The Relative Importance of Memory Latency, Bandwidth, and Branch Limits to Performance

Norman P. Jouppi
Digital Equipment Corporation
Western Research Lab
250 University Avenue
Palo Alto, California 94301
jouppi@pa.dec.com

Parthasarathy Ranganathan
Dept. of Electrical and Computer Engineering
Rice University
Houston, Texas 77005
parthas@ece.rice.edu

Abstract

This study investigates the relative importance of memory latency, memory bandwidth, and branch predictability in determining limits to processor performance. We use an aggressive simulation model with few other limits to study the performance of SPEC92 benchmarks. Our basic machine model assumes a dynamically scheduled processor with a 16536 entry instruction window. Up to 16536 instructions of any type can be issued each cycle, subject to data dependencies.

In systems with unlimited memory bandwidth and perfect branch predictability, we find that memory latency is not a significant limit to performance until it exceeds 100 to 200 cycles. Memory bandwidth is not usually a significant limit either. In systems with memory latency of 16 cycles and perfect branch predictability, many applications require less than 6 bytes per cycle, while all but one perform well if 100 bytes per cycle are available. Based on current trends in the semiconductor industry and current research in packaging and interface technology, we expect these latency requirements and bandwidth requirements to be achievable in future processors.

By far, the biggest limit to performance was branch unpredictability. The use of one of the best existing branch predictors provided with very large table sizes resulted in performance three to five times less than that achievable with perfect branch prediction for many benchmarks. Additionally, many benchmarks had significantly lower performance than the perfect case even with a futuristic branch predictor (with a mix of 75% perfect and 25% existing) was simulated. These results suggest that improved branch prediction techniques are crucial for improving future uniprocessor performance.

1 Introduction

The performance of microprocessors has now been increasing at a rate of approximately 60% a year for more than a decade. Many different obstacles could limit the eventual performance of microprocessors. Some of these include memory bandwidth, memory latency, control-flow dependencies, design complexity, fabrication process limits, as well as economic limits such as microprocessor and fabrication process development costs. In this paper we compare the relative importance of the first three possible limits to performance.

Possible limits to processor performance imposed by memory systems (the "Memory Wall") have recently received widespread attention. Unfortunately the original "Memory Wall" study [7] was based on very simplistic models, in particular a non-pipelined memory system with only one memory operation in progress at a time. High-end processors have had access to pipelined memory systems since the 1960's [15, 16], and more recently even PC's have had the benefit of pipelined memory systems [5]. The assumption of non-pipelined memory systems puts a profoundly small and unrealistic limit on possible processor performance. In a non-pipelined memory system, the bandwidth is roughly equal to the bus width times the inverse of the latency. In pipelined memory systems, memory latency and bandwidth are independent variables. In this study we consider both memory latency and bandwidth as possible limits to performance. In recent terms, these could be called the "Memory Bandwidth Wall" and a "Memory Latency Wall".

Over the last three decades, work on instruction-level parallelism has made varying assumptions about control-flow hazards. Some work assumed that all branches were perfectly predictable, which yielded very large bounds on available instruction-level parallelism[13]. Other work considered only parallelism within basic blocks, which resulted in small estimates of instruction level parallelism. More recent work has assumed some type of practical branch prediction[2, 23, 3]. As commercial processors issue ever greater numbers of instructions per cycle, the control flow dependence problem is likely to impose a greater limit on processor performance. In terms of recent nomenclature, this is the "Branch Wall" problem.

This study seeks to investigate the relative importance of memory latency, memory bandwidth, and branch predictability in determining limits of processor performance. Limits studies are useful for suggesting research areas that are likely to be important in the future. Although the performance numbers obtained in limits studies today may sound farfetched, the thought of a 500MHz quad-issue dynamically scheduled machine such as the DEC 21264 would have been similarly hard to imagine a decade ago when a single-issue 16MHz MIPS R2000 was the state of the art.

The next section reviews technology trends in processors and local interconnects to put our limits study in context, and briefly reviews previous work. Section 3 describes our experimental framework. Section 4 examines memory latency, memory bandwidth, and branch predictability induced limits and discusses our results. Section 5 summarizes the paper and suggests future work.

2 Technology Trends and Related Work

2.1 Technology Trends

Semiconductor. The number of transistors on a microprocessor die has been increasing at a rapid rate, almost doubling every two years. Similarly, the operating frequency of microprocessors has been increasing at approximately 30% every year. The semiconductor industry association has a roadmap for continued scaling at the historic rates for at least 15 more years. In reality this scaling may slow down or have certain delays, but the eventual attainment of the scaling goals is highly likely. In addition to these technology trends, there has been a number of innovations in microprocessor design to increasingly use instruction-level parallelism (ILP) to improve performance. The net result of these architectural and technological innovations has been to increase microprocessor performance at a rate of 60% per year. This has resulted in a compound improvement in microprocessor performance of about 100X per decade.

Interconnects. The number of pins on a microprocessor has been increasing relatively slowly, at 16% per year [19]. Moreover, most PC's only have busses that run at a fraction of the CPU clock speed (e.g., 1/4, typically 50-66Mhz). However, techniques such as the RAMBUS memory interface[24] have shown that it is possible to achieve signaling rates of about 2X faster than CPU clock rates in similar technology. By using fully differential signaling, as is normally used in communications applications, even higher signaling rates can be achieved. The current highest speeds in communications applications approach 50GHz[20]. Recently, work on 4GHz differential signaling in 0.5um CMOS has begun[21]. Communication speeds in CMOS should scale with the processor clock frequency, assuming the package electrical characteristics are sufficient.

Pins on microprocessor pin-grid array packages have typically presented very large impediments to very high-speed transmission line signaling. However, ball-grid arrays with very small impedance discontinuities offer the potential for very large numbers of very-high quality interconnects. Small bumps have been used in microwave circuits for connections up to 70GHz[22]. Manufacturers have recently announced plans to support packages with up to 1500 bumps[25]in the near future.

2.2 Related Work

We are unaware of any other studies focusing on the relative impact of memory system latency, memory system bandwidth, and control-flow predictability on the limits of processor performance, although there has been previous work in individual areas.

Burger, Goodman, and Kägi [19] show that techniques to reduce latency increase the bandwidth requirements of the system. They then present an excellent comparison of many different techniques for reducing memory bandwidth requirements. They contend that pin bandwidth is likely to be the dominant bottleneck for performance of future-generation processors and propose techniques like processor/memory integration and flexible caching policies to solve this problem. However, they assumed processor I/O busses with 1/4 the processor clock frequency for advanced chips and at most 128 bit wide busses. They also assumed instruction reordering comparable to the existing state-of-the-art. Even with these limits, most applications spent less than 20% of their time in memory related stalls.

Theobald, et. al. [23], in their study on the effects of resource limitations on program parallelism, looked at the impact of control dependences and memory latencies on performance. They evaluated five different branch prediction strategies. However, they used a probabilistic model for their memory system to mark random memory accesses as misses. This model does not account for the clustering of misses in an application, which as our results show, has significant impact on the performance variation with latency.

Finally, Wall's study on limits to instruction-level parallelism [2] is by far the most detailed study on the subject and studies the impact of various parameters like register renaming, alias analysis, branch and jump prediction, window size, cycle width, and functional unit latency on 18 benchmarks. However, this study also does not model a detailed memory system and assumes that all load instructions take 2 cycles.

3 Experimental Framework

This section describes the experimental framework used in this paper. We use trace-driven simulation for this study. Traces are collected at run-time using a flexible code-instrumentation interface provided by ATOM [1]. ATOM instruments the executable code to call the simulator before every instruction (we do not instrument special instructions like barriers, etc.). The instrumentation code converts the instructions into a dynamic flow graph which is then used to simulate the processor, memory, and branch prediction models. The simulation methodology is similar to that used in many previous studies [2, 3]. Sections 3.1, 3.2, and 3.3 describe the processor, memory, and branch prediction models used in this paper and Section 3.4 describes the applications used.

3.1 Processor Model

The processor model implements an aggressive superscalar architecture based on the Alpha instruction-set architecture. The processor supports non-blocking loads and stores, speculative execution, out-of-order execution, and register renaming. Such features already exist in current processor designs [4, 5, 6]. In addition, we assume an instruction window with 16536 entries and an issue rate restricted only by the window size. This window size was chosen because it was two orders of magnitude larger than the typical IPC in our limits simulations. Thus there would typically be plenty of window entries for good instruction scheduling. Larger window sizes than this would provide little benefit but would slow the simulations down. Instructions whose dependences have all been resolved are issued in the earliest possible instruction word. The large window size is also meant to model some of the effects possible with large-scale code motions performed by advanced compilers. For example, loop interchange can result in instructions being moved past thousands of other instructions.

We assume single cycle functional unit latency for all non-memory operations. Additionally, we assume that all memory references are checked at run-time for aliases (similar to the perfect alias analysis assumed in [2]). Our assumptions are no doubt more aggressive than existing sys-

Branch prediction *(label)*	Notation	Size
bimodal *(2c)*	2c(1K)	2Kbits
gshare *(Gsh)*	gshare(12,12)	8Kbits
bimodal/gshare *(2c/g)*	2c(1K)/gshare(12,12)/sel(4K)	18Kbits
local/gshare *(pa/g)*	PA(4K,12,1)/gshare(12,12)/sel(4K)	72Kbits
local/gshare/big *(PA/G)*	PA(32K,15,1)/gshare(15,15)/sel(32K)	672Kbits
Futuristic *(Futr)*	25% loc/gsh/big, 75% perfect	Infinite
Perfect *(Perf)*	Oracle-based	Infinite

Figure 1: Branch prediction strategies used.

tems. However, the purpose of this study is to determine the *limits* on the instruction-level parallelism imposed by memory latency, memory bandwidth, and branch unpredictability. These assumptions thus give us a good upper bound on the performance that we can get with future-generation processors and compilers under these constraints.

3.2 Memory Models

We model separate data and instruction caches with a 100% hit rate in the instruction cache. The data cache is a non-blocking 64 Kilobyte four-way associative cache, with a 64 byte line size. Obviously a processor with unlimited branch prediction hardware and a 16K entry dispatch queue is likely to have several megabytes of cache on-chip. However, we chose only 64KB cache sizes since 64KB caches on our benchmarks are a reasonable approximation of future demanding applications on future 10MB on-chip caches. Many other non-SPEC benchmarks such as transaction processing place greater demands on the memory system. We hope that by using small caches, we come closer to approximating the behavior of these more demanding programs on multi-megabyte caches.

For our experiments, we look at three kinds of memory systems – perfect, non-pipelined and pipelined. The perfect memory system is ideal in all respects. It has no misses and suffers from no bandwidth restrictions. The non-pipelined memory system is similar to the memory system assumed by [7]. It can service only one request at a time and the bandwidth is limited by the fraction of the bus width to the latency. Multiple requests cannot be overlapped with one another. The third kind of memory system which we simulate is a pipelined memory system. We vary the bandwidth and the latency of the memory system independently for our experiments. The pipelined memory system has no limit on the number of outstanding requests other than that implied by the product of the specified bandwidth and latency divided by the cache line size.

We assume that the bandwidth of the system is provided using unidirectional busses. Unidirectional data busses can provide higher bandwidths than busses that switch directions. In all the systems we simulate, we assume there are unidirectional input and output data busses connected to the microprocessor. The outgoing data can be buffered and delayed with little consequence in uniprocessor systems given appropriate bypassing. Also, we only consider the effects of limited load bandwidth and latency. Store bandwidth is usually about half the load bandwidth and can be provided by making the output bus somewhere between the same width or half the width of the input bus.

3.3 Branch Prediction Strategies

We model various branch prediction strategies described previously in the literature[10, 9, 11, 12]. Figure 1 summarizes the branch prediction strategies used. The seven branch prediction strategies used in this study are *bimodal*, *gshare*, *bimodal/gshare*, *pa/gshare*, *PA/Gshare*, *futuristic*, and *perfect*. The sizes of the branch predictors were chosen to have increasing orders of aggressiveness and increased requirements of chip area to model successive generations of branch predictors.

Bimodal prediction uses a simple two-bit-saturating counter scheme for branch prediction. Our implementation has 1K two-bit counters.

Global branch prediction tries to exploit correlation between different branches and uses the global branch pattern history to predict branch direction. To optimize the use of the branch prediction table in the global branch prediction scheme, the *gshare* scheme uses an exclusive-or combination of the global history pattern and the branch address. Our implementation XORs 12 bits of global history and the low 12 bits of branch address to index into a table of 4K two-bit counters.

Hybrid branch predictors use a two-bit saturating counter to choose between two branch prediction strategies and use the branch predictor that is more likely to be correct. All our hybrid branch predictors use a two level branch predictor selection algorithm similar to the one used in [9]. We simulated three hybrid predictors. The first (*bimodal/gshare*) uses a table of 4K two-bit counters to select between the bimodal and gshare predictors described above. Local branch prediction tries to find and exploit patterns in the behavior of individual branches and uses the previous history pattern of a branch to predict its subsequent occurrences. The second hybrid predictor we simulated (*pa/gshare*) uses 4K two-bit counters to select between the gshare predictor described above and a local history based predictor with a 4K entry table of 12-bit histories that index 4K two-bit counters. The third predictor (which we call *PA/Gshare*) is a larger version of the second predictor. It has 32K two-bit counters that select between a local predictor with 32K 15-bit histories that index 32K two-bit counters and a 32K entry gshare predictor.

The *perfect* branch prediction strategy is an oracle-based predictor which always predicts correctly.

The *futuristic* branch predictor is meant to model a currently unknown future branch prediction strategy and is obtained by randomly choosing the perfect branch predictor for 75% of the branches and the best hybrid branch predictor (local/gshare with the largest table sizes) for the other 25% of the branches at run time. Other currently proposed techniques to improve the performance of branches (e.g. guarding [33], fanout-based prediction [2]) are not expected to

Application	Notes	Input	Load %	Branch %
Integer Applications				
Compress	Reduction of size of file using Lempel-Ziv coding	ref	26.2	13.34
Espresso	Minimization of a boolean function	bca	22.5	17.81
Eqntott	Translation of a boolean equation to truth table	ref	12.9	10.91
Floating point applications				
Doduc	Monte Carlo simulation of a thermo-hydraulic model	short	31.5	6.07
Tomcatv	Mesh generation program	ref	29.1	3.32
Mdljsp2	Solution of equations of motion for system of interacting atoms	short	29.5	8.59
Swm256	Solution of shallow-water equations using finite difference approx.	short	36.0	1.85
Hydro2d	Solution of hydrodynamic Navier-Stokes equations	short	21.7	7.83
Fpppp	Calculation of two-electron integral derivative	short	34.9	1.21

Figure 2: Applications used.

give more than 75% improvements in the branch prediction accuracy. The futuristic branch predictor can therefore be used as a reasonable approximation of the limits of branch prediction.

The prediction strategies described above apply for conditional branches. We assume that indirect jumps (procedure returns) are predicted correctly using Return Address Registers (RARs) as in current processors. (Previous studies [2] have shown that we can get almost perfect jump prediction with a small amount of hardware.)

If a simulated branch predictor mispredicts a branch, the simulator does not issue any instructions from after the branch until the branch condition is computed. Because of the large number of instructions in the dispatch queue, it may take many cycles for the true branch condition to be computed. Our simulation model approximates a machine where the only penalty of a mispredicted branch is the *number of cycles* that is lost executing down the wrong path. We do not model any good or bad effects from executing down wrong paths during this time. Once the true branch condition is available we assume that only one additional cycle is required to begin execution down the correct path. There are no restrictions on the number of branches that can be issued per cycle.

3.4 Applications

For understanding processor performance one to two decades in the future, we would like to simulate a range of workloads from the year 2010 (perhaps including SPEC10?) on processors which are likely to have many megabytes of on-chip cache. Since these applications do not exist yet, we have scaled our cache size assumptions to match the SPEC92 benchmarks (SPEC95 benchmarks could have been used, but would have required much larger on-chip cache size assumptions and would have taken much longer to simulate, due to the complexity of our out-of-order model).

We present data from 3 integer applications and 6 floating point applications from the SPEC92 benchmark suite. Figure 2 gives details about the applications used, the input sizes, and the percentage of load and branch instructions of the total instructions at run-time. We use the instructions per cycle (IPC) as the metric for measuring performance.

4 Results

The following section describes the performance trends obtained from our simulations on the applications described in Section 3.4. The impact of memory limits on performance is discussed in Section 4.1. Section 4.2 discusses the impact of control dependence on performance. Section 4.3 compares the relative importance of these effects.

4.1 Impact of Memory System on Performance

This section discusses the impact of the memory latency wall and the memory bandwidth wall on performance. All the results in this section assume a perfect branch prediction strategy (Section 4.3 examines the effect of other branch prediction strategies).

4.1.1 Memory Latency

Figure 4 shows the effect of varying memory latency on the performance of Compress. These results assume a perfect branch prediction strategy and a bandwidth of 64 bytes (one cache line) per cycle. These graphs plot the performance on the y axis and the memory miss latency on the x axis (on a logarithmic scale). The three curves in the figure represent the three memory systems considered in our study. The *perfect* memory system assumes an ideal memory system with all cache accesses being serviced in the first-level cache. The *non-pipelined* memory system models the memory system assumed in [7]. In this memory system, the bandwidth is the reciprocal of the latency times the cache line size, which implies that only one request can be serviced at a time. Subsequent requests will have to wait for all previous requests to be processed before they are serviced. The third curve shows the performance of a fully *pipelined* memory system.

The arrows in Figure 4 mark the knees of the curves and indicate the value of the memory miss latency at which the performance drops to $1/\sqrt{2}$ (0.707X) of that of the given memory system model using a latency of one cycle[1]. These knee values thus indicate the point at which the memory miss latency starts limiting the performance of the processor and give a quantitative indication of the memory latency wall.

Figure 3 summarizes the effect of varying memory latency and bandwidth on the performance for all the ap-

[1] For the latency curves, we use curve-fitting with equations modeling a low-pass filter ($y = d + \frac{c}{1+(\frac{x}{a})^n}$).

Application	Mem %	Total Miss% (Primary)	Perf IPC (ideal)	Lat knee non-piped	Lat knee pipelined	BW knee (bpc)	BW knee (bpi)
Compress	26.20	16.0 (11.9)	28.37	2	535	29	1.02
Espresso	22.50	2.5(0.4)	63.84	20	460	< 6	0.10
Eqntott	12.90	32.3 (2.9)	268.39	2	70	92	0.34
Doduc	31.45	3.6 (0.3)	77.77	15	90	< 6	0.08
Tomcatv	29.11	55.1 (5.6)	238.45	2	280	483	2.03
Mdljsp2	29.51	4.0 (0.03)	113.35	15	245	< 6	0.05
Swm256	35.96	63.0 (3.3)	170.09	2	265	143	0.84
Hydro2d	21.68	19.8 (1.8)	113.06	2	200	42	0.37
Fpppp	34.89	0.2 (0.1)	79.51	40	180	< 6	0.08

Figure 3: Summary of memory effects on performance

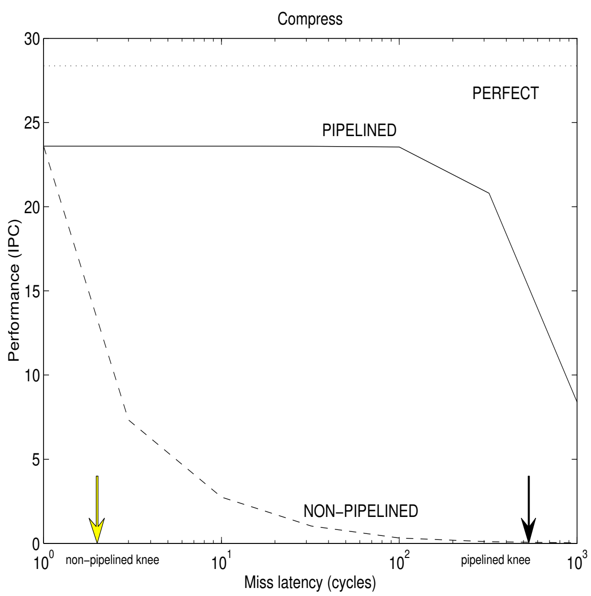

Figure 4: Impact of miss latency variation on performance – Compress

plications. The second column indicates the percentage of memory instructions in the program. The third column gives the miss rate expressed as a percentage. The value in parentheses represents the primary miss rate of the application. Primary misses [32] are misses to blocks that are not already being fetched by earlier (still outstanding) misses. The next column gives the IPC obtained from running this program under ideal conditions with a perfect branch prediction strategy and a perfect memory system and indicates the ILP inherent in the application for this issue rate and instruction window size. The next two columns summarize the knees of the latency curves for the non-pipelined and pipelined memory systems. The columns relating to bandwidth are explained in the next subsection.

The results demonstrate that the pipelined memory system is able to tolerate a relatively high increase in the miss latency before the performance degrades significantly. In contrast, the non-pipelined memory system exhibits degradation of performance even at relatively small increases in miss latency. This difference in the impact of increasing miss latency on the performance of the two models is mainly due to the overlap of the memory latency with other useful instructions in the pipelined memory system. In particular, the overlap of misses with other misses plays an important role in the amount of tolerance the system exhibits to increasing memory miss latency. The knees of the curves for both the pipelined and non-pipelined memory system are primarily governed by the percentage contribution of memory misses to the execution time and the parallelism available in the application. Most of the applications we consider in this study have enough inherent (without any control-related dependences) parallelism so that the use of an aggressive pipelined memory system is enough to tolerate fairly high increases in memory miss latency.

4.1.2 Memory Bandwidth

Figure 5 shows the impact of variation of bandwidth on the performance of Compress. These results are on a system with a base memory miss latency of 16 cycles and a perfect branch prediction strategy. The graphs plot performance measured by the IPC on the Y axis and the bandwidth measured in bytes per cycle on the X-axis (again on a logarithmic scale) for a pipelined memory system. The upper and lower bounding lines in the figure represent the perfect and non-pipelined memory systems respectively. We define the knee as before to get an idea of the point at which the bandwidth limits performance significantly[2]. Figure 3 summarizes the knees of the bandwidth curves for all the applications. The values in this table are measured in bytes per cycle as well as in bytes per instruction. The position of the bandwidth knee in bytes per instruction gives the knee of the bandwidth curve divided by the perfect system's IPC for that application.

The results indicate that four of the nine applications considered in this study have a bandwidth knee of less than 6 bytes per cycle. Three of the remaining five have a bandwidth knee less than 100 bytes per cycle. Only tomcatv requires a large amount of bandwidth to be within 30% of its maximum performance of 238 IPC. The bimodal nature of application bandwidth requirements is probably due to the low bandwidth applications fitting in the 64K byte on-chip data cache.

Note that high bandwidth requirements are also related to the IPC of the program (as indicated by the bandwidth knees represented in bytes per instruction). These results show that the performance of systems targeting a moderately high IPC in the range of 100-150 will not be highly bandwidth restricted. A bandwidth of 100 bytes per cycle

[2] For the bandwidth curves, we use curve-fitting with equations modeling a high-pass filter ($y = \frac{ax}{bx+c}$).

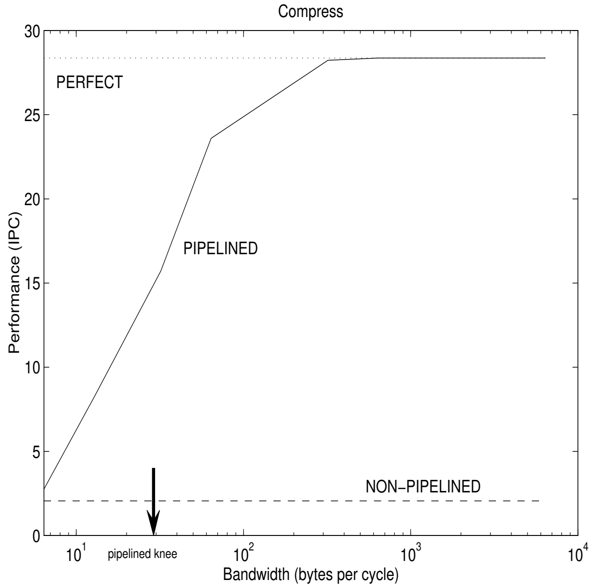

Figure 5: Impact of bandwidth variation on performance – Compress

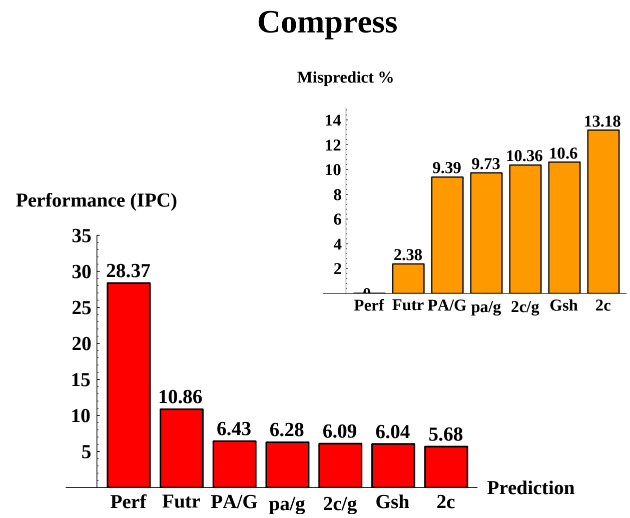

Figure 6: Impact of branch prediction on performance – Compress

implies an incoming data bus width of 100 bytes operating at the processor clock frequency. 100 incoming bytes implies 800 I/Os (not including address or ECC). This and a smaller outgoing data bus are almost small enough to fit in currently announced packages of 1500 I/Os[25]. However in a decade, packages with even more I/Os should be available.

These results are consistent with (but not implied) by the results of Burger, Goodman, and Kägi (BGK). The ratio of their second-level cache latency to the processor cycle time is similar to ours (20 vs. 16). Although BGK do not list the IPC achieved in their simulations, their results are for a 4-way issue processor, so the IPCs are probably between 1 and 2. We have shown that IPCs 50 to 100 times larger than this can be achieved by reordering instructions across a much bigger window (approximately 100X larger), by providing an interface that runs at the processor clock frequency instead of a multiple like 1/4, by increasing the memory bus width by an order of magnitude, and by fully saturating the memory interface.

4.2 Impact of Branches on Performance

Figure 6 shows the impact of branch misprediction on the performance of Compress. We model the effect of the seven branch prediction strategies described in Section 3.3 assuming a perfect memory system. The two graphs in the figure plot the performance (measured by the IPC) and the prediction accuracy obtained with the seven branch prediction models. This shows that the misprediction rate for compress has only been reduced by 29% when moving from 1981 branch predictor algorithms [26] to 1996 branch predictor algorithms. This is an annual rate of misprediction reduction of 2.2%. Improving branch predictor algorithms from 1981 to 1996 results in an IPC improvement of 13%, for an annual rate of IPC improvement of 0.83%. Although the remaining unpredictability of the branches in compress may be due to entropy in the input data set, we assume that some future predictor will reduce the misprediction rate to 25% of the PA/G predictor. Even so, this yields performance almost three times lower than that of a perfect branch predictor. Figure 7 presents the corresponding graphs for the other benchmarks.

Figure 8 summarizes the branch information for all the applications. The second column gives the (dynamic) percentage of branches in the application. Then for each branch prediction method, the left subcolumn gives the misprediction rate and the right subcolumn gives the ratio of the IPC using an ideal perfect branch predictor over the IPC when using the predictor listed at the top of the column. The loss in performance due to branches is proportional to the percentage of branches in the program, the misprediction rate and the impact of the mispredictions (determined primarily by the ILP in the program). The results show that there is a significant loss in performance due to branch mispredictions. This loss in performance increases with higher IPCs. Note that only hydro2d achieves more than a factor of two improvement in IPC moving from the two-bit counters to PA/G. For all but two benchmarks there is also a big difference between the performance with the most aggressive current branch predictors and the futuristic predictor models. Significantly, in many cases, even the aggressive futuristic model has a significant performance loss compared to the perfect branch predictor model.

4.3 Relative Impact of Memory and Branch Walls

The previous sections discussed the impact of memory latency and bandwidth, and control dependence in isolation on the performance of the processor. This section compares the impact of memory and branch wall effects relative to one another.

Figure 9 shows the effect of varying the memory latency and the branch prediction model on the performance. These graphs plot the performance (measured by the IPC) on the Z axis, the memory miss latency on the Y axis, and the branch prediction strategy on the X axis. Figure 10 describes the various points chosen. The latency and branch prediction

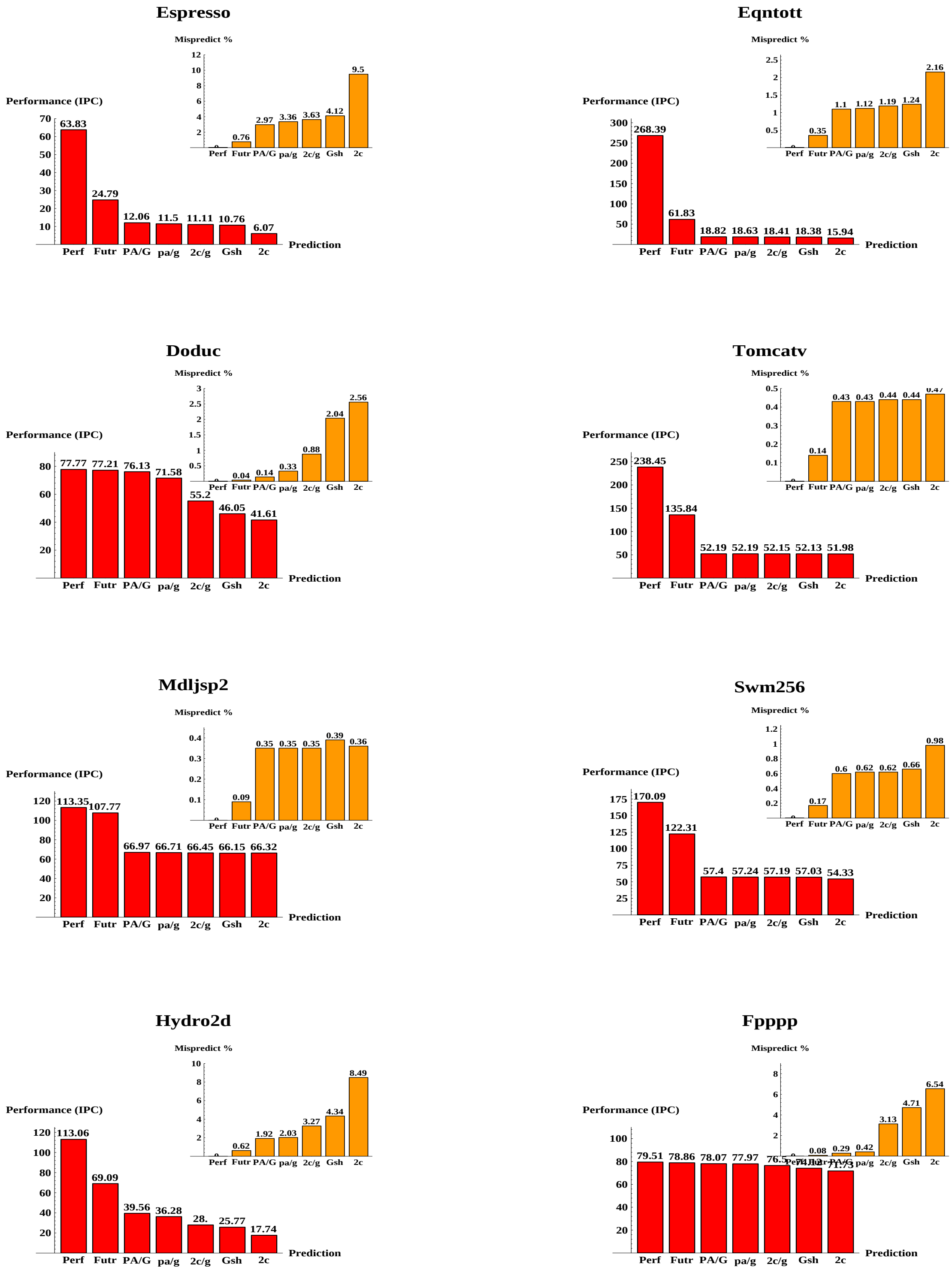

Figure 7: Impact of branch prediction on performance

Code	branch %	Future miss pred %	Future Ideal vs. IPC	PA/G miss pred %	PA/G Ideal vs. IPC	pa/g miss pred %	pa/g Ideal vs. IPC	2c/g miss pred %	2c/g Ideal vs. IPC	Gsh miss pred %	Gsh Ideal vs. IPC	2c miss pred %	2c Ideal vs. IPC
Compress	13.34	2.38	2.61	9.39	4.41	9.73	4.52	10.36	4.66	10.60	4.70	13.18	4.99
Espresso	17.81	0.76	2.57	2.97	5.29	3.36	5.29	3.63	5.55	4.12	5.75	9.50	10.52
Eqntott	10.91	0.35	4.34	1.10	14.26	1.12	14.41	1.19	14.58	1.24	14.60	2.16	16.84
Doduc	6.07	0.04	1.01	0.14	1.02	0.33	1.09	0.88	1.41	2.04	1.69	2.56	1.87
Tomcatv	3.32	0.14	1.76	0.43	4.57	0.43	4.57	0.44	4.57	0.44	4.57	0.47	4.58
Mdljsp2	8.59	0.09	1.05	0.35	1.69	0.35	1.70	0.39	1.71	0.36	1.71	0.26	1.71
Swm256	1.85	0.17	1.39	0.60	2.96	0.62	2.97	0.62	2.97	0.66	2.98	0.98	3.13
Hydro2d	7.82	0.62	1.64	1.92	2.86	2.03	3.12	3.27	4.04	4.34	4.39	8.49	6.37
Fpppp	1.21	0.08	1.00	0.29	1.02	0.42	1.02	3.13	1.04	4.71	1.07	6.54	1.10

Figure 8: Summary of branch effects on performance

points were chosen to represent current trends in memory latency and branch prediction strategies.

These graphs show the relative impact of varying the memory latencies and branch prediction strategies on the performance of the system. The results demonstrate that, for most of the applications, the branch wall is a more significant problem than the memory latency wall; i.e., the loss in performance due to less-capable branch prediction is much higher than the loss in performance due to increases in memory miss latency. Thus, as processors get more and more aggressive in the future and exploit increased amounts of instruction-level parallelism, the loss in expected performance gains will be more due to the inadequacy of such processor designs to account for control dependences than due to the increased memory miss latency in such processors.

Figure 11 shows the geometric mean of the results on varying both latency and bandwidth with four branch prediction strategies (chosen to demonstrate a fair variation of the misprediction rates). Due to constraints of simulation time, we studied hydro2d, tomcatv, compress and doduc as representative applications for this analysis. The graphs plot the performance (measured by the IPC) on the Z axis, the bandwidth on the Y axis, and the latency on the X axis. Figure 10 shows the points chosen on the X and Y axes. These results demonstrate that the three effects – increasing memory miss latency, decreasing available bandwidth to the system and varying branch prediction accuracy – have different relative impacts on the performance. The inadequacy to account for control dependences leads to the most significant loss of performance, while the ability to overlap memory operations with other instructions in the program leads to the maximum tolerance to increased memory latency effects. Furthermore, these results also demonstrate tradeoffs in the design space available to future-generation processors. A targeted IPC may be achieved by increasing the aggressiveness of either the branch prediction accuracy, the bandwidth, or the latency of the proposed system.

5 Conclusions

In this paper we have investigated the relative importance of memory latency, memory bandwidth, and branch predictability in determining limits of processor performance. The studies we performed are not specific to any existing processor or set of parameters. Although future processors are likely to have many megabytes of on-chip cache, we assumed 64KB 4-way set-associative on-chip caches when using the SPEC92 benchmarks as an approximation of results with very large on-chip caches and very large future benchmarks.

We found that, for our applications, memory latency did not become a limiting factor to performance in systems with very heavily pipelined memory systems until the latency was on the order of 100-200 cycles. For many applications this allowed the IPC to be close to 100. These numbers were only achievable with perfect branch prediction. For future processors with sub-nanosecond cycle times, these latency requirements should be achievable with hard work and off-chip caching (no off-chip caches levels were used in our simulations).

Limited memory bandwidth reduced the ultimate limit performance by 30% when the available bandwidth was reduced below 100 bytes per cycle for many benchmarks, although half the benchmarks performed well down to less than 6 bytes per cycle. Bandwidths of 100 bytes per cycle could be provided by running a 100 byte unidirectional bus of incoming data at the future processor clock frequency, with a smaller bus for outgoing data. Based on current research in packaging and interface technology we expect that to be obtainable with good engineering.

By far, the biggest limit to performance was branch unpredictability. The use of one of the best existing branch predictors provided with very large table sizes resulted in performance three to five times less than that achievable with perfect branch prediction for many benchmarks. Many benchmarks even had significantly lower performance than the perfect case when an aggressive futuristic branch predictor (a mix of 75% perfect and 25% existing branch predictor) was simulated. Other recent work on branch predictor performance for operating systems[27, 28, 11] and other workloads suggests that branch prediction accuracies are likely to further decrease in such cases; our results on the importance of the branch wall therefore tend to err on the optimistic side (by underestimating the importance of the branch wall).

This work suggests two directions for future research. First, improved branch prediction techniques appear to be crucial for improving future uniprocessor performance. Our experiments on the remaining unpredicted branches show that much of the performance loss is due to branches that are reasonably well predicted, but have isolated rare mispredicts. Additionally, the performance impact of branch mispredicts can vary widely; a small portion of branch mispredicts can contribute to a large portion of the loss in exe-

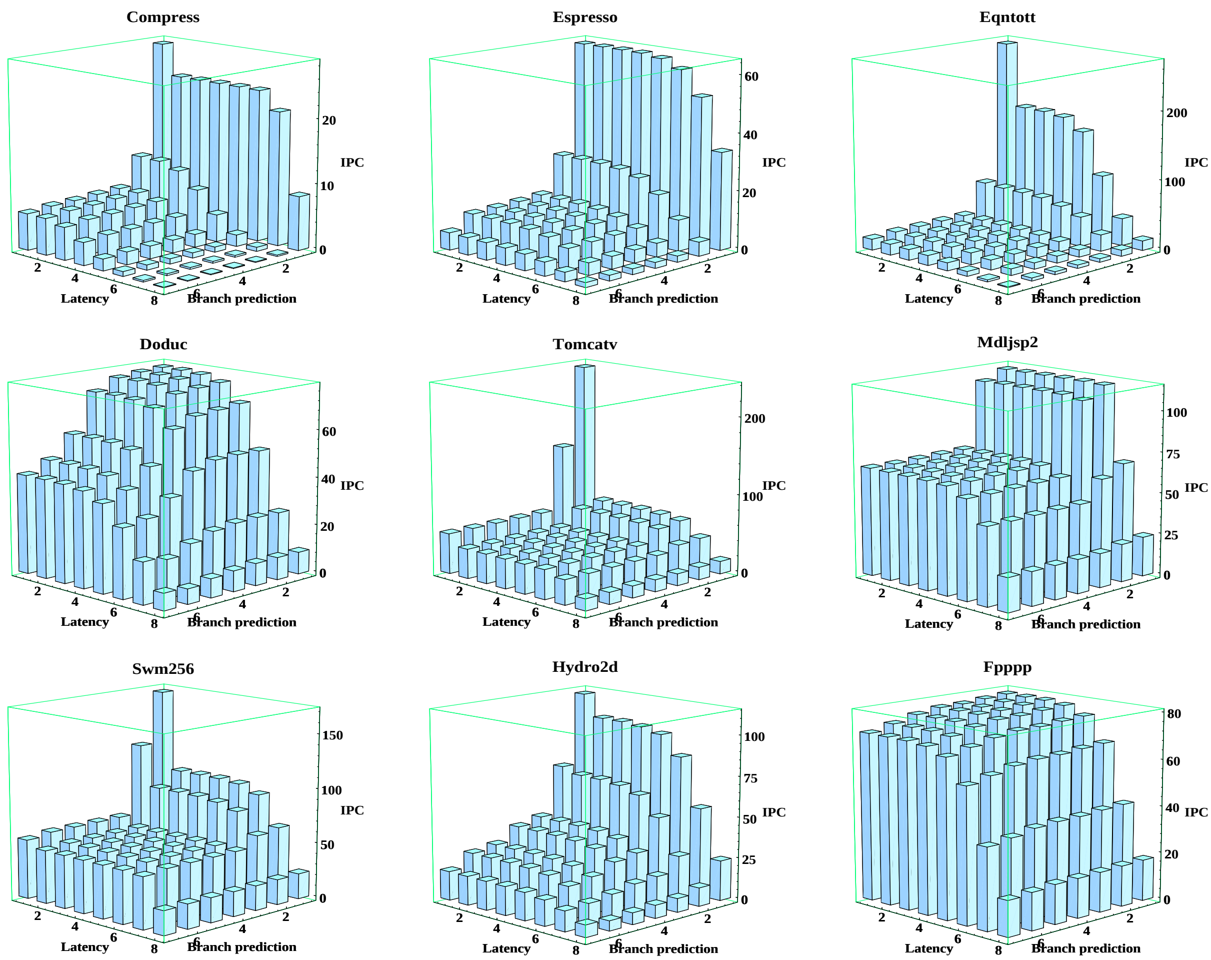

Figure 9: Impact of latency and branch prediction on performance

cution time due to branches. Future branch prediction techniques may exploit these characteristics to yield higher performance (possibly without necessarily achieving lower mispredict rates). Second, even with very low mispredict rates, if sufficient numbers of branch predictions are outstanding the odds of fetching useful instructions in order will become low. Hence techniques which exploit larger-grain flow predictability at the procedure or fine-grain task level are very attractive alternatives for research. These techniques include multiscalar[29], multithreading[30], or parallel processors on a single chip[31].

6 Acknowledgments

This work was supported by an internship at DEC Western Research Laboratory. Thanks to all the staff at DEC WRL for putting up with our simulations. Special thanks to Alan Eustace, Silvio Turrini, Annie Warren, and Jason Wold for their help in setting up the infrastructure for this work. Thanks to Sarita Adve, Brad Calder, and Vijay Pai for very helpful comments on earlier drafts of this paper.

References

[1] Amitabh Srivastava and Alan Eustace. ATOM: A System for Building Customized Program Analysis Tools. *Proceedings of the ACM SIGPLAN '94 Conference on Programming Languages*, March 1994.

[2] David W. Wall. Limits of instruction-level parallelism. In *WRL Research Report 93/6*, November 1993.

[3] M Butler, T.-Y. Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebanow. Single instruction parallelism is greater than two. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 276–286, May 1991.

[4] Kenneth Yeager, et. al. R10000 Superscalar Microprocessor. *In the Proceedings of Hot Chips VII*, August 1995. See also http://www.mips.com/HTMLs/T5_B.html.

[5] Intel Corporation. *Pentium (r) Pro Family Developer's Manual.*

[6] Linley Gwennap. PA-8000 Combines Complexity and Speed. *Microprocessor Report*, 8(15):1–9, November 14 1994.

[7] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. In *ACM Computer Architecture News. Vol. 23, No. 4*, September 1995.

[8] Eric E. Johnson. Graffiti in "the memory wall". In *ACM Computer Architecture News*, 1995.

[9] Po-Yung Chang, Eric Hao and Yale N. Patt. Alternative Implementations of Hybrid Branch Predictors. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995.

[10] Scott McFarling. Combining branch predictors. In *WRL Technical Note TN-36*, 1993.

	0	1	2	3	4	5	6	7
Latency (cycles)	perf	1	3	10	32	100	320	1000
Bandwidth (bytes per cycle)	6	19	64	192	640	2048	perf	–
Branch prediction	Perf	Futr	PA/G	pa/g	2c/g	gsh	2c	–

Figure 10: Legend for graphs in Figures 9 and 11

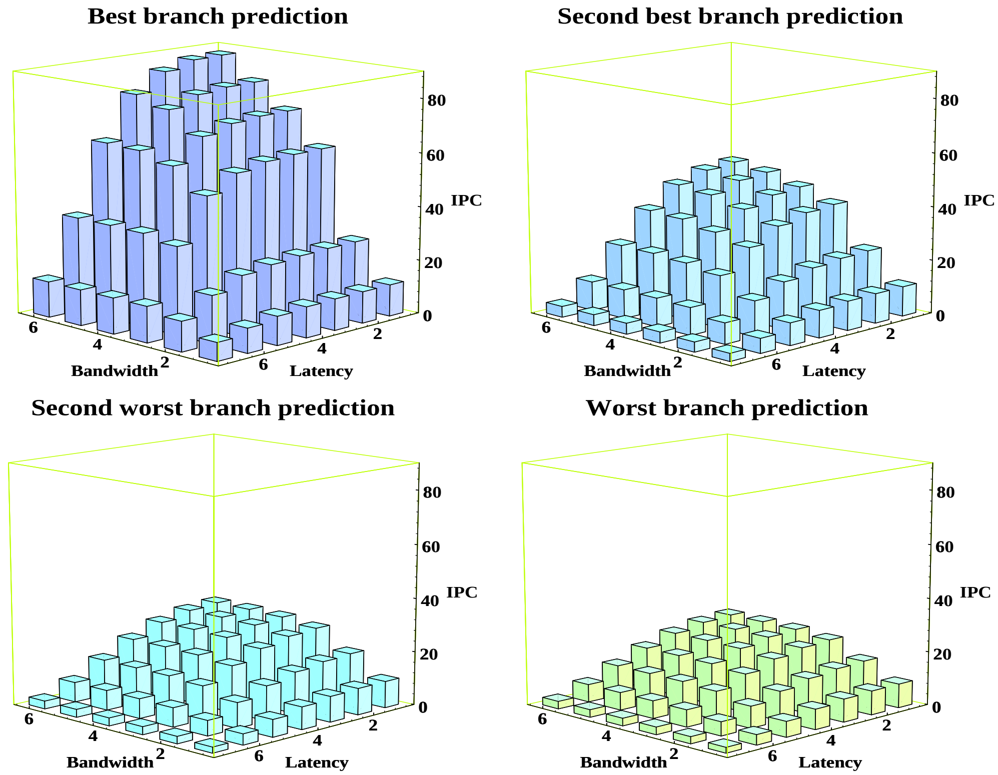

Figure 11: Relative impact of latency, bandwidth and branch prediction

[11] Marius Evers, Po-Yung Chang, and Yale N. Patt. Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 3–11, May 1996.

[12] T.-Y.Yeh and Y.N.Patt. Alternative Implementations of Two-Level Adaptive Branch Prediction. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[13] Todd Austin and Guri Sohi. Dynamic Dependency Analysis of Ordinary Programs. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[14] John L. Hennessy and David A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, second edition, 1992.

[15] J. E. Thornton. *Design of a Computer - The Control Data 6600*. Scott, Foresman, 1970.

[16] L. J. Boland, G. D. Granito, A. U. Marcotte, B. U. Messina, and J. W.Smith. The IBM System/360 Model 91: Storage System. In *IBM Journal of Research and Development*, pages 54-68, January 1967.

[17] Keith I. Farkas and Norman P. Jouppi. Complexity/performance tTadeoffs with Non-blocking Loads. In *Proc. of the 21st Annual International Symposium on Computer Architecture*, pages 211–222, June 1994.

[18] A. Saulsbury and F. Pong and A. Nowatzk. Missing the Memory Wall: The Case for Processor Memory Integration. In *Proc. of the 23rd Annual International Symposium on Computer Architecture*, pages 90–101, May 1996.

[19] Doug Burger, James Goodman, and Alain Kagi. Memory Bandwidth Limitations fo Future Microprocessors. In *Proc. of the 23rd Annual International Symposium on Computer Architecture*, pages 78-89, May 1996.

[20] H. M. Rein and M. Moller. Design Considerations for Very-High-Speed Si-Bipolar IC's Operating up to 50Gb/s. In the *IEEE Journal of Solid-State Circuits*, pages 1076–1090, August 1996.

[21] William J. Dally and John Poulton. Transmitter Equalization for 4Gb/s Signalling. In the proceedings of *Hot Interconnects IV*, August 1996.

[22] Hiroyuki Sakai, et. al. A Millimeter-Wave Flip-Chip IC using Micro-Bump Bonding Technology In the proceedings of the

International Solid-State Circuits Conference, pages 408-409, February 1996.

[23] Kevin Theobald, Guang Gao, and Laurie Hendren. On the Limits of Program Parallelism and its Smoothability. In *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pages 10–19, December 1992.

[24] Mark Horowitz, et. al. PLL Design for a 500 MB/s Interface. In the proceedings of the *International Solid-State Circuits Conference*, pages 160–161, February 1993.

[25] LSI Logic Corporation. LSI Logic Announces Flip Chip Technology. *PR Newswire Press Release*, Septtember 30, 1996.

[26] James E. Smith. A Study of Branch Prediction Strategies. In *Proceedings of the 8th Annual International Symposium on Computer Architecture*, pages 135–148, 1981.

[27] Nicolas Gloy, Cliff Young, J. Bradley Chen, and Mike Smith. An Analysis of Dynamic Branch Prediction Schemes on System Workloads. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 12–21, May 1996.

[28] Stuart Sechrest, Chih-Chieh Lee, and Trevor Mudge. Correlation and Aliasing in Dynamic Branch Predictors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 22–32, May 1996.

[29] Gurindar Sohi, Scott Breach, and T.N. Vijaykumar. Multiscalar Processors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 414–425, June 1995.

[30] Dean Tullsen, Susan Eggers, Joel Emer, Henry Levy, Jack Lo, and Rebecca Stamm. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 191–202, May 1996.

[31] Basem Nayfeh, Lance Hammond, and Kunle Olukotun. Evaluation of Design Alternatives for a Multiprocessor Microprocessor. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 67–77, May 1996.

[32] David Kroft. Lockup-free Instruction Fetch/Prefetch Cache Organization. In *Proceedings of the 8th International Symposium on Computer Architecture*, pages 81–87, May 1981.

[33] Scott. A. Mahlke, Richard E. Hank, Roger A. Bringmann, John C. Gyllenhaal, David M. Gallagher, and Wen-mei W. Hwu. Characterizing the Impact of Predicated Execution on Branch Prediction. In *Proceedings of the 27th International Symposium on Microarchitecture*, pages 217-227, Dec 1994.