Generation, Validation and Analysis of SPEC CPU2006 Simulation Points Based on Branch, Memory and TLB Characteristics

Karthik Ganesan, Deepak Panwar, and Lizy K John

University of Texas at Austin,
1 University Station C0803, Austin, TX 78712, USA

Abstract. The SPEC CPU2006 suite, released in Aug 2006 is the current industry-standard, CPU-intensive benchmark suite, created from a collection of popular modern workloads. But, these workloads take machine weeks to months of time when fed to cycle accurate simulators and have widely varying behavior even over large scales of time [1]. It is to be noted that we do not see simulation based papers using SPEC CPU2006 even after 1.5 years of its release. A well known technique to solve this problem is the use of simulation points [2]. We have generated the simulation points for SPEC CPU2006 and made it available at [3]. We also report the accuracies of these simulation points based on the CPI, branch misspredictions, cache & TLB miss ratios by comparing with the full runs for a subset of the benchmarks. It is to be noted that the simulation points were only used for cache, branch and CPI studies until now and this is the first attempt towards validating them for TLB studies. They have also been found to be equally representative in depicting the TLB characteristics. Using the generated simulation points, we provide an analysis of the behavior of the workloads in the suite for different branch predictor & cache configurations and report the optimally performing configurations. The simulations for the different TLB configurations revealed that usage of large page sizes significantly reduce the translation misses and aid in improving the overall CPI of the modern workloads.

1 Introduction

Understanding program behaviors through simulations is the foundation for computer architecture research and program optimization. These cycle accurate simulations take machine weeks of time on most modern realistic benchmarks like the SPEC [4] [5] [6] suites incurring a prohibitively large time cost. This problem is further aggravated due to the need to simulate on different micro-architectures to test the efficacy of the proposed enhancement. This necessitates the need to come up with techniques [7] [8] that can facilitate faster simulations of large workloads like SPEC suites. One such well known technique is the Simulation Points. While there are Simulation Points for the SPEC CPU2000 suite widely available and used, the simulation points are not available for the SPEC CPU2006 suite. We used the SimPoint [9] [10] [11] tool to generate these simulation points for the SPEC2006 benchmark suite and provide it for use at [3].

The contributions of this paper are two-fold. The first contribution is the creation of the simulation points, which we make it available at [3] to the rest of the architecture research community. We also provide the accuracy of these simulation points by comparing the results with the full run of select benchmarks. It must be noted that 1.5 years after the release of SPEC CPU2006, simulations based papers using CPU2006 are still not appearing in architecture conferences. The availability of simulation points for CPU2006 will change this situation.

The second contribution is the use of CPU2006 simulation points for branch predictor, cache & TLB studies. Our ultimate goal was to find the optimal branch predictor, the cache

and the TLB configurations which provide the best performance on most of the benchmarks. For this, we analyzed the benchmark results for different set of static and dynamic branch predictors [12] and tried to come up with the ones that perform reasonably well on most of the benchmarks. We then varied the size of one of these branch predictors to come up with the best possible size for a hardware budget. A similar exercise was performed to come up with the optimum instruction and data cache design parameters. We varied both the associativity and size of caches to get an insight into the best performing cache designs for the modern SPEC CPU workloads. The performance for different TLB configurations was also studied to infer the effect of different TLB parameters like the TLB size, page size and associativity.

It should be noted that such a study without simulation points will take several machine weeks. Since the accuracy of the simulation points were verified with several full runs, we are fairly confident of the usefullness of the results.

2 Background

Considerable work has been done in investigating the dynamic behavior of the current day programs. It has been seen that the dynamic behavior varies over time in a way that is not random, rather structured [1] [13] as sequences of a number of short reoccurring behaviors. The SimPoint [2] tool tries to intelligently choose and cluster these representative samples together, so that they represent the entire execution of the program. These small set of samples are called simulation points that, when simulated and weighted appropriately provide an accurate picture of the complete execution of the program with large reduction in the simulation time.

Using the Basic Block Vectors [14] , the SimPoint tool [9][10][11] employs the K-means clustering algorithm to group intervals of execution such that the intervals in one cluster are similar to each other and the intervals in different clusters are different from one another. The Manhattan distance between the Basic Block Vectors serve as the metric to know the extent of similarity between two intervals. The SimPoint tool takes the maximum number of clusters as the input and generates a representative simulation point for each cluster. The representative simulation point is chosen as the one which has the minimum distance from the centroid of the cluster. Each of the simulation points is assigned a weight based on the number of intervals grouped into its corresponding cluster. These weights are normalized such that they sum up to unity.

3 Methodology

In this paper we used sim-fast, sim-outorder simulators of the simplescalar toolset [6] along with the SimPoint tool to generate the simulation points for the SPEC CPU2006 suite. Figure 1 shows the flowchart representation of the methodology. We used sim-fast simulator to identify the different basic blocks in the static code of the benchmark and generate a Basic Block Vector for every fixed dynamic interval of execution of the program. We chose the interval size to be 100 million instructions. Further, these basic block vectors are fed as input to the clustering algorithm of the SimPoint tool, which generates the different

simulation points (collection of Basic Block Vectors) and their corresponding weights. Having obtained the simulation points and their corresponding weights, the simulation points are tested by fast-forwarding (i.e., executing the program without performing any cycle accurate simulation, as described in [3]) up to the simulation point, and then running a cycle accurate simulation for 100 million instructions. The sim-outorder tool provides a convenient method of fast-forwarding, to simulate programs in the manner described above. Fast-forwarding a program implies only a functional simulation and avoids any time consuming detailed cycle accurate measurements. The statistics like CPI (Cycles Per Instruction), cache misses, branch mispredictions etc. are recorded for each simulation point. The metrics for the overall program were computed based on the weight of each simulation point. Each of the individual simulation point is simulated in parallel and their results were aggregated based on their corresponding normalized weight. For example, the CPI was computed by multiplying the CPI of each individual simulation point with its corresponding weights as in eqn (1).

$$CPI = \sum_{i=0}^{n} (CPI_i * weight_i) \tag{1}$$

On the other hand, the ratio based metrics like branch misprediction rate, cache miss ratio were computed by weighing the numerator and denominator correspondingly as in eqn (2).

$$MissRatio = \frac{\sum_{i=0}^{n} (misses_i * weight_i)}{\sum_{i=0}^{n} (lookups_i * weight_i)} \tag{2}$$

The accuracy of the generated simulation points were studied by performing the full program simulation using sim-outorder simulator and comparing the metrics like CPI, cache miss ratios and branch mispredictions. This validation was performed to know the effectiveness of the SimPoint methodology on SPEC CPU2006 [15] suite in depicting the true behavior of the program. Since, sim-outorder runs on SPEC CPU2006 take machine weeks of time, we restricted ourselves to running only a few selected benchmarks for this purpose.

For studying the branch behavior of the suite we once again used the sim-outorder simulator available in SimpleScalar [6]. This tool has in-built implementation for most of the common static and dynamic branch predictors namely Always Taken, Always Not-Taken, Bimodal, Gshare and other Twoway adaptive predictors. We studied the influence of above predictors on the program behavior in terms of common metrics like execution time, CPI, branch misprediction. One of the best performing predictors was chosen and the Pattern History Table (PHT) size was varied and the results were analyzed to come up with an optimal size for the PHT.

To get an insight into the memory and TLB behavior of the Suite, the same sim-outorder simulator was employed, using which the configurations for the different levels of the cache hierarchy and TLB were specified. We obtained the corresponding hit and miss rate for various configurations along with their respective CPIs.

4 Simulation Points Generation and Verification

Figures 2 shows the sim-fast results for the SPECINT and SPECFP benchmarks. The tables in the Figures. 2 and 3 show the number of simulation points generated for each of the bench-

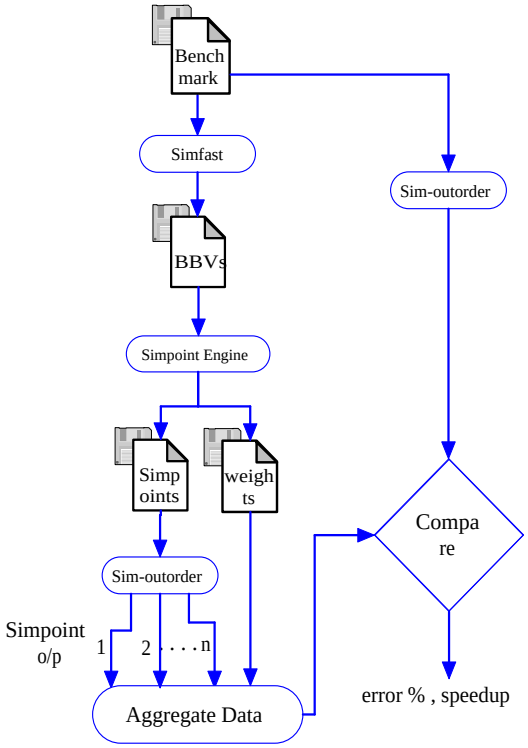

Fig. 1. Simulation point Generation and Verification

marks along with their instruction count and simulation time on a 2 GHz Xeon machine. The interval of execution given to the sim-fast simulator was 100 million instructions. Also, maximum number of clusters given to the SimPoint tool were 30. These simulation points were launched as parallel jobs on the Texas Advance Computing Center (TACC) using the sim-outorder simulator. A node on TACC could have been 2x to 3x faster than the other xeon machine to which the execution times are compared. But, still the speedup numbers here are too high that this discrepancy in machine speeds can be safely ignored. The final aggregated metrics for the simulation point runs were calculated using the formulae mentioned in the previous section. The full run simulations were also carried out for a few integer and floating point benchmarks and the accuracy of the generated simulation points were obtained by comparing the results.

To verify the accuracy of the simulation points, we further compared the CPIs and cache miss ratios of the simulation point run to that of full run and analyzed the speedup obtained due to the usage of simulation points. The configuration that we used to simulate the various full and the simulation point runs is with a RUU size of 128, LSQ size of 64, decode, issue and commit widths of 8, L1 data and instruction cache size of 256 sets, 64B block size, an associativity of 2, L2 data and instruction cache size of 4096 sets, 64B block size, an associativity of 4. The ITLB size used was 32 sets with 4K block size, and an associativity of 4. The DTLB size used was 64 sets, 4K block size and an associativity of 4. The number of Integer ALUs were set to 4 and the number of Floating Point ALUs were set to 2. A combined branch predictor with a meta table size of 2048. The error percentage in CPI and the speed-up obtained due to the use of simulation points are given in Figures 3 and 4 . Clearly, performing

the simulation using the generated simulation points results in considerable speedup without much loss in the accuracy, reducing machine weeks of time to a few hours. The CPI values obtained using simulation points was within 5 percent of the full run CPI values for all the benchmarks except 401.bzip where the value was off by around 8 percent. Even the error in Data, Instruction cache miss rates, DTLB miss rates and the branch misprediction ratios were within a limit of 5 percent for most of the benchmarks excepting bzip and libquantum that have an error of 11% and 13% for the branch missprediction rates. Figures 4, 5, 6, 7 show the errors in the values of CPI, branch mispredictions, data cache, instruction cache and DTLB miss rates for a set of benchmarks. Though the concept of simulation points have been widely used in various studies about caches, branch predictors etc., this is the first attempt towards validating and studying the TLB characteristics based on simulation points. It is quite evident from the results that these simulation points are representative of the whole benchmark even in terms of the TLB characteristics. Though the methodology used by SimPoint is micorarchitecture independent, this validation is performed by taking one specific platform (alpha) as a case study and the error rates may vary for other platforms.

We hope that these simulation points that are provided [3] will serve as a powerful tool aiding in carrying out faster simulations using the large and representative benchmarks of the SPEC CPU2006 Suite. The reference provided has the simulation points for 21 benchmarks and we are in the process of generating the remaining simulation points, which will also be added to the same reference.

SPECINT Benchmark	No. of Sim-points	Total No. of Instructions (in Billions)	Simulation time	SPECFP Benchmark	No. of Sim-points	Total No. of Instructions (in Billions)	Simulation time
401.bzip	21	213.9	15 hrs	410.bwaves	14	2317.9	7 days
445.gobmk	19	430.7	22 hrs	435.gromacs	20	2387	5 days 17 hrs
456.hmmer	9	2593.1	7 days	437.leslie3d	24	2609.3	7 days 3 hrs
458.sjeng	12	3187.7	9 days	444.namd	21	2223.8	6 days
462.libquantum	20	1989	5 days 13 hrs	447.dealII	3	3.7	13 minutes
464.h264ref	14	5663	8 days 6 hrs	450.soplex	21	486.4	1 days 10 hrs
471.omnetpp	11	729.9	2 days16 hrs	482.sphinx3	21	4004.8	9 days 22 hrs
473.astar	9	966.5	3 days 17 hrs	459.GemsF	25	223.6	17 hrs
400.perlbench	14	184.5	2.6 hrs	434.zeusmp	26	1992	1 day
				433.milc	21	1222.4	15 hrs
				436.cactusA	9	4560.1	14 days 7 hrs

Fig. 2. SPEC CPU2006 - Number of simulation points, total number of instructions and the simulation time taken by the Simfast simulator of the SimpleScalar LLC. It is to be noted that Simoutorder will take an order more time than Simfast

Benchmark	Simpoint run	Full run	Speedup
445.gobmk	5.53 hrs	34 days	148
450.soplex	3.62 hrs	77 days	502
473.astar	11 hrs	78 days	171
471.omnetpp	10.21 hrs	91 days	213
401.bzip	5 hrs	11 days	53.3
447.dealII	2.6 mins	1.9 hrs	42.6
400.perlbench	3.7 hrs	4 days 13 hrs	29.6
462.libquantum	19 hrs	42 days	53

Fig. 3. Speedup obtained by using the simulation points. The simulation point runs were done on the Texas Advance Computing Center and the full runs on a quad core 2 Ghz Xeon Processor

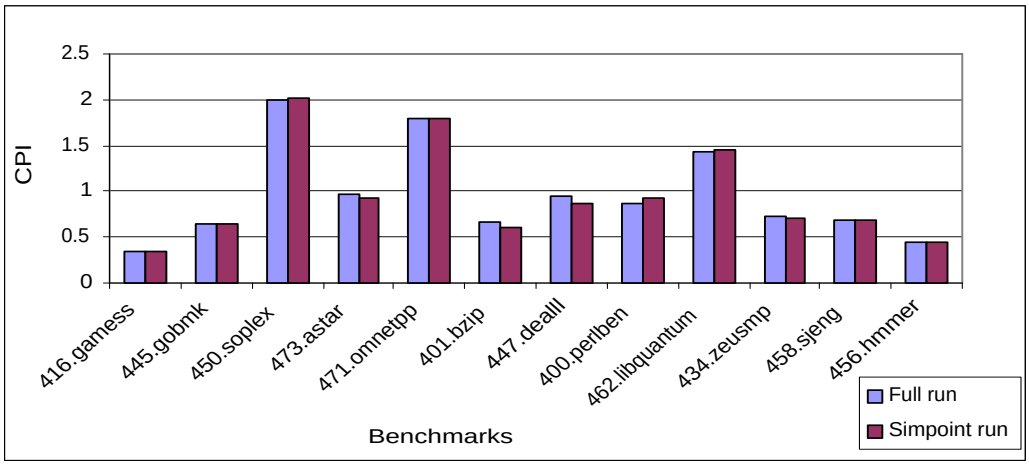

Fig. 4. CPI comparison between full runs and simulation point runs

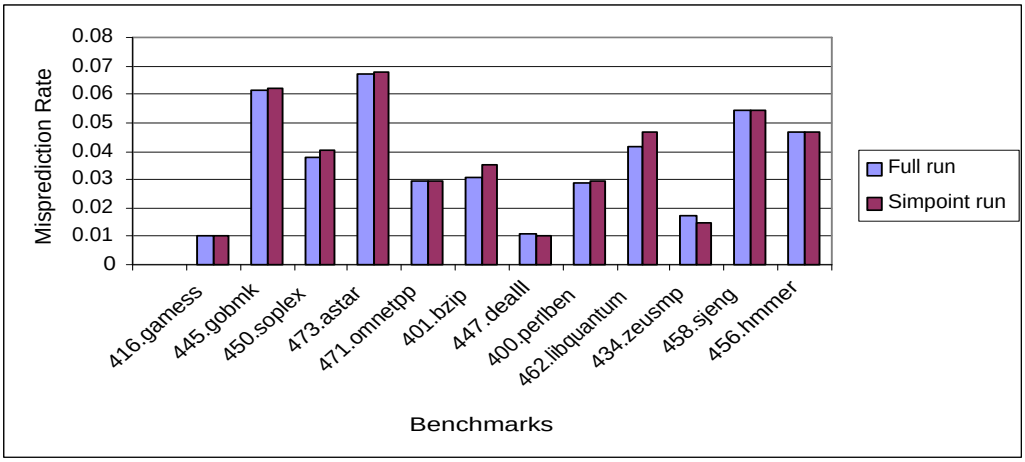

Fig. 5. Branch misprediction rate comparison between full runs and simulation point runs

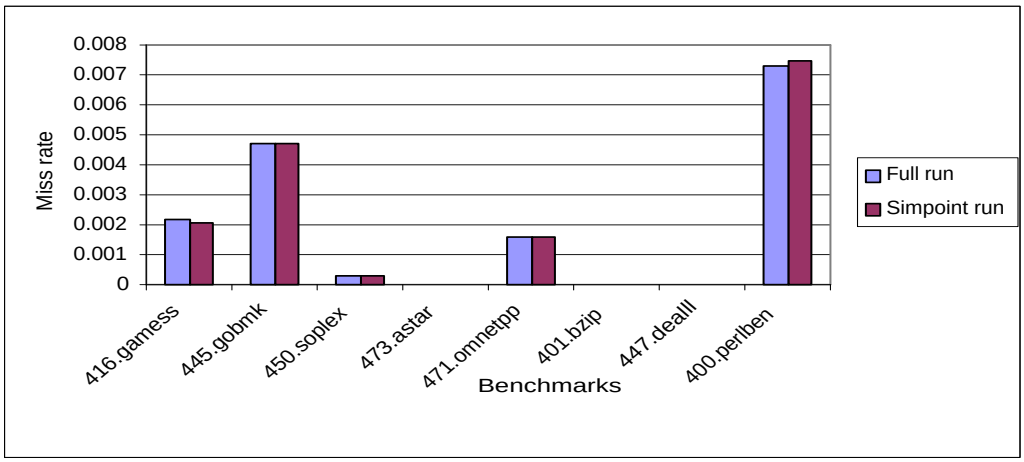

Fig. 6. Instruction cache miss ratio comparison between full runs and simulation point runs

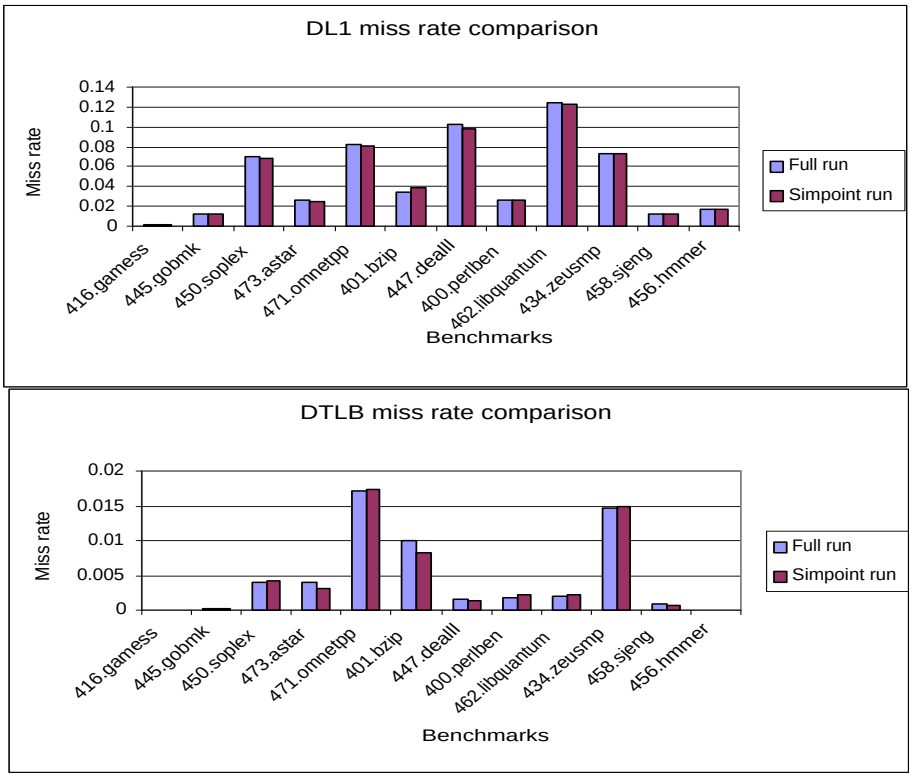

Fig. 7. Data cache and DTLB miss rate comparison between full runs and simulation point runs

5 Simulation Results and Analysis

5.1 Branch Characteristics

As mentioned earlier, sim-outorder supports both static and dynamic branch predictors. static predictors are quite ideal for the embedded applications due to their simplicity and low power requirements. Static predictors are also employed in designing simple cores in case of single chip multiprocessors like Niagara [15], where there exists strict bounds on area and power consumption on each core. It is also commonly used as backup predictors in superscalar processors that require an early rough prediction during training time and when there are misses in the Branch Targe Buffer. On the other hand, dynamic predictors give superior performance compared to the static ones but at the cost of increased power and area, as implemented in the modern complex x86 processors.

Fig. 9 shows the CPI results for two common type of static branch predictors viz., Always Taken and Always Not-Taken. As expected, it is clear from Fig. 9 and Fig. 11 that the performance of static predictors is quite poor compared to the perfect predictor. Always taken has the overhead in branch target calculation, but most of the branches in loops are taken.

Fig. 10 shows the CPI results for some common dynamic branch predictors. In this paper, we have studied the performance of the following dynamic predictors viz., Bimodal, Combined, Gshare, PAg and GAp. The configurations that were used for these predictors respectively are,

- Bimodal - 2048
- Combined - 2048 (Meta table size)
- Gshare - 1:8192:13:1
- PAg - 256:4096:12:0
- GAp - 1:8192:10:0

Gshare, PAg and GAp are 2level predictors and their configurations are given in the format {l1size:l2size:hist_size:xor}. Clearly, the CPI values obtained using dynamic predictors is much closer to the values obtained from the perfect predictor. Also, among these predictors, Gshare and Combined branch predictors performs much better compared to others. Taking a closer look at the graphs, we see that the Gshare predictor is ideal in the case of FP benchmarks while combined predictors fares better for the integer benchmarks. Also, PAg performs better than GAp predictor which indicates that a predictor with a global Pattern History Table (PHT) performs better than one with a private PHT. This clearly shows that constructive interference in a global PHT is helping the modern workloads and results in an improved CPI.

Looking at the performance of the private and the global configurations of the Branch History Shift Register (BHSR), it is evident that each of them perform well on specific benchmarks. Fig. 12 shows the misprediction rates for the different dynamic predictors. The performance improvement in CPI and Misprediction rate by using a dynamic predictor to a static predictor is drastic for the cases of *471.omnetpp* and *416.gamess*. Both of these benchmarks are pretty small workloads, that their branch behavior is easily captured by these history based Branch Predictors. *462.libquantum* and *450.soplex* also have a significant improvement in the CPI compared to their static counterparts, which can be attributed to

fact that the dynamic predictors are able to efficiently capture the branch behavior of these benchmarks.

For the purpose of analyzing the effect of PHT size on the behavior of the programs, we chose one of the best performing predictors obtained in the previous analysis i.e. Gshare and varied the size of it's PHT. We used PHT of index 12, 13 and 14 bits and observed the improvement in both CPI and branch misprediction rate (Fig 13. & 11). Different benchmarks responded differently to the increase in the PHT size. It can be observed that the integer benchmarks respond more to the increase in the PHT size compared to the floating point benchmarks. The floating point benchmarks have the least effect on the CPI for the increase in the PHT size. This is because of the fact that the floating point benchmarks have lesser number of branches and thus their behavior can be captured with a smaller PHT.

For instance, considering *435.gromacs*, although there is a significant reduction in the misprediction rate with an increase in the PHT size, there is not much improvement observed in the CPI. After analyzing this benchmark, we found that *435.gromacs* has only 2 percent of the instructions as branches. So, improving the accuracy of branch predictor does not have much effect on the CPI of the FP benchmarks. On the other hand, for the case of *445.gobmk* which is an integer benchmark, the improvement in misprediction rate shows a proportional change in the CPI. This is expected since *445.gobmk* has higher percentage of branches (15 percent) to the total instructions.

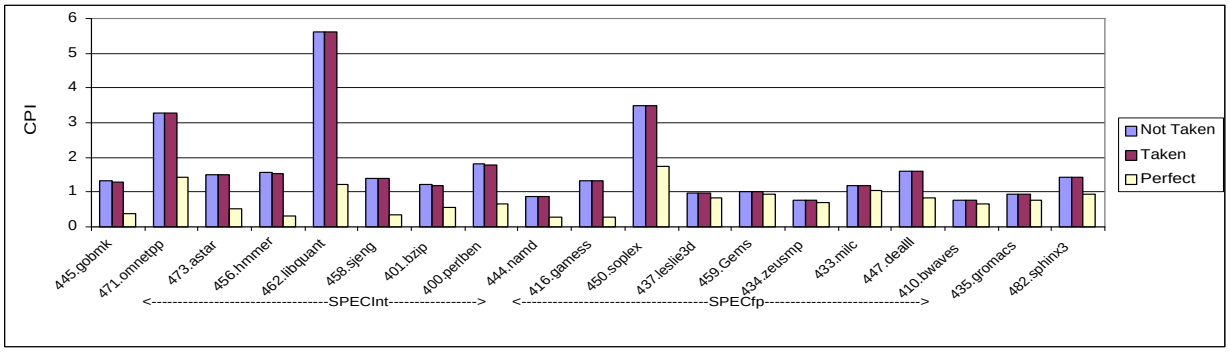

Fig. 9. Static branch predictor CPI

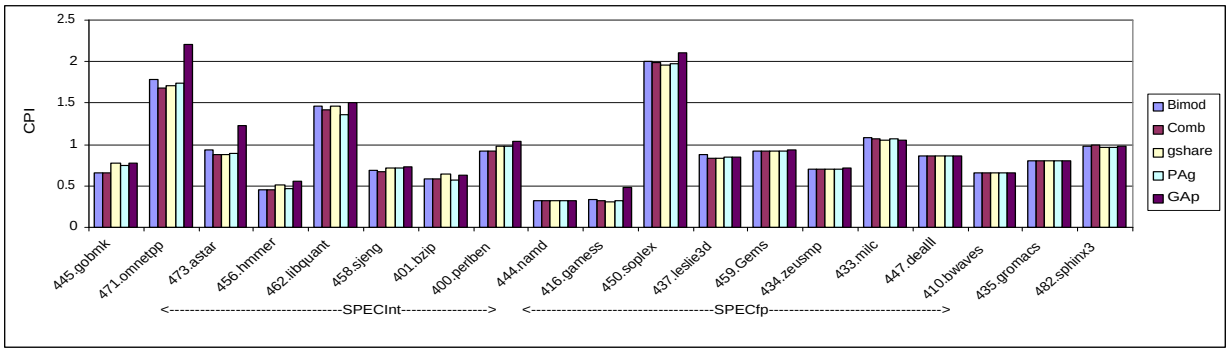

Fig. 10. Dynamic branch predictor CPI

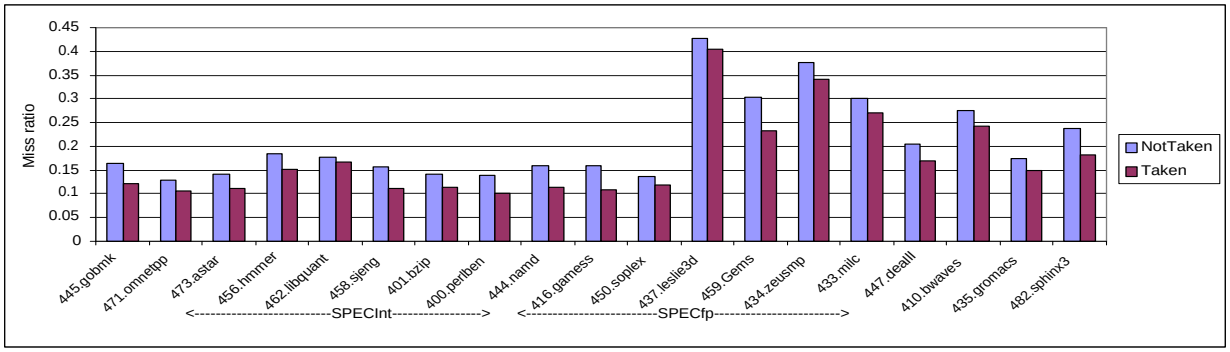

Fig. 11. Static branch predictor misprediction rate

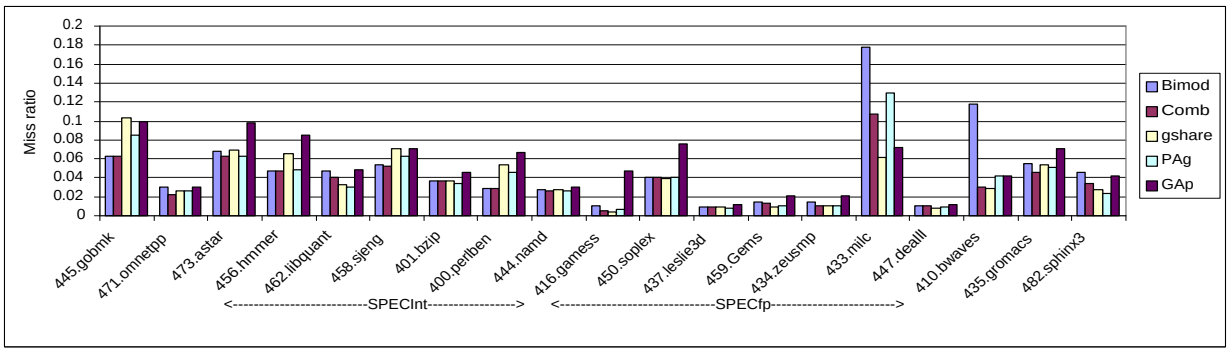

Fig. 12. Dynamic branch predictor misprediction rate

5.2 Memory Characteristics

The memory hierarchy design is of paramount importance in modern superscalar processors because of the performance loss due the Von Neumann bottleneck. It necessitates the need to come up with the optimal cache design parameters, so that it is capable of hiding the memory latencies efficiently. In this paper, we analyzed both the instruction and data level I caches and tried to come up with the optimal design parameters.

For the purpose of analyzing the L1 caches, we varied both the cache size and the associativity and compared the values of CPI and the miss ratios. We used the LRU replacement policy for all our experiments which is given as one in specifying the configuraion of the cache in the figures. From the graph in Fig. 15 & 16, it is evident that the effect of increasing associativity has a prominent effect on the performance than just increasing the size of the data cache. For some benchmarks like *445gobmk*, increasing the associativity to 2 result in a colossal reduction in the miss ratios, which can be attributed to smaller foot prints of these benchmarks. Other benchmarks where associativity provided significant benefit are *456.hmmer*, *458.sjeng* and *482.sphinx3* in which case increasing the associativity to 2 resulted in more than 50 percent reduction in miss ratio. However, some benchmarks like *473.astar* and *450.soplex* responded more to the size than associativity. It can be concluded that *473.astar*

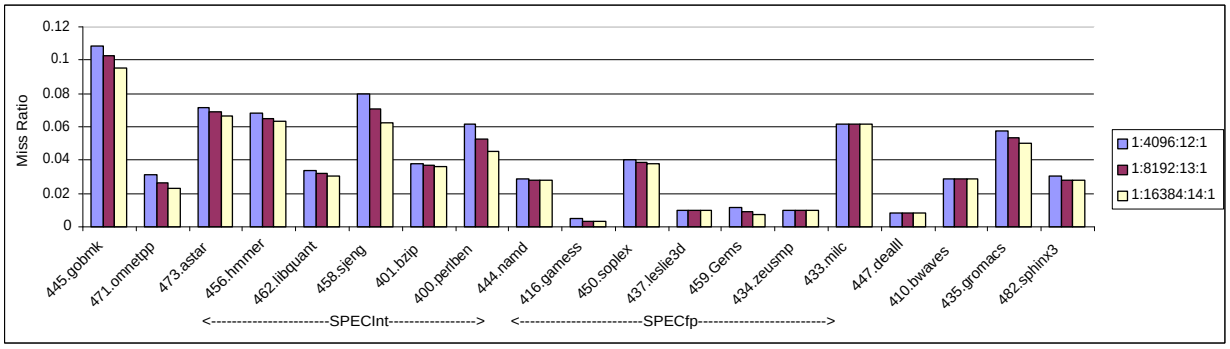

Fig. 13. Misprediction rate for Gshare configurations given as L1size:L2size:hist size & xor

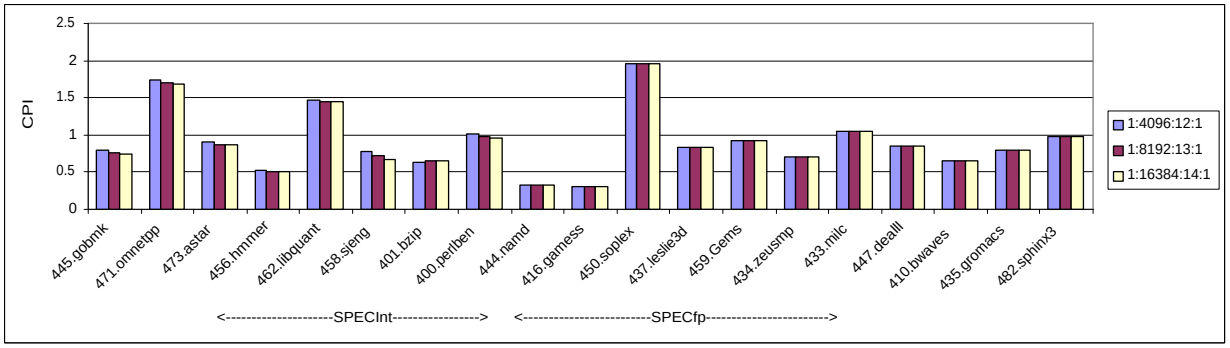

Fig. 14. CPI for Gshare configurations given as L1size:L2size:hist size & xor

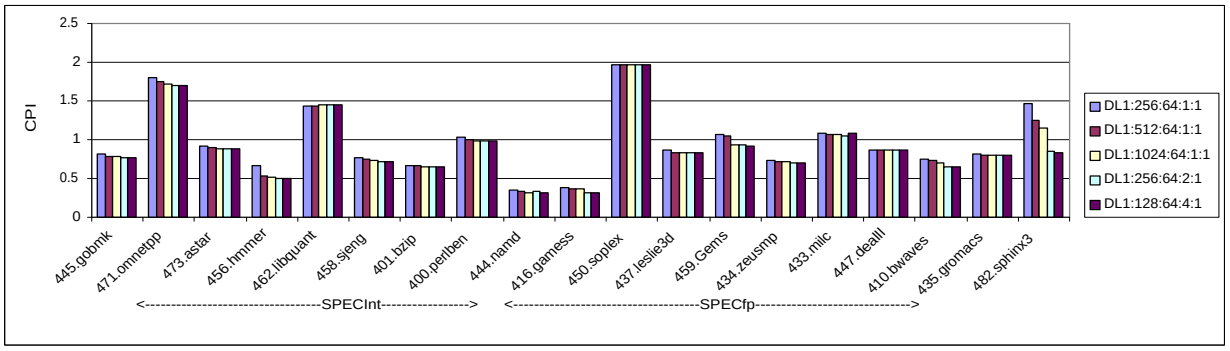

Fig. 15. CPI for DL1 configurations in format name:no.sets:blk size:associativity&repl. policy

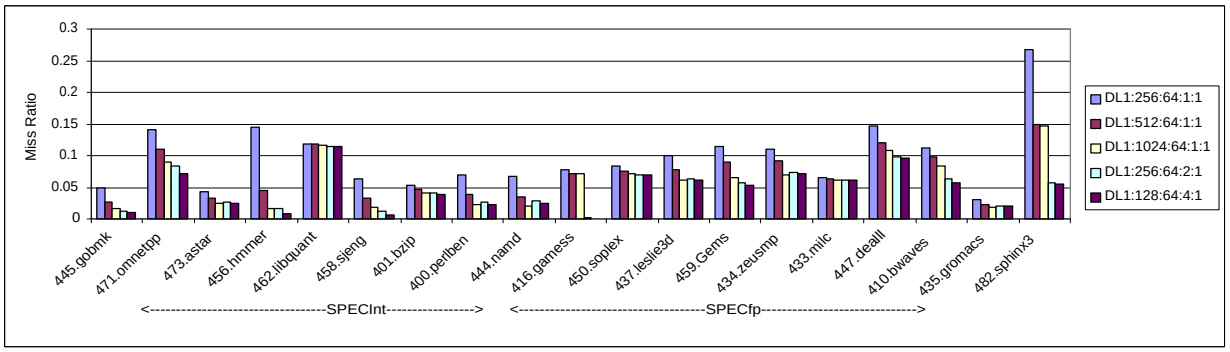

Fig. 16. Missrate for DL1 configs in format name:no.sets:blk size:associativity&repl. policy

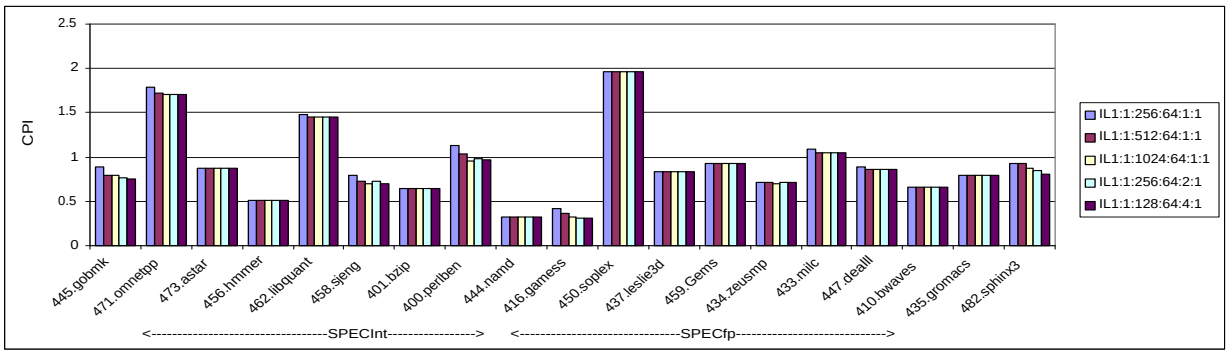

Fig. 17. CPI for IL1 configs in format name:no.sets:blk size:associativity&repl. policy

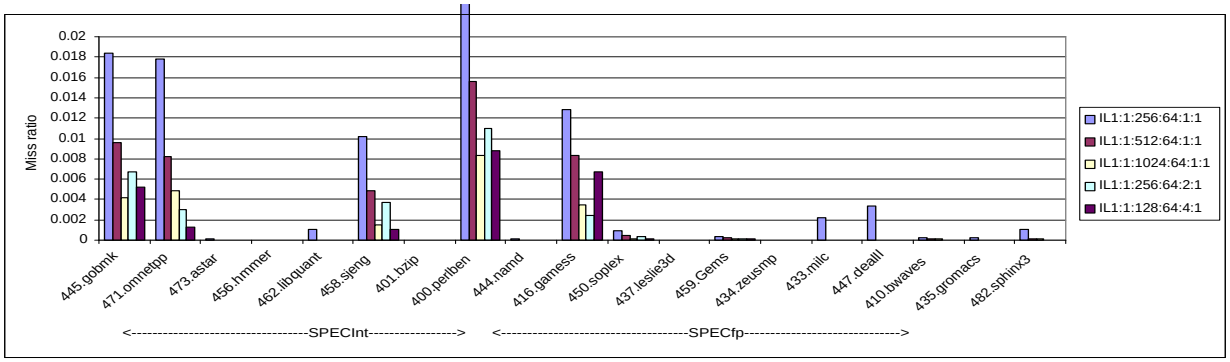

Fig. 18. Missrate for IL1 configs in format name:no.sets:blk size:associativity&repl. policy

and *450.soplex* has lot of sequential data and hence we cannot extract much benefit by increasing the associativity. The CPIs of the benchmarks *462.libquantum* and *433.milc* neither respond to the increase in the cache size nor to that in associativity. This may be due to a smaller memory footprint of these benchmarks which can be captured completely by just a small direct mapped cache.

The CPI and the miss ratios for different Level 1 instruction cache configurations are shown in Fig. 17 and 18. As expected, the miss ratios of the instruction cache is much lesser than that of the data cache because of the uniformity in the pattern of access to the instruction cache. For some of the benchmarks like *473.astar*, *456.hmmer*, *435.gromacs*, the miss ratio is almost negligible and hence further increase in the cache size or associativity does not have any effect on the performance. The performance benefit due to increase in associativity compared to cache size in instruction cache is not as much as the data cache. This is because of the fact that the instruction cache responds more to the increase in the cache size to that of associativity because of high spatial locality in the references. Considering the tradeoff between the performance and complexity, an associativity of two at the instruction cache level seems to be optimal.

5.3 TLB Characteristics

Although designing the data cache is an important step in processor design, it has to be coupled with an efficient TLB usage to achieve good performance. Choosing the TLB page size is becoming critical in modern memory intensive workloads with large foot prints. This can

be attributed to the recent addition of features like multiple page sizes to modern operating systems.

Using Simplescalar, we performed simulations on the SPEC 06 suite for different TLB page sizes & associativities and observed the TLB miss ratio, which characterizes the part of the CPI due to the time incurred in page translation.

First, we fixed the page size as 16KB and varied the associativity to see the corresponding impact on miss ratios and CPI. As expected, the direct mapped TLB has performed worse than the 2-way and 4-way TLBs as seen in Fig. 19 & 20. It looks like the improvement in the performance from 2-way to 4-way is not much and is not worth the extra hardware complexity required for the same. Thus, an associativity of two seems to be optimal for the modern workloads. As we increased the TLB size from 16KB to 16MB, we found that the change in associativity did not have any effect on the performance and this can be attributed to the fact that a page size of 16MB is large enough to reduce the conflict misses to zero.

Second, we performed simulations with various page sizes for a 2-way associative TLB. Our results as shown in Fig. 21 & 22 had a close match with that of the results specified in [16] for a power5 processor. We found that large page sizes resulted in the least translation misses, leading to a better CPI. Firstly, it can be observed that there is a reduction in the TLB miss ratio around 30% for *471.omnetpp*, 80% for *473.astar* when the page size is increased from 4KB to 16KB. There is a consistent improvement in the performance of all the benchmarks for an increase in the page size. When a page size of 16MB is used, the TLB misses reduces to nearly zero for most of the benchmarks except *445.gobmk* and *450.soplex*. One possible cause for the increase in CPI for *445.gobmk* and *450.soplex* for a 16MB page size could be due to serious wastage of memory caused due to internal fragmentation problems. Other reasons could be having higher numbers of conflicts amongst the cache lines if the virtual address bits used in cache tag matches are insufficiently distinct from each other under larger sized TLB mappings.

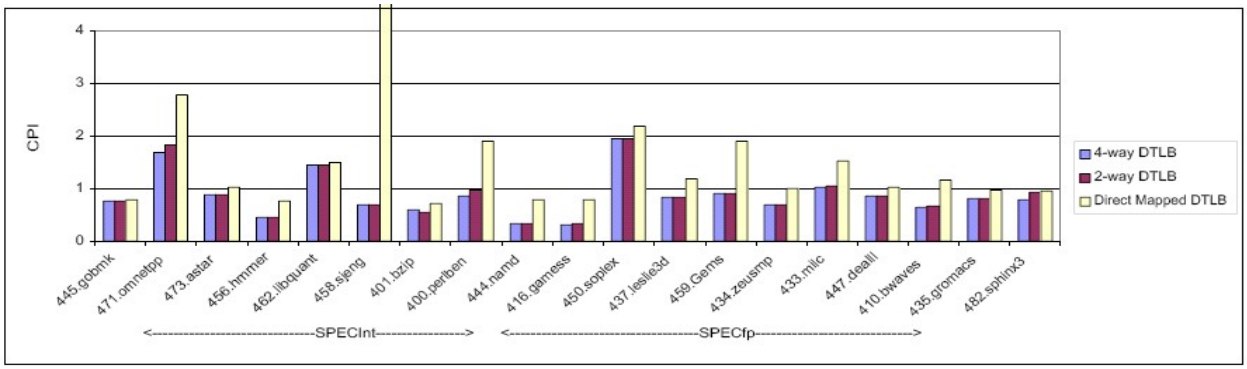

Fig. 19. CPI for varying associativity with 16KB page sizes

6 Conclusion

The simulation points have proved to be an effective technique in reducing the simulation time to a large extent without much loss of accuracy in the SPEC CPU2006 Suite. Using

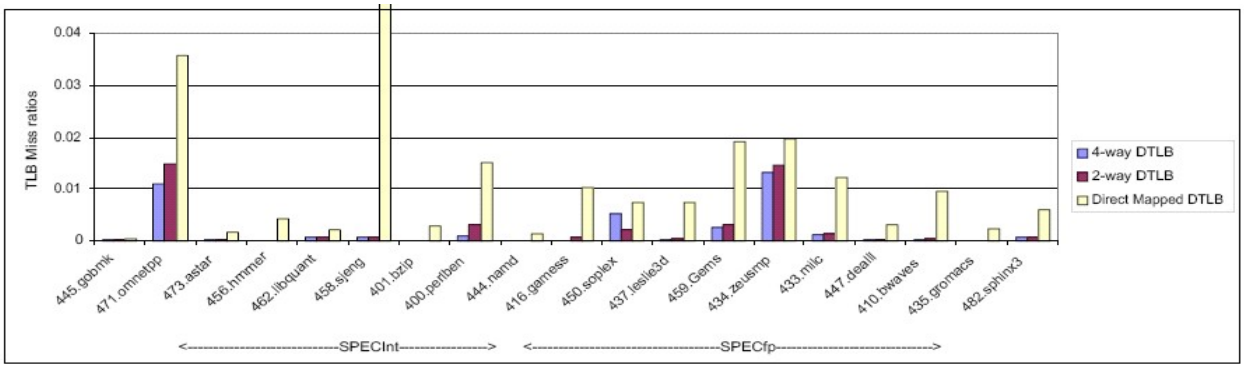

Fig. 20. TLB miss ratios for varying sssociativity with 16KB page sizes

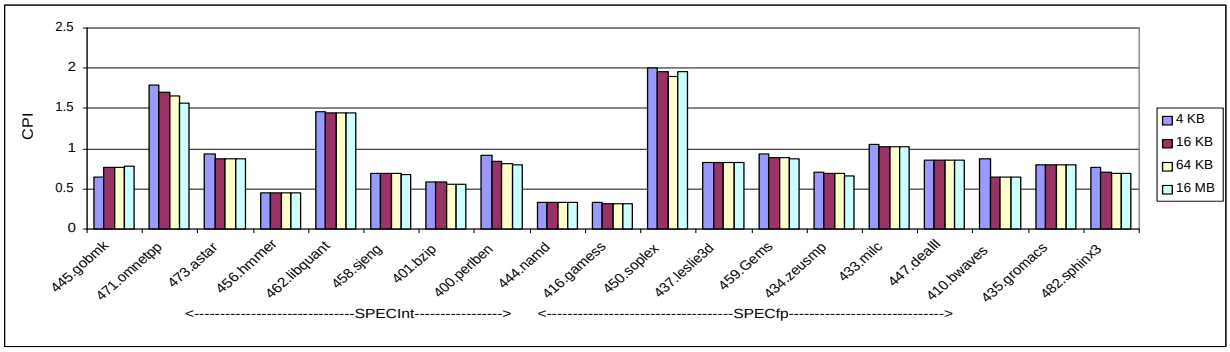

Fig. 21. CPI for varying page sizes with 2-way associative TLB

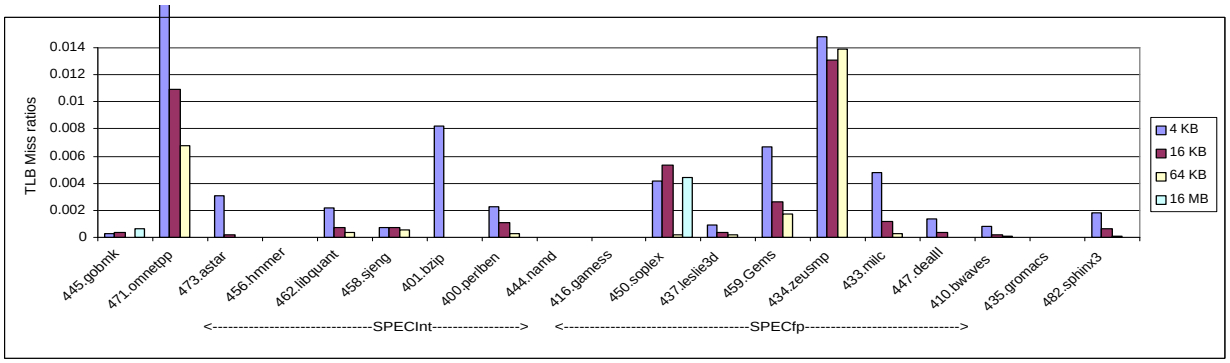

Fig. 22. TLB miss ratio for varying page sizes with 2-way associative TLB

simulation points not only reduces the number of dynamic instructions to be simulated but also makes the workload parallel, making them ideal for the current day parallel computers.

Further, simulating the different benchmarks with the different branch predictors, gave an insight into understanding the branch behavior of modern workloads, which helped in coming up with the best performing predictor configurations. We observed Gshare and the combined (Bimodal & 2-level) to be the ideal predictors, predicting most of the branches to near perfection. Looking at the effect of different cache parameters, it is observed that the design of level-1 data cache parameters proves to be more important in affecting the CPI than that of the instruction cache parameters. Instruction accesses, due to their inherent uniformity, tends to miss less frequently, which makes the task of designing the Instruction cache much easier. The line size of the Instruction cache seems to be the most important, while for the data cache, both the line size and the associativity needs to be tailored appropriately to get the best performance. The simulations for the different TLB configurations revealed that usage of large page sizes significantly reduce the translation misses and aid in improving the overall CPI of the modern workloads.

7 Acknowledgement

We would like to thank the Texas Advance Computing Center (TACC) for the excellent simulation environment provided for performing all the time consuming simulations of SPEC CPU2006 with enough parallelism. Our thanks to Lieven Eeckhout and Kenneth Hoste of the Ghent University, Belgium for providing us the alpha binaries for the SPEC suite. This work is also supported in part through the NSF award 0702694. Any opinions, findings and conclusions expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

References

1. T. Sherwood and B. Calder. Time varying behavior of programs. *Technical Report UCSD-CS99-630, UC San Diego,*, August 1999.
2. Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. *ASPLOS*, October 2002.
3. http://www.freewebs.com/gkofwarf/simpoints.htm.
4. SPEC. Standard performance evaluation corporation. http://www.spec.org.
5. John L. Henning. SPEC CPU2000: Measuring cpu performance in the new millennium. *IEEE Computer, vol. 33, no. 7, pp. 28-35*, Jul 2000.
6. M.J.Charney and T.R.Puzak. Prefetching and memory system behavior of the SPEC95 benchmark suite. *IBM Journal of Research and Development, 41(3)*, May 1997.
7. J. Haskins and K. Skadron. Minimal subset evaluation: warmup for simulated hardware state. *Proceedings of the 2001 International Conference on Computer Design,*, September 2001.
8. Aashish Phansalkar, Ajay Joshi, and Lizy K. John. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. *The 34th International Symposium on Computer Architecture (ISCA)*, June 2007.
9. Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. Simpoint 3.0: Faster and more flexible program analysis. *Workshop on Modeling, Benchmarking and Simulation*, June 2005.
10. Greg Hamerly, Erez Perelman, and Brad Calder. How to use simpoint to pick simulation points. *ACM SIGMETRICS Performance Evaluation Review*, March 2004.
11. Erez Perelman, Greg Hamerly, and Brad Calder. Picking statistically valid and early simulation points. *International Conference on Parallel Architectures and Compilation Techniques*, September 2003.
12. Tse-Yu Yeh and Yale N.Patt. Alternative implementations of two-level adaptive branch prediction. *19th Annual International Symposium on Computer Architecture*, May 1992.

13. Jeremy Lau, Jack Sampson, Erez Perelman, Greg Hamerly, and Brad Calder. The strong correlation between code signatures and performance. *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2005.
14. E. Perelman, T. Sherwood, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. *International Conference on Parallel Architectures and Compilation Techniques*, September 2001.
15. Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded sparc processor. *MICRO (Vol. 25, No. 2) pp. 21-29*, April 2005.
16. Wendy Korn and Moon S. Chang. SPEC CPU 2006 sensitivity to memory page sizes. *ACM SIGARCH Computer Architecture News*, March 2007.