Benchmarking and Analysis of Architectures for CAD Applications

Amit Mehrotra   Shaz Qadeer   Rajeev K. Ranjan   Randy H. Katz

email :{mehrotra,shaz,rajeev,randy}@eecs.berkeley.edu

Department of Electrical Engg. and Computer Science

University of California at Berkeley

Berkeley, CA 94720

Abstract

The SPEC benchmark system has traditionally been used for evaluating computer architectures. However, this system is too general and does not accurately reflect the performance of architectures on domain-specific applications. Moreover, the CPU95 benchmark suite used in the SPEC system is compute-intensive, while many important domains of applications have memory intensive algorithms. In this work, we present a benchmarking methodology for such an application domain – CAD for VLSI design. We have created a benchmark suite consisting of CAD applications from each stage in a typical VLSI design flow. To exercise the memory organization, each application is run on a sequence of input designs of increasing size. We observed that increasing the input size causes non-monotonic variations in the performance of different machines. We simulate the caches of the benchmarked architectures to assess the effect of memory organization on performance.

1 Introduction

Benchmarking is a way of comparing the performance of computer systems by running a set of applications on them and comparing their performance with a reference architecture. However the presence of extremely diverse application domains of computers poses a major problem in the interpretation of benchmark numbers. SPEC (Standard Performance Evaluation Corporation) [1] attempts to come up with a range of "typical" applications for benchmarking purposes. Since its first release in 1989, SPEC benchmarks have come a long way by incorporating more applications, increasing the input size, and making the benchmarking methodology more rigorous.

There are three main problems with SPEC benchmarks as a measure of performance. First, they include a set of general purpose programs motivated by typical software engineering and design environments. We believe that extrapolations from the performance on these programs to specific application domains would be dubious. SPEC reduces the performance of a system on a set of different programs to just two numbers–SPECint and SPECfp. We believe that this methodology is inadequate and that benchmarks consisting of a set of applications that represents a typical workload should exist in each domain. These benchmarks will not necessarily be valid for other domains. Secondly, SPEC benchmarks are supposed to measure CPU performance and hence they are compute-intensive. There are application domains, CAD tools for VLSI design being a prime example, where a lot of applications are memory-intensive and the memory organization of a machine is an important factor in determining the tool response time. Hence, a meaningful benchmark suite should contain applications that exercise the memory hierarchy. The third problem with SPEC numbers is that only one input is used for each application in the benchmark suite. However, the relative performance of computers is not monotonic in the size of the input. To come up with meaningful metrics for the performance of computer systems, it is necessary to take the variation of relative performance with increasing problem size into account.

With the proliferation of CAD tools to support synthesis and verification of electronic circuits, performance evaluation of computers for CAD applications is becoming important. In this work, we take a set of CAD applications spanning the whole ASIC design flow from high level specification to physical design, and benchmark five architectures against them. Each application is fed by a sequence of input designs of increasing size. A direct consequence of using different input sizes for the applications is that memory organization and its performance will play an important role in determining the performance of the machines. Small problems which fit in the cache will be relatively independent of the on-board memory performance. However, for large problems, the performance of the memory organization will strongly affect the overall architecture performance. We try to identify features of the memory organization of the machines that lead to differing performance characteristics for different applications and input sizes.

Other benchmarks have been designed for specific application areas, e.g., SFS benchmark to measure performance of file servers, DebitCredit for benchmarking transaction processing system [2], self–scaling benchmark for I/O performance analysis [3], and PAS [4] and PERFECT [5] for scientific computing, to name a few. However, to the best of our knowledge there has been no work in benchmarks in the domain of CAD applications. In this work, we propose a static benchmark[1] consisting of a set of CAD applications with single user response time as our performance measure. We believe that single user response time is more appropriate because human designers use these applications and there is a direct correlation between tool response time and productivity.

The rest of the paper is organized as follows. In Section 2, we describe our methodology including the selection of applications, architectures and input designs. We present and analyze our experimental results in Section 3.

2 Methodology

Traditionally, the following four questions have been asked for evaluating machines in a particular domain.

1. What are the applications that form the benchmark suite?
2. What machines will be measured?
3. What is the criterion for performance evaluation?
4. How will the data be collected?

We add another question to the above list.

[1] As against scalable benchmarks which scale themselves according to the system.

Application	Memory	Floating point	Integer
VL2MV			$\surd$
SIS			$\surd$
VIS-1	$\surd$		
VIS-2			$\surd$
IRSIM		$\surd$	$\surd$
SPICE	$\surd$	$\surd$	
TimberWolfMC		$\surd$	$\surd$

Table 1: Characteristics of the computation in the applications

VIS-1: VIS with random memory access

VIS-2: VIS with local memory access

5. What input sizes will be used to exercise the applications in the benchmark suite?

2.1 Selection of Applications

Current research in design automation of VLSI systems visualizes system design as a multi–stage process. First the system is described at a behavioral level. In the successive design steps, the description is refined to impose more structure into the design with the filling in of the implementation details. At each step of the refinement process, the behavior of the refined design is verified against the preceding one.

A typical top down design methodology would start with the design description in a hardware description language. This description is compiled into a gate level implementation by a hardware compiler. The gate level design is optimized using combinational and sequential optimization tools. The functionality of the implementation is checked by some verification or simulation tools. The gate level design is mapped into transistors using technology mapping. The transistor level design is further simulated for critical path analysis. And finally transistors are mapped into rectilinear blocks using place and route tools.

Keeping in mind the state of the art in electronic design methodology, we chose the following six tools.

1. **Hardware Compiler: VL2MV** [6] compiles a Verilog description into a gate level netlist of finite state machines (FSMs) which preserves the system behavior defined in terms of simulated results. After creating the parse tree representing the control and data flow, semantic checking is performed and the hardware gates are generated. The underlying computation is largely symbolic with local memory accesses.

2. **Formal Verification Tool: VIS** [7] is a unified environment for formal verification, synthesis, and simulation of finite state systems. We chose the problem of combinational verification by building the BDD [8] representation of two circuits and comparing them. The major underlying computation is memory intensive. We have benchmarked machines for equivalence checking with two kinds of underlying BDD computations – random memory access [9] and localized memory access [10].

3. **Logic Synthesis Tool: SIS** [11] is a logic synthesis system for combinational and sequential circuits. It takes as input a representation of a set of Boolean functions in the form of a finite state machine or a netlist of gates and latches and provides a set of commands that can be used to optimize the circuit for either area, performance or power. We use a script that tries to optimize for area. The underlying computation was mainly symbolic with largely local memory accesses.

4. **Switch Level Circuit Simulator: IRSIM** [12] models the transistor as a bi-directional switch with a finite ON resistance and capacitances and simulates the resulting RC network using an event driven simulator. The main computation is dynamic partitioning of circuits involving integer operations, and computation of delay times and node voltages using floating point operations. Since all the computation is performed in isolated stages of the circuit, the memory accesses are mostly local in nature.

5. **Transistor Level Circuit Simulator: SPICE** [13] is a transistor level simulator. The input is a netlist that describes the various circuit elements and their interconnections, a set of models for the circuit elements and a specification of the type of analysis to be performed.

A lot of floating point computations are needed involving equation formulation, solution of non-linear equations, and numerical integration.

6. **Layout Generator: TimberWolfMC** [14] uses simulated annealing to solve the floor–planning and placement problem for macro-cells . The underlying algorithm proceeds in two stages. In the first stage, simulated annealing is used to place the cells approximately. This involves a lot of floating point computations. In the second stage, channel definition, global routing and channel routing are done and the placement obtained from the first stage is refined if it does not meet the constraint. This stage mostly involves symbolic computation.

2.2 Input Selection

In our benchmarking effort, we do not attempt to measure the performance of whole system. We limit ourselves to measure the performance of CPU and the cache organization. Further we feel that it is a limitation of the SPEC benchmarks that they use just one input for each program. We think that this does not represent the full spectrum of work loads experienced by a particular CAD application in a typical design environment. In our work, for each program in the benchmark suite, we chose six input examples of increasing size. All the six inputs for any program fit in the main memory sub-system of memory hierarchy.

2.3 Selection of Machines

The considerations that went into choosing the machines to be benchmarked were availability, variety and portability of the CAD tools we are using. We selected five architectures whose specifications are given in Table 2[2]. All except Intel 80586 are RISC architectures. We decided to make Sparc 10 our base machine. A very practical concern was portability of the tools we were benchmarking across the different architectures.

Note that the benchmarking experiments were performed almost a year ago and machines used do not represent the state-of-the-art architectures of current days. However, our benchmarking methodology remains applicable to any set of architectures.

[2]Future references to machines in this report are of the form “architecture_clockrate”.

Architecture		Sparc10	MIPS R4400	Sparc20	Alpha 21064	Intel 80586
L1 Cache	Size (K)	16	16	8	8	8
	Block Size	32	16	32	32	32
	Associativity	4-way	direct	direct	direct	2-way
L2 Cache	Size (M)	0.25	1	0.25	0.5	0.25
	Block Size	128	64	32	32	32
	Associativity	16-way	8-way	direct	2-way	2-way
Main Memory		64M	440M	80M	128M	32M
TLB Size		32	32	64	16	N/A
Page Size		4	4	4	8	N/A
Clock Speed (MHz)		50	60	100	182	120

Table 2: Machine specifications

2.4 Performance Criterion

The CAD programs in our benchmark suite are a very important aid to system designers today. Ideally, the designers would like to use these programs interactively. Therefore, response time seems to be the natural performance criterion. But, we chose to use the user time (system + CPU) as our performance measure because the machines that we were benchmarking were all compute servers and the response time is heavily dependent on the load. Since the loading of these machines was not under our control, we chose the next best performance criterion i.e. user time. It seems reasonable to assume that these times will be very close to the response times for personal machines. We also limited the size of the examples we ran our programs on to those which could fit into the main memory of all machines to make the user time even closer to the response time.

2.5 Data Collection

The confidence level of conclusions drawn from a set of measured data depends on the size of the data set. The larger the sample, the higher is the confidence level. To estimate the mean performance of a system with an accuracy of $\pm r\%$ and a confidence level of $100(1-\alpha)\%$, the needed number of observations n is given by [15] as

$$n = \left(\frac{100 z_\alpha s}{r\overline{x}}\right)^2$$

where

z_α = normal variate of the desired confidence level

$\overline{x}$ = observed mean

s = observed standard deviation

We take a few observations and find the required number of iterations from the calculated mean and standard deviation. All our results are reported with an accuracy of $\pm 2\%$ at a confidence level of 95%.

To make the experiments fair the following guidelines were used.

1. To reduce the setup overhead, example sizes were chosen such that the run time was at least of the order of seconds.

2. For applications which involved a lot of terminal I/O operations we found a difference in running time if the output was directed to a file instead of the terminal. Hence we chose to direct all outputs to a file.

3 Technical Results and Analysis

In Section 3.1, we discuss the performance of various machines on the applications in our benchmark suite. Sparc10 with the clock speed of 50MHz is the base machine for computing performance ratios. We simulated the cache configurations of the machines to learn more about the effect of their memory organization on their performance. In Section 3.2, we discuss the cache simulation results. Cache parameters of various architectures are listed in Table 2.

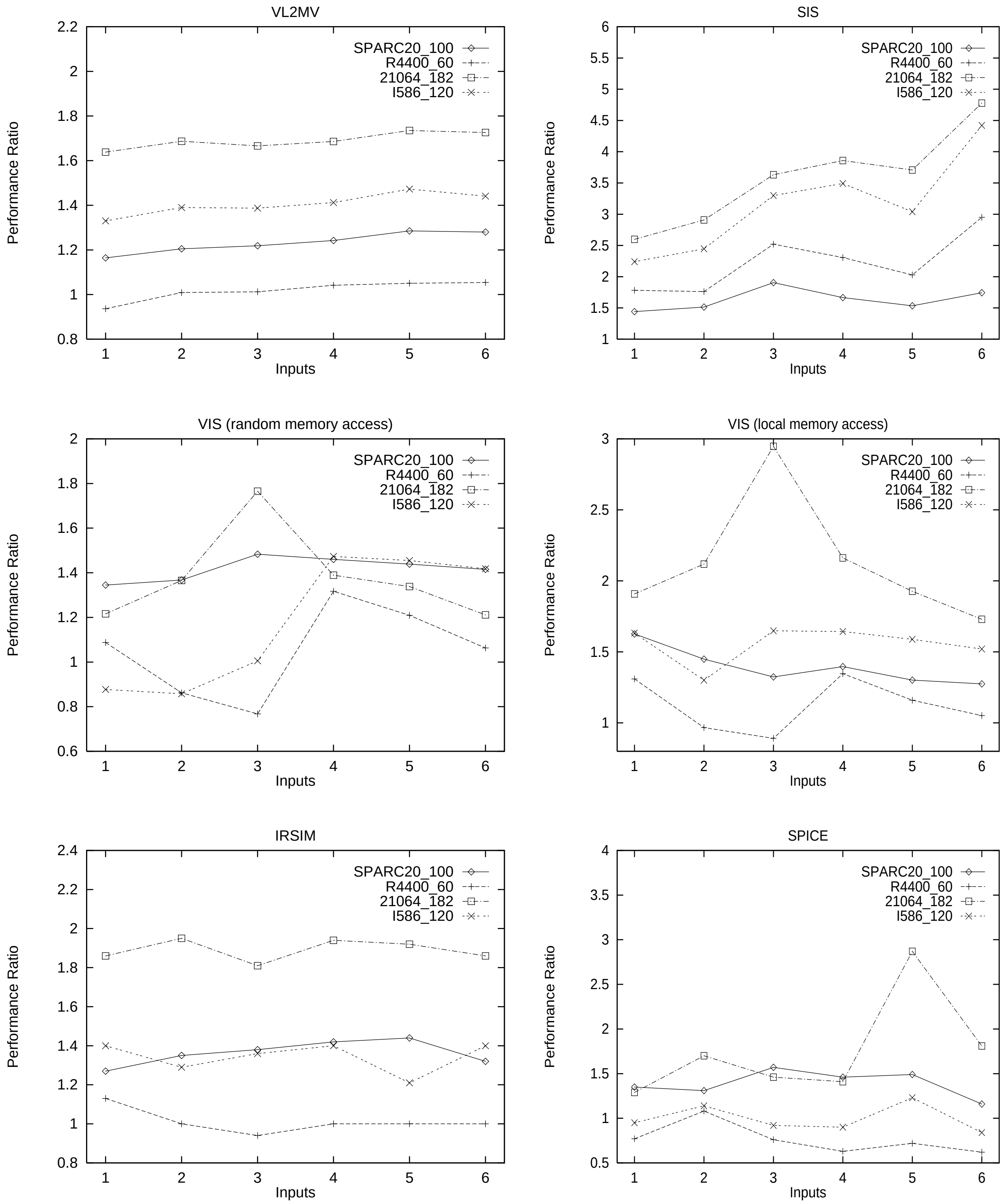

Figure 1: Comparison of performance ratios for various applications

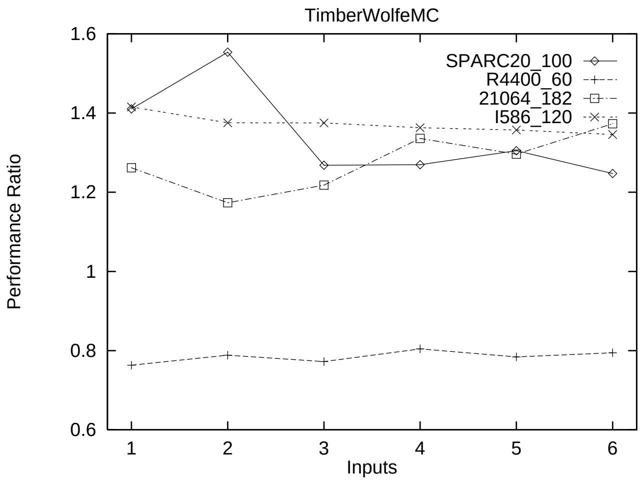

Figure 2: Comparison of performance ratios for various applications

3.1 Analysis of Results

3.1.1 VL2MV

For this application, we chose an industrial design from Motorola. This circuit acts as the controller between a CPU and a serial bus. Input designs of different sizes were created by replicating the number of controllers in the design. The performance ratios are shown in Figure 1. We notice that 1) relative performance of various machines remain unchanged with increase in the input size, and 2) performance of the machines is proportional to the clock rate. These observations substantiate our hypothesis that the computations in the hardware compiler are very local and the performance is dominated by the clock speed of the machine with little or no effect from memory organization.

3.1.2 SIS

The inputs were taken from ISCAS benchmark set. The optimization script performs only symbolic computations on the nodes of the circuits. Moreover, all the examples could fit in the memory. Our intuition would lead us to expect that memory effects would be secondary in this case and the performance will be a reflection of the MIPS rating of the machine. However, as shown in Figure 1 we notice an anomaly which is that R4400_60 is performing much better than its low clock speed would indicate. In Section 3.2.1, we make an attempt to explain this anomaly basing our arguments on cache simulation results. We notice that its running times are quite faster than the Sparc20_100. We also observe that both the Sparc machines perform poorly on this application. Since Sparc10_50 is the base machine, the performance ratios of the non-Sparc machines is much higher compared to those in other applications.

3.1.3 VIS

This application is highly memory intensive. As mentioned earlier, the underlying data structure (BDDs) can be manipulated with either random or local memory access behavior. We make use of two BDD packages – CMU [9] and CAL [10] which have random and local memory access patterns respectively, to get two sets of results. The inputs were taken from MCNC and ISCAS benchmark sets. The performance ratios are shown in Figure 1. In the case of random memory access behavior, we would expect that the memory organization of an architecture would have a bigger role than the clock rate in determining the performance. From the graph we observe that Sparc20_100 and i586_120 both outperform 21064_182 for larger inputs. The reason for this behavior is the lower second level access time for Sparc20_100. As expected, localized memory access makes CPU speed a dominating factor in the performance. As seen in Figure 1, 21064_182 far outperforms all other architectures. R4400_60 with minimum clock rate performs the worst on almost all the examples.

3.1.4 IRSIM

Figure 1 shows the relative performance of various machines for this application. We observe that increasing the circuit size does not have too much effect on the performance ratio. This is because the circuit is dynamically partitioned into stages and computation is performed on each stage separately. These stages are typically small and fit the first level cache of the machines. The instructions performed are mostly floating point. Hence, L-1 cache size and access time and the floating point unit are the major factors in determining the performance. 21064_182, which has the smallest L-1 cache access time, was found to be consistently the fastest machine. The i586_120 and the Sparc20_100 were found to have comparable performance due to their similar L-1 access time. R4400_60 and Sparc10_50 were found to be the slowest.

3.1.5 SPICE

SPICE sets up a matrix equation representing the Newton-Raphson iterations at each time point and solves it. The critical parameters of performance are the floating point unit and the cache organization. The cache organization is critical because the temporal access pattern for matrix entries is non-local resulting in each entry being read into and knocked out of the cache several times for a large matrix. Figure 1 shows the relative performance for various architectures. 21064_182 generally gave the best performance. Sparc20_100 performs better than i586_120 probably due to a faster L-2 cache. R4400_60 ended up being the slowest of the lot, consistently performing worse than Sparc10_50, which has smaller clock rate and slower L-2 cache. These two observations seem to indicate a better floating point unit for the Sparc machines.

3.1.6 TimberWolfMC

Figure 2 shows the relative performance for various architectures. In the first stage, it performs simulated annealing which has floating point computation in every iteration. In the second stage, it does an iterative place and route in which there is symbolic computation, but experiments with TimberWolfMC has shown that very few iterations are needed in practice. So the computation is dominated by floating point operations. i586_120, 21064_182 and Sparc20_100 have running times in the same range with i586_120 having a slight edge. Sparc10_50 performs better than R4400_60. This again leads us to conjecture that the Sparc machines have a comparatively better floating point unit.

3.1.7 Combined analysis of results

Increasing the input size does seem to have an effect on the relative performance of the different machines. This can be observed for most of the applications. Only VL2MV which is computation intensive and doesn't use much memory has uniform relative performance.

A lot of exceptions to the "rule" (or rather misconception) that faster is better were found. For the application SIS, it was found that the MIPS R4400 machine (60MHz) was performing better than the Sparc20 (100MHz). For VIS with non-local memory accesses, Sparc20 performed very well because of its very fast 256K second level cache compared to the Alpha 21064 machine with higher

Application	% reads	% writes
SIS	17	2
VIS-1	10	2
VIS-2	19	10
IRSIM	18	8

Table 3: Read and write percentages for various applications

VIS-1: VIS with random memory access

VIS-2: VIS with local memory access

clock speeds. But with local memory accesses, the advantage of fast second level cache decreased and the greater clock speed of the Alpha 21064 (182MHz) made it faster. In general, we found that for CAD applications with inputs that fit in the main memory, performance is a complex function of clock speed and cache organization.

3.2 Cache Simulation Results and Analysis

To gain further insight into the performance of machines for CAD applications, we made use of a commercial cache simulator Shade [16]. We simulated the cache configuration of all the machines on a few applications to determine the contribution of the memory system towards the performance.

3.2.1 Simulating cache configurations for the different machines

We selected three applications from our benchmark suite – SIS, VIS and IRSIM, and simulated the caches of the different machines on them. We wanted to observe the correlation between performance ratio and the L1 and L2 miss rates for the selected applications. A single input example was chosen for each application toward this goal. Figure 3 shows the performance ratio and miss rates in the form of a bar graph. Table 3 shows the memory references (reads and writes) as a fraction of the total number of instructions.

A number of general observations could be made from the output of cache simulation. The number of instruction misses in both L-1 and L-2 caches were always very small. Therefore, we feel

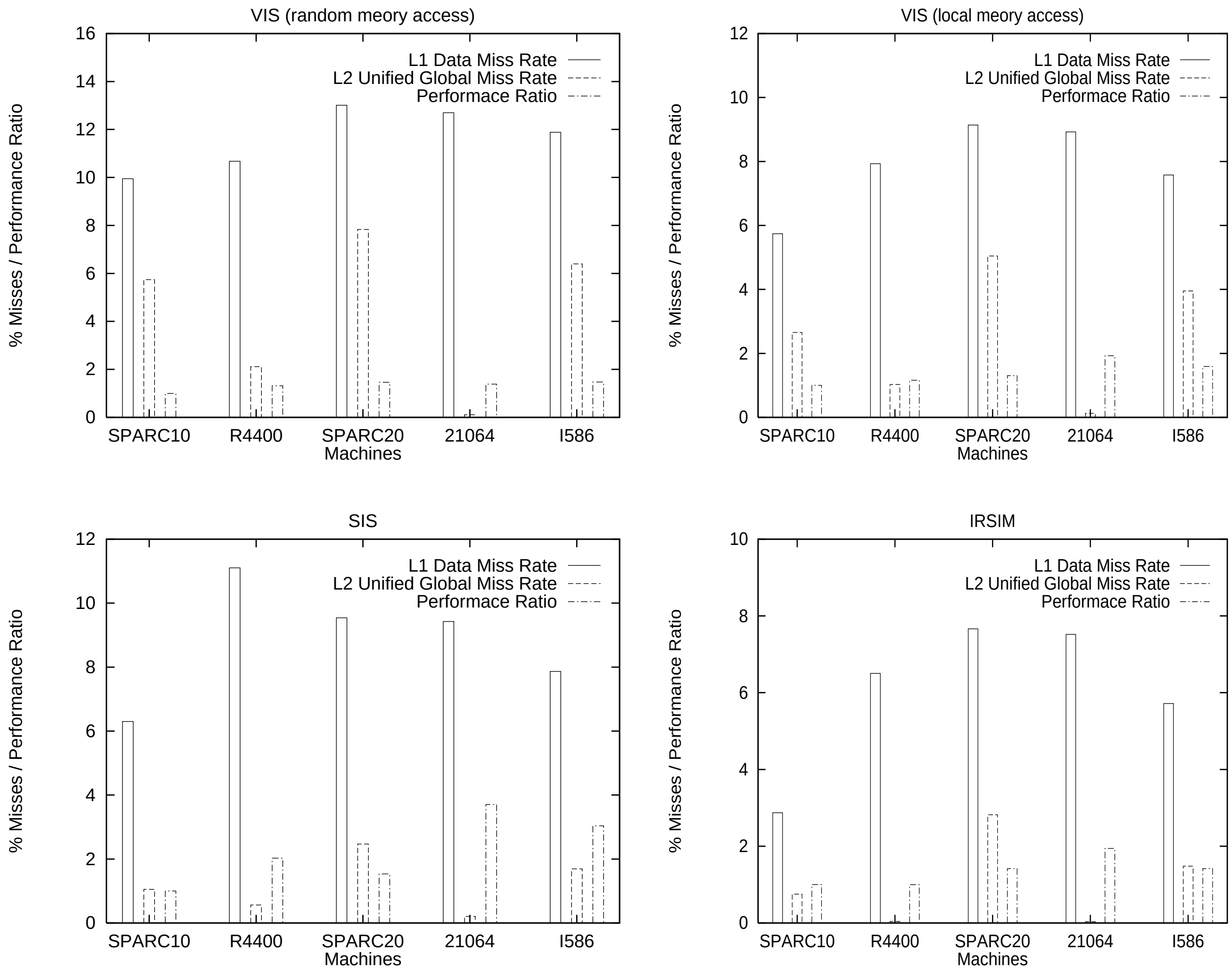

Figure 3: Comparison of L1 and L2 miss rates and performance ratios for various applications

that instruction misses do not affect the performance of the machines. Writes dominate reads for data references to the L-2 cache since the L-1 cache in all machines is write through. On the other hand, reads dominate writes for the L-1 cache. We found that the size of the L-2 cache of 21064_182 (largest at 4M) was sufficient for both data and instructions for all the applications except SIS. But, the sizes of the other second level caches was not sufficient for any application. We now analyze the cache simulation results for the three applications.

SIS The global miss rate of L2 cache of Sparc20_100 is about six times that of R4400_60. This is because of the large L2 cache size for R4400_60. This explains the better performance of R4400_60 over Sparc20_100 even though the latter has a larger clock rate. 21064_182 performs the best because of the largest second level cache and the smallest access time for both the first and the second level cache.

VIS The two versions of this application (random and local memory access patterns) exhibit some interesting features of the effect of memory organization on the overall performance. We focus on the performance of Sparc_20, 21064_182 and R4400_60.

For the case of random memory accesses, we observe that Sparc20_100 has the best performance overall even though its second level miss rate is higher than that of 21064_182. This is because in 21064_182, even if a hit occurs at the second level, the small number (16) of TLB entries causes TLB misses, increasing the second level hit time. Sparc20_100, on the other hand, has a 64 entry TLB with 4K page size. So, a hit in the second level cache (256K) will always result in a TLB hit. Therefore, the access time for the second level cache is smaller and in spite of a higher miss rate, this system performs better 21064_182 running at almost twice the speed.

For the case of local memory accesses, the cache misses are not significant. Note that the increase in the fraction of read/write instructions (Table 3) is due to the nature of BDD algorithm. Also, local memory accesses reduce the number of TLB misses significantly, making 21064_182 perform the best, because of much higher clock speed.

IRSIM This application mostly involves floating point calculations and hence the performance does not depend on memory hierarchy. Therefore, despite higher second level cache miss rates

Application	MIPS R4400	Sparc20	Intel 80586	Alpha 21064
VL2MV	1.012	1.219	1.387	1.667
SIS	2.521	1.905	3.230	3.631
VIS-1	0.768	1.483	1.005	1.765
VIS-2	0.891	1.324	1.649	2.947
IRSIM	0.940	1.380	1.360	1.810
SPICE	0.760	1.570	0.920	1.460
TimberWolfMC	0.772	1.268	1.375	1.218
G.M.	0.995	1.435	1.443	1.935

Table 4: Performance ratios for a single input

VIS-1: VIS with random memory access

VIS-2: VIS with local memory access

G.M. : Geometric mean

Sparc20_100 and i586_120 outperform R4400_60 because of their presumably better floating point units.

3.2.2 Discussion

Cache analysis proved helpful in understanding the overall behavior of the system for the given application. We found that in general for memory intensive programs, miss rate is usually a good performance criterion. However it was pointed out in one case that we cannot ignore the miss penalty altogether. On more than one occasion we found that a slower machine actually performs better because of better memory organization. We found that the percentage of read/write instructions varied from 12% to 29% for the applications we investigated. Hence, application type varies from compute intensive to memory intensive.

To illustrate the fact that a single "performance number" cannot be accurate across all applications, we have given performance ratios for all the architectures in Table 4. These ratios have been taken for the third input design for each application. We also give the geometric mean of ratios across all the applications. We observe that the order of performance given by the geometric mean is not consistent with four out of seven applications. Therefore, even for a particular domain it is not appropriate to provide a single performance number.

4 Conclusions and Future Work

We strongly believe that reporting performance as a single number is inadequate for benchmarking machines. Even in this very restricted field of CAD applications we found extreme variation of performance of machines depending on the application as well as the input size. We feel this vindicates our viewpoint that computer system performance benchmarks should be domain specific. Moreover, they should report results on a range of inputs of varying sizes. We hope this study can provide some guidelines for computer system designers who are vying for the lucrative CAD market, and to the users of such systems for evaluating the various options available to them. Clearly no one system will be suitable for all the applications and an effort to select a system on the basis of the expected workload should be made. We also found that performance of systems running CAD applications critically depends on memory and for some applications it depends solely on the memory performance. Hence a lot of design effort must be put in the memory organization.

We have presented a benchmark suite for CAD applications, which we believe is representative of the tools used in the industry at different levels of the design flow ranging from hardware compilation to physical design. We also presented performance results for five machines on inputs of varying sizes and highlighted the impact of input size on the performance.

The access to source code for uniform compilation restricted our application choice to public-domain programs. It would be interesting to benchmark these machines on industrial CAD tools. We would also like to use some industrial inputs for the applications. It would be instructive to repeat our experiments by using native compilers for all the architectures. We could not do this for reasons of availability and porting problems. We limited ourselves to the first and second level cache as far as assessing the performance of the memory organization was concerned. A study of the disk I/O for CAD applications would be interesting.

References

[1] *The Standard Perfomance Evaluation Corporation*, 1996. `http://www.specbench.org`.

[2] Anonymous et al., "A Measure of Transaction Processing Power," *Datamation*, Apr. 1985.

[3] P. M. Chen and D. A. Patterson, "A New Approach to I/O Performance Evaluation - Self-scaling I/O Benchmarks, Predicted I/O Performance," *ACM Transactions on Computer Systems, 12:4*, Nov. 1994.

[4] D. Bailey, J. Barton, T. Lasinski, and H. Simon, "The NAS Parallel Benchmarks," Tech. Rep. RNR-91-002, NASA Ames Research Center, Aug. 1991.

[5] M. Barry et al., "The PERFECT Club: Effective Performance Evaluation of Supercomputers," in *Intl. J. Supercomputing Appl.*, Dec. 1989.

[6] S.-T. Cheng, G. York, and R. K. Brayton, "VL2MV: A Compiler from Verilog to BLIF-MV," Oct. 1993. Supplied documentation.

[7] R. K. Brayton *et al.*, "VIS: A System for Verification and Synthesis," in *Proc. of the 8th International Conference on Computer Aided Verification*, vol. 1102 of *Lecture Notes in Computer Science*, pp. 428–432, Springer-Verlag, 1996.

[8] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 677–691, Aug. 1986.

[9] D. E. Long, "BDD Manipulation Library." Public software. Carnegie Mellon University, Pittsburgh, PA, June 1993. `ftp://emc.cs.cmu.edu/pub/bdd/bddlib.tar.Z`.

[10] J. V. Sanghavi, R. K. Ranjan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "High Performance BDD Package Based on Exploiting Memory Hierarchy," in *Proc. of the Design Automation Conf.*, (Las Vegas, NV), June 1996.

[11] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Cir-

cuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[12] A. Salz and M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," in *Proc. of the Design Automation Conf.*, pp. 173–178, 1989.

[13] L. Nagel, "A Computer Program to Simulate Semiconductor Circuits," Tech. Rep. UCB/ERL M75/520, Electronics Research Lab, University of California, Berkeley, CA94720, May 1975.

[14] C. Sechen, "Chip-planning, Placement, and Global Routing of Macro-cell Integrated Circuits using Simulated Annealing," *International Journal of Computer Aided VLSI Design, vol. 2*, pp. 1–10, 1990.

[15] R. Jain, *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, 1991.

[16] *SHADE Cache Simulator*, 1996.