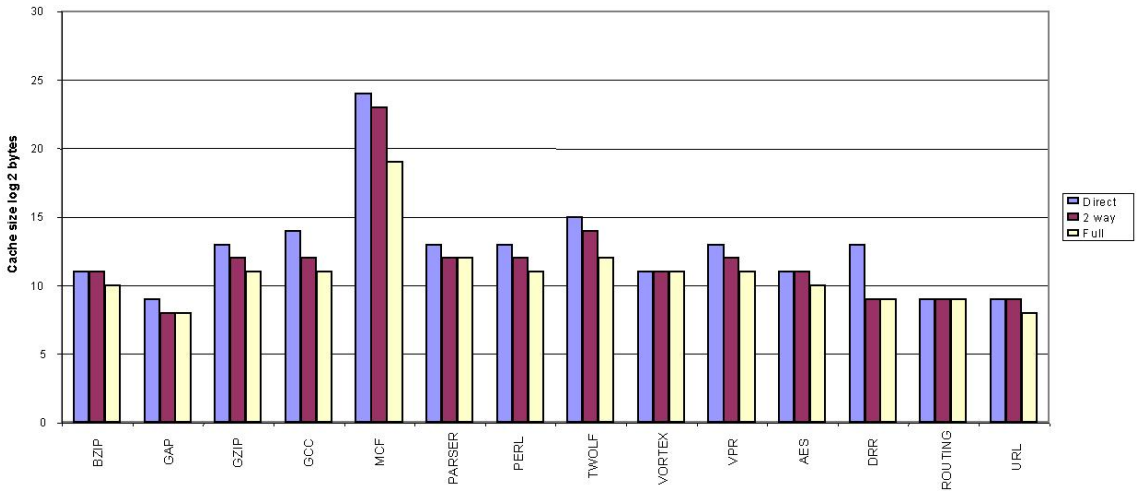

Figure 6.4: SPEC2k and Netbench - L1 Data Cache Design (32 byte block) for Max 10% Miss Ratio

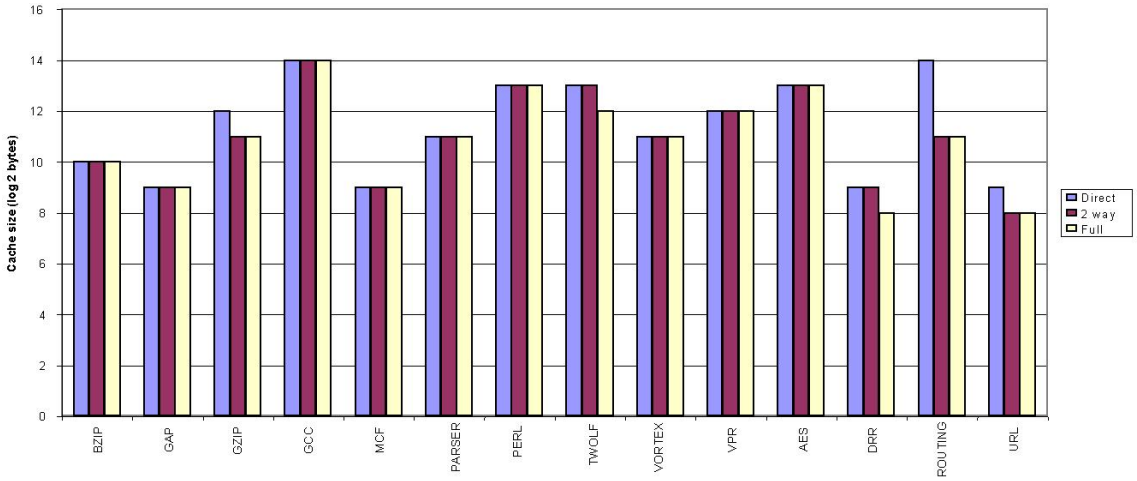

Figure 6.5: SPEC2k and Netbench - L1 I-Cache Design (32 byte block) for 10% Miss Ratio

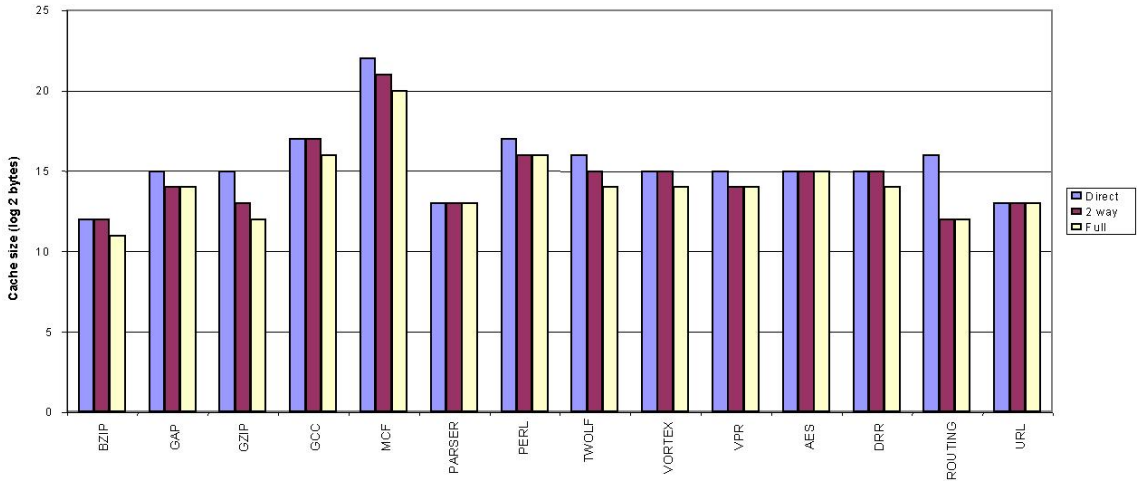

Figure 6.6: SPEC2k and Netbench - L2 Unified Cache Design (64 byte block) for Intrinsic or Max 1% Miss Ratio

6.2 TLB Requirements

TLBs are common in embedded systems because they provide memory protection, even if the memory system is purely physical. In general, the TLB caches virtual memory translation results for the frequently accessed pages in the virtual space. Without a TLB, it takes a significant number of cycles to translate a virtual address into a physical address (typically 10 to 40 cycles). On the other hand, the cost of translation can be eliminated if the translation results are found in the TLB. A small increase in the miss ratio of the TLB usually results in a significant loss in performance. Therefore, it is important to design the TLB to provide a very small miss ratio (that is, 0.1 percent or intrinsic). Tables 6.1, 6.2 and 6.3 presents the TLB dimensions required to guarantee a 0.1 percent miss ratio for each benchmark suite. (Because the size of a page-table entry is dependent upon the size of the virtual memory system, these sizes are presented in units of page-table entries instead of bytes.)

Table 6.1: TLB Requirements for Mediabench

	Page sizes	**512 byte**				**1K byte**				**2K byte**				**4K byte**		
Benchmark	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**
adpcm	32	16	8	8	16	8	8	4	8	8	8	4	4	4	4	4
epic	1024	128	128	128	512	32	16	16	256	32	16	16	128	32	8	8
unepic	1024	128	128	256	1024	64	32	32	512	64	16	16	256	64	16	8
g721decode	16384	128	128	64	8192	64	64	32	4096	32	32	32	2048	16	16	16
g721encode	16384	128	64	64	8192	64	32	32	2048	32	32	32	1024	16	16	16
gsmtoast	256	32	16	16	128	16	8	8	64	8	8	8	64	8	8	8
gsmuntoast	64	32	32	32	32	16	16	16	16	8	8	8	64	8	8	8
Jpeg	256	128	64	32	256	64	32	32	128	32	32	16	64	32	32	16
mipmap	2048	128	64	32	1024	64	64	32	512	64	32	32	256	64	32	32
osdemo	4096	512	256	256	1024	256	128	128	512	128	32	32	512	128	32	32
texgen	2048	512	512	128	1024	256	256	128	512	128	128	64	256	128	64	64
mpegdecode	256	128	64	32	256	64	32	16	256	32	16	16	2048	32	16	8
mpegencode	1024	64	64	32	512	64	32	32	1024	32	32	16	512	16	16	8

In general, the TLB size requirement decreases with the increase in page size because larger page sizes result in fewer active pages. Therefore, the TLB has to accom-

Table 6.2: TLB Requirements for SPEC2K

	Page sizes	**512 byte**				**1K byte**				**2K byte**				**4K byte**		
Benchmark	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**
bzip	16384	8192	2048	2048	8192	4096	2048	1024	4096	2048	2048	1024	2048	1024	512	512
gap	1024	256	128	128	1024	128	64	64	1024	64	64	32	1024	64	32	32
gcc	16384	16384	4096	2048	8192	1024	512	256	4096	512	256	256	2048	256	128	128
gzip	16384	1024	512	512	8192	512	256	256	4096	256	128	128	2048	256	128	64
mcf	16384	4096	4096	1024	8192	2048	2048	1024	4096	1024	1024	512	2048	512	512	512
parser	16384	2048	2048	1024	8192	2048	1024	1024	4096	1024	1024	512	2048	512	512	512
perl	8192	2048	1024	512	8192	1024	512	256	2048	512	256	256	1024	512	256	128
twolf	4096	2048	2048	2048	2048	2048	1024	1024	2048	1024	1024	512	2048	1024	512	512
vortex	2048	2048	2048	1024	1024	1024	512	256	512	128	64	64	256	64	64	32
vpr	16384	16384	8192	2048	8192	8192	4096	1024	4096	4096	2048	1024	2048	2048	512	512

modate fewer address translation results. An interesting phenomenon is that neither GCC nor MCF is the only most demanding benchmarks for TLB design. Although GCC consistently requires more than four times the cache size that VPR does, their TLB requirements are very comparable. This is because VPR accesses a comparable number of active pages but accesses a much smaller number of blocks within them. This is a good example of a demanding benchmark for TLB design that is undemanding in the context of cache design.

Table 6.3: TLB Requirements for Netbench

	Page sizes	**512 byte**				**1K byte**				**2K byte**				**4K byte**		
Benchmark	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**	**1**	**2**	**4**	**F**
aes	16384	256	256	128	8192	128	128	64	4096	128	64	32	2048	64	32	32
drr	2048	256	128	64	1024	128	128	64	512	128	64	64	256	64	32	32
routing	128	128	64	32	8192	128	32	32	4096	64	32	16	2048	32	16	16
url	512	128	128	64	256	64	64	32	256	32	32	32	256	16	16	16

6.3 Memory Requirements

The design of main memory differs from that of cache memory in several ways. First, main memory management is more affordable since it is implemented by software. Second, the page size is usually much larger than the cache block size due to address translation and disk access considerations. Third, the page fault penalty is much higher than the cache miss penalty because page faults cause context switches and disk accesses. A common goal in main memory design is to achieve the intrinsic page-fault rate of programs. This fault rate refers to an infinite main memory. All intrinsic page faults are due to the loading of accessed pages into the infinite main memory. The intrinsic page-fault rate of a program is a function of the page size.

The main memory sizes required to achieve an intrinsic page fault rate for each benchmark are shown for four page sizes in Figures 6.7 and 6.8. We can find that main memory requirement increases as page size increases. This is because of internal fragmentation. However, the increase may be justified by smaller TLB sizes and simpler physical cache design.

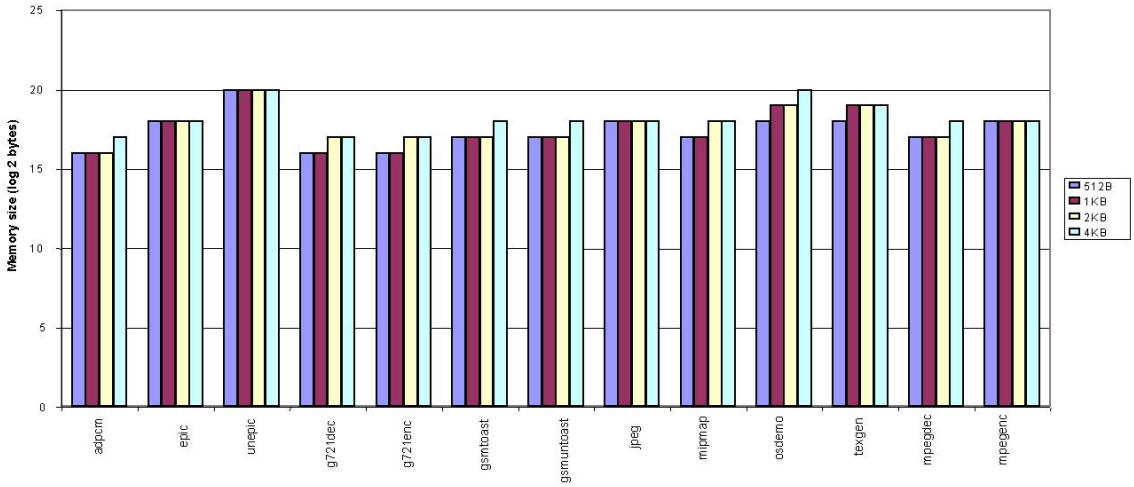

Figure 6.7: Mediabench Memory Requirements

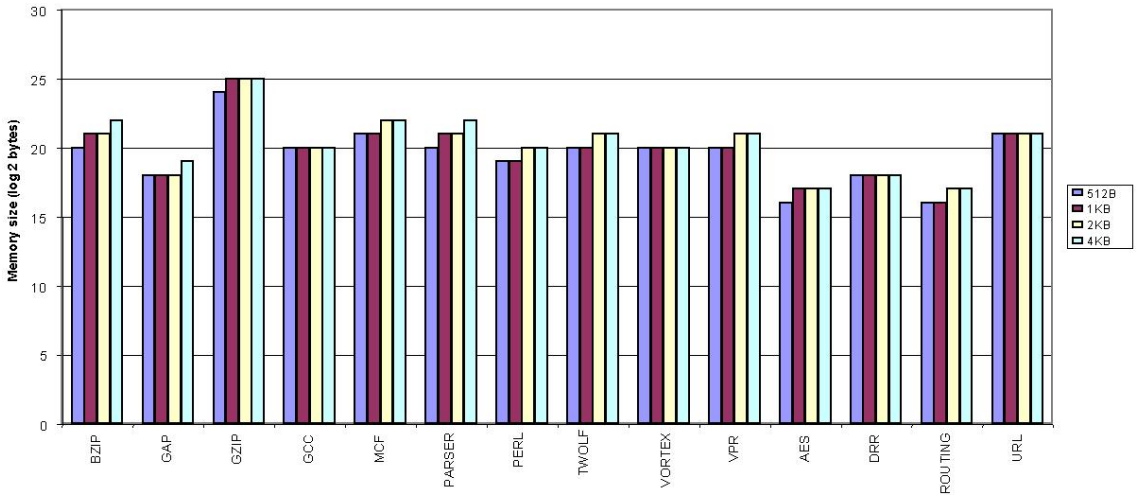

Figure 6.8: SPEC2k and Netbench Memory Requirements

Chapter 7

Similar Benchmarks

This section describes a model using Kiviat structures to determine the similarity between benchmarks based on its characteristics. A Kiviat structure consists of many characteristics of a particular entity combined in a single graph, so that entities can be easily compared as a whole. We have considered Kiviat graphs for each benchmark using the processor and memory characteristics only. Based on these graphs, a visual comparison can be done to determine which benchmarks are similar. Some of the sets of similar benchmarks are shown in Figures 7.1 - 7.6. The main use of this model is that it can also be used to find benchmarks similar to the programs that a user eventually runs. This can help the designer to select benchmarks similar to the workload of the processor, and also help users to select processors designed with benchmarks that are similar to the workloads they shall run.

It is interesting to note that some benchmarks have similar characteristics across application domains, and across benchmark sets (e.g., MPEG2Decode and GAP). But still it can be found that most of the sample sets are dominated by a single suite of the benchmarks rather than a combination of two or more suites. This shows that all three suites are fundamentally different. SPEC is found to have greater memory requirements than both Mediabench and Netbench while Netbench has different branch prediction characteristics from the other two. The instruction level parallelism of both Mediabench and Netbench are also higher than that of SPEC.

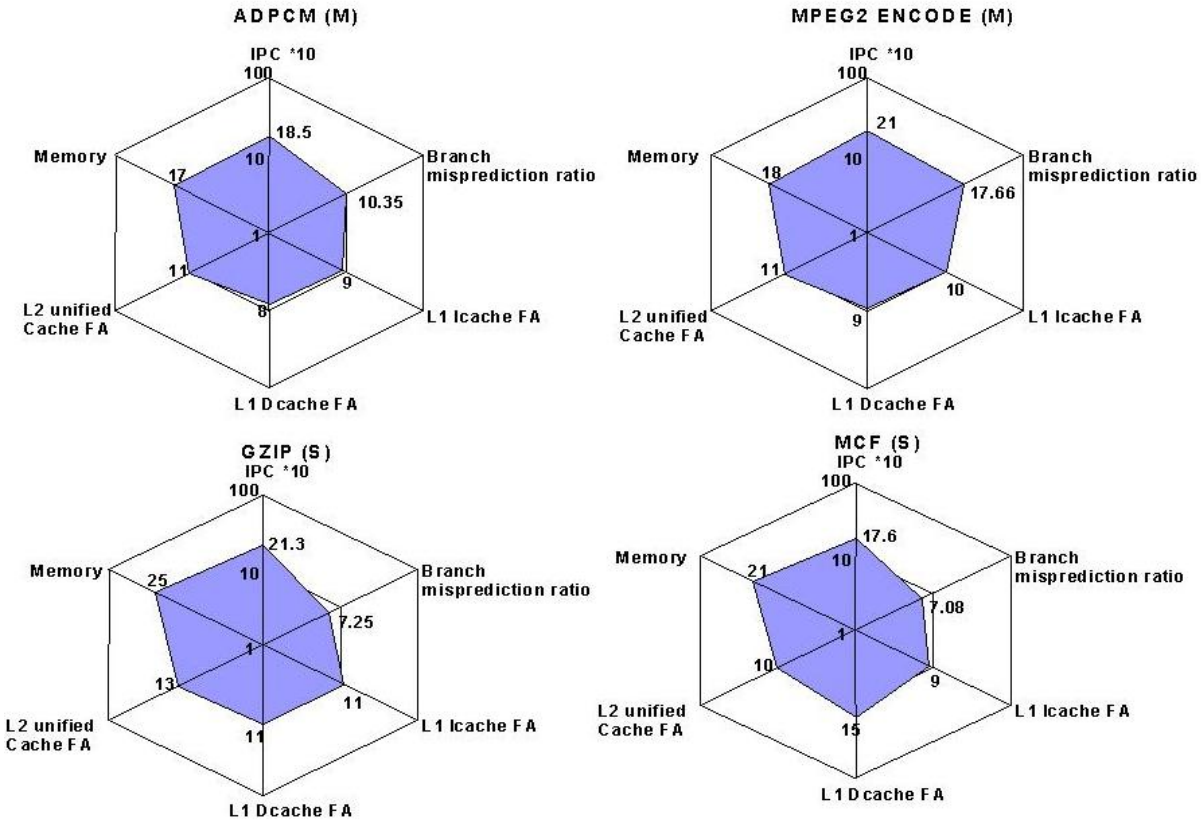

Figure 7.1: Similar Kiviat Graphs - Sample set 1

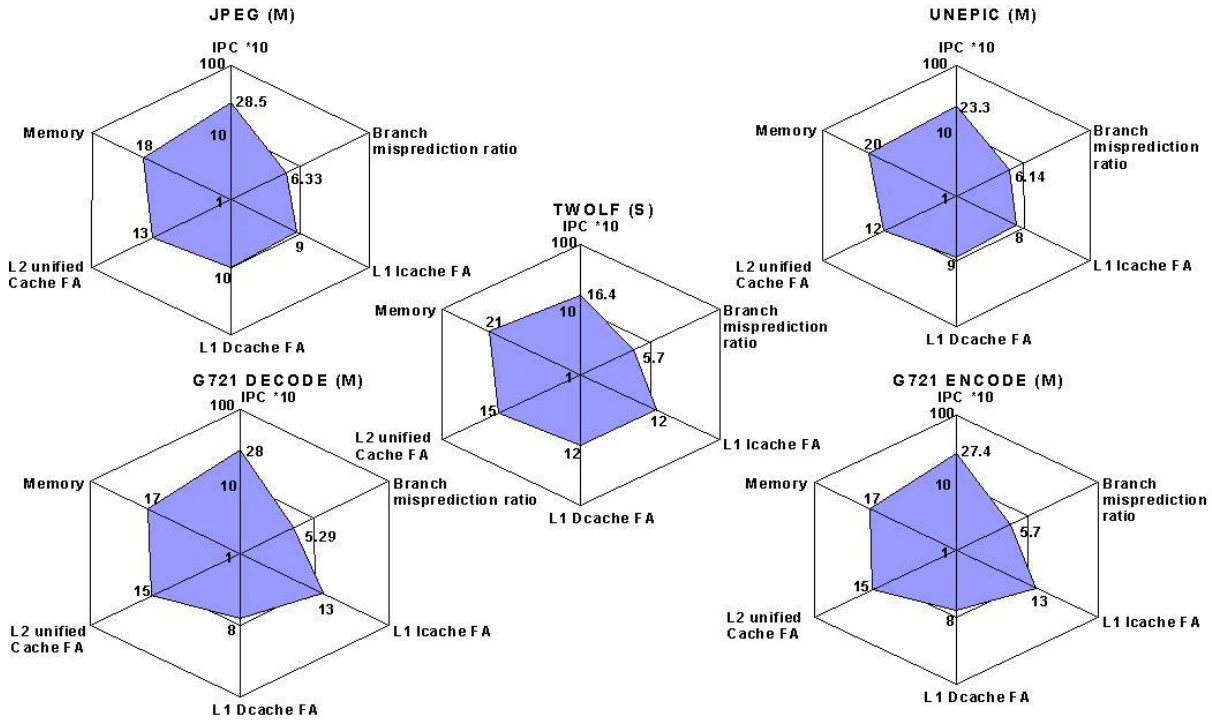

Figure 7.2: Similar Kiviat Graphs - Sample set 2

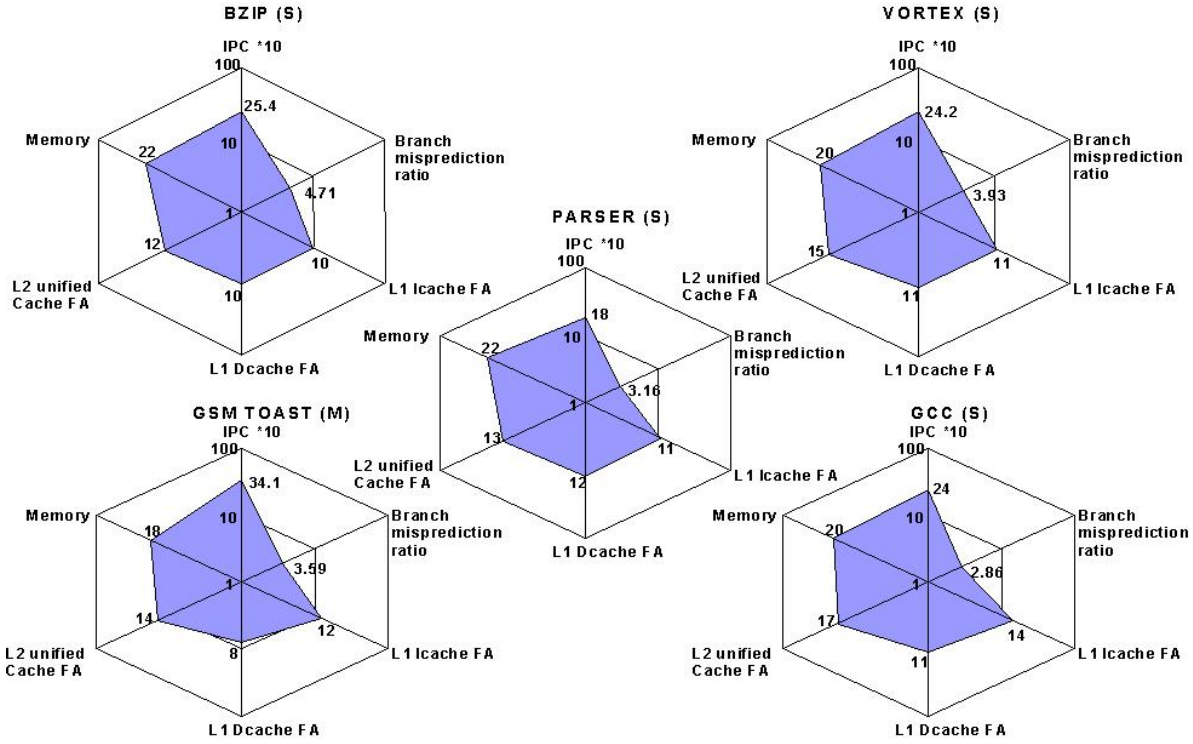

Figure 7.3: Similar Kiviat Graphs - Sample set 3

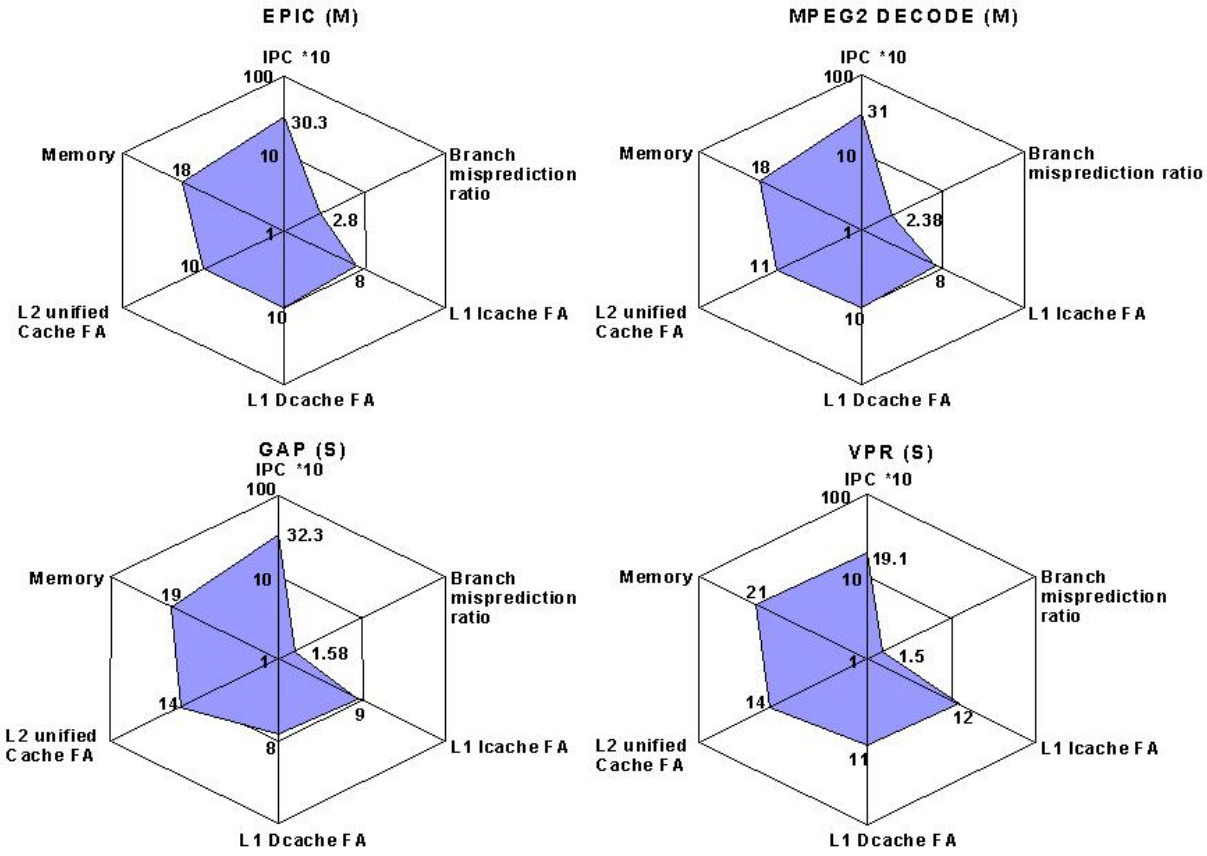

Figure 7.4: Similar Kiviat Graphs - Sample set 4

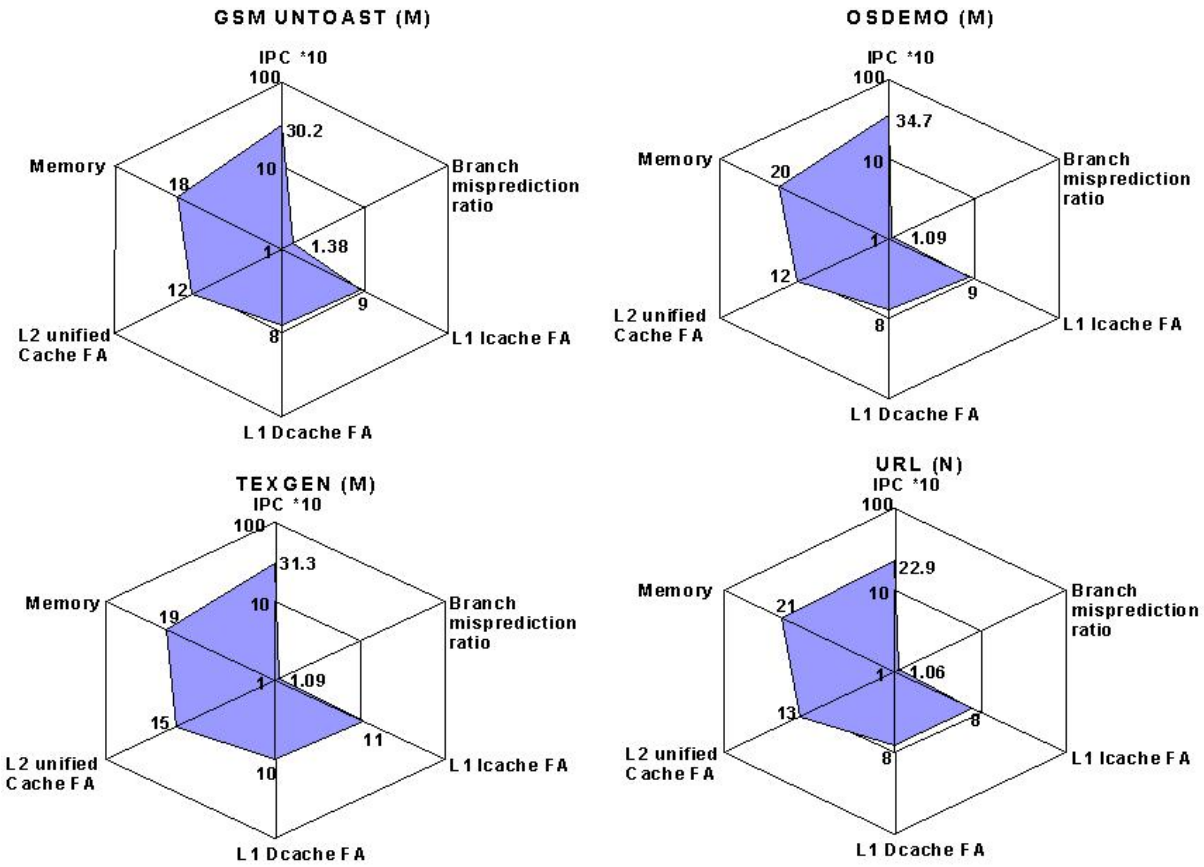

Figure 7.5: Similar Kiviat Graphs - Sample set 5

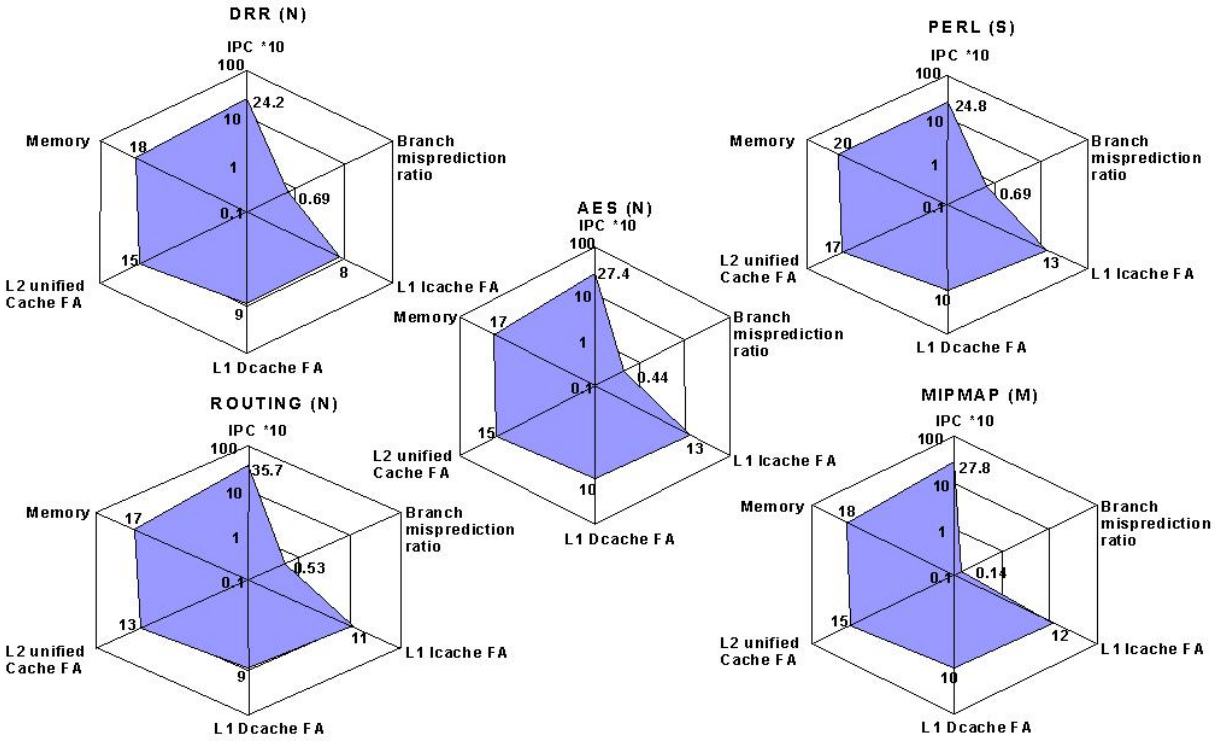

Figure 7.6: Similar Kiviat Graphs - Sample set 6

Chapter 8

Conclusion

Benchmark characterization is a very important step in processor design and is involved in reducing the number of iterations required to attain the best possible initial design. Characterizing benchmarks for embedded processors involves generating power and system-constrained characteristics in addition to the normal architectural characteristics that are generated for a normal processor. An abstract system model that has been designed with simulators can be used to characterize benchmarks. This characterization provides us with processor, memory, power and compression requirements that can indicate at a good possible initial design for a workload similar to that benchmark. Using these characteristics it can also be found that the three benchmark suites (SPEC, Mediabench and Netbench) differ widely in their characteristics. It is observed that SPEC has greater requirements compared to the embedded benchmark suites Mediabench and Netbench. So, a design based on a SPEC benchmark will be overdone for an embedded processor. Therefore, we can conclude that SPEC is not appropriate for embedded processor design.

Bibliography

[1] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling, February 2002. Volume 35, Issue 2, Page(s):59–67.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*, number 10-14 in Computer Architecture, pages 83–94, 2000.

[3] T.M. Conte. Systematic computer architecture prototyping, Septemeber 1992. PhD thesis, Department of Computer Science, University of Illinois, Urbana IL.

[4] T.M. Conte, M.A. Hirsch, and W.W. Hwu. Combining trace sampling with single pass methods for efficient cache simulation, 1999. IEEE Transactions on Computers.

[5] T.M. Conte and W.W. Hwu. Benchmark characterization. In *Proceedings of the Twenty-Fourth Annual Hawaii International Conference*, number 8-11 in System Sciences, pages 365–372, 1991.

[6] T.M. Conte and S.Y. Larin. Compiler-driven cached code compression schemes for embedded ilp processors. In *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, 2000.

[7] J.L. Henning. Spec cpu2000: measuring cpu performance in the new millennium, July 2000. Volume 33, Issue 7,Page(s):28–35.

[8] D.A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the IRE*, pages 1098–1101, 1952.

[9] Chunho Lee, M. Potkonjak, and W.H. Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings. Thirtieth Annual IEEE/ACM International Symposium of Microarchitecture*, pages 330–335, 1997.

[10] G. Memik, W.H. Mangione-Smith, and W. Hu. Netbench: a benchmarking suite for network processors, November 2001. IEEE/ACM International Conference on Computer Aided Design, 2001. ICCAD 2001.