Impact of Technology Scaling on Instruction Execution Throughput

M.S.Hrishikesh* Doug Burger Stephen W. Keckler

Computer Architecture and Technology Laboratory
Department of Computer Sciences
*Department of Electrical and Computer Engineering

Tech Report TR2000-06

The University of Texas at Austin
cart@cs.utexas.edu — www.cs.utexas.edu/users/cart

Abstract

Future technologies will enable faster and more numerous transistors on-chip. However, poor wire scaling as semiconductor devices shrink will increase on-chip communication delays. In this report, we explore two methods in which processor pipelines of the future may be designed – deep pipelines with large structures capacities and short pipelines with small structure capacities. We perform our study for fifteen different clocks and across seven technologies. We show that for both design methods optimal performance is obtained when the amount of useful logic per pipeline stage corresponds to a delay of 6 fan-out-of-four (FO4). We present an extention of the optimal pipeline study in [1]. We also study the effect of the size and latency of critical microarchitectural structures on performance, measured in instructions per cycle (IPC). We quantify the effect of the latency of structures and show that the access penalties of the level-1 cache, the branch predictor and the instruction window have the largest effect on IPC. In addition, we also quantify the effect of functional unit execution latencies on IPC.

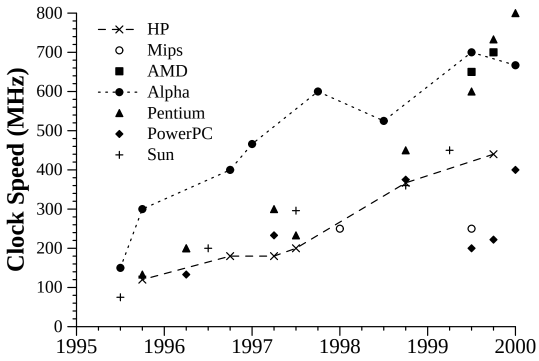


Figure 1: Processor clock rates 1995-2000

1 Introduction

Microprocessors have been improving in performance approximately at the rate of 40-50% per year over the last decade. This improvement in performance has been largely due to improvements in clock frequencies and improvements in *instructions per cycle* (IPC). While processor designs have benefited from improvements in both clock and IPC, some designers have emphasized improvements in clock frequency, as can be seen from Figure 1. Others have focused their efforts on improving instruction throughput, as shown in Figure 2. Both design approaches have yielded similar improvements in performance across processor generations.

Improvements in IPC can be largely attributed to innovative architectural designs and increased capacity of microarchitectural components. Smaller technologies have enabled fabrication of more transistors on-chip. This ability has in turn allowed architects to devote more transistors to microarchitectural structures designed to extract program parallelism. The other parameter contributing to processor performance is clock frequency. Improvements in clock frequency arise in part from reducing the amount of work done each cycle and also from changes in process technology. The shorter gate lengths of smaller technologies improve switching speed of transistors and thereby help achieve higher clock frequencies. Another effect of fabricating devices in smaller technologies is the increasing dominance of wire delays. As technology scales the resistance of interconnects increases while their capacitance remains more or less constant. The effect of technology scaling on interconnects is to increase the RC propagation delay. In short, as technology scales wire propagation delay increases relative to transistor switching delay [2]. While wires are predicted to get slower in future technologies, the on-chip clock frequency is projected to rise super-linearly [3]. This combination of slow wires and rapidly increasing clock frequencies will reduce the area on the chip that can be reached in a single clock cycle [2].

Improving IPC requires microarchitectural structures to store runtime state information. However, poor scaling of wires and increasing clock frequencies will increase number of cycles to access these structures. A high performance design will have to balance the access latency of key microarchitectural structures and the clock frequency. This report examines two important issues that will help demonstrate the tradeoffs between clock frequency and pipeline depth (structure size) – (a) the scalability of microarchitectural structures and (b) the impact of structure latency and capacity on performance.

The remainder of this report is organized in the following fashion. In section 2 we describe the methodology used to scale microarchitectural structures and our assumptions regarding how

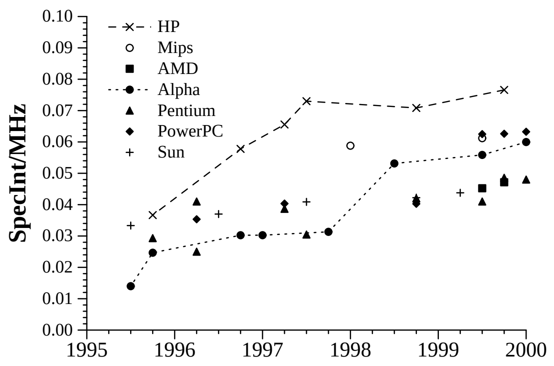


Figure 2: Normalized processor performance (SpecInt/MHz)

Integer	Floating Point
164.gzip	171.swim
175.vpr	172.mgrid
176.gcc	173.applu
181.mcf	177.mesa
197.parser	178.galgel
252.eon	179.art
253.perlbmk	183.equake
256.bzip2	188.ammp
300.twolf	189.lucas

Table 1: SPEC 2000 Benchmarks

the structures are pipelined. Section 3 presents a study of the sensitivity of IPC with respect to individual structure latency and capacity. Sections 4 and 5 examine two design methodologies future designers could adopt: (a) pipeline scaling, wherein the capacity of the structures is kept constant and their access latencies are scaled per technology and clock frequency and (b) capacity scaling, wherein the access latency of structures is held constant and their capacities are scaled with technology. In section 6 we select the best possible design configurations from the capacity and pipeline scaling methods, trading off structure capacity against its access latency and section 7 summarizes the results and presents the conclusions.

2 Methodology

2.1 Simulation Methodology

To study the effect of technology scaling on microarchitectural pipeline, we chose to scale the Alpha 21264 core since it is representative of a modern high performance design. The Alpha 21264 processor is implemented as a seven stage pipeline capable of issuing up to four instructions every cycle. The memory organization for this processor consists of level-1 instruction and data caches and a unified level-2 cache. The instruction cache is direct mapped and accessible in a single cycle. The data cache is 2-way set associative and has an access latency of 3 cycles. Both level-1 caches are 64KB in size and the level-2 cache is 2MB large.

For our experimental simulations we used a simulator developed by Desikan, et al. [4]. This

	Capacity (bits)	# entries	Bits/entry	Ports	Latency
L1 I-Cache	512K	1K	512	1	1
L1 D-Cache	512K	1K	512	2	3
L2 Cache	16M	16K	1024	2	6
I-TLB	6K	128	44	1	1
D-TLB	6K	128	44	2	1
Local pred.	3K	1K	3	1	0.5
Local history	10K	1K	10	1	0.5
Global pred.	8K	4K	2	1	1
Choice pred.	8K	4K	2	1	1
Reorder buffer	10K	80	128	8	1
Rename Table	80	80	1	12	1
Issue window	800/320	20	40	8/6	1
Register File	5K	80	64	10	1

Table 2: Capacities of structures used in delay calculations

simulator models the Alpha 21264 micro-architecture to within 18% error. We simulated benchmarks from the SPEC 2000 suite for up to 500 million instructions. Table 1 lists the benchmarks used for our simulations. In the rest of this section we describe the methodology used to scale microarchitectural components.

2.2 Modeling Micro-architectural Structures

To estimate the access latency of microarchitectural structures we model them using Cacti [5]. Cacti is an analytical tool that models access and cycle times of caches and cache-like components. Given a specific configuration – the size of the cache, cache block size, associativity, the number of address and output bits and the number of ports – the tool computes the access delay of the cache. Cacti models different parts of cache circuitry by decomposing them into simple RC equations. The total delay of the circuit is the sum of the delays of each individual component. The original version of Cacti models a cache at 0.25 micron technology, Agarwal, et al. [6] extended the tool to model caches at smaller technologies.

We model the following microarchitectural structures using Cacti: instruction and data caches, register file, branch predictor, translation look-aside buffer (TLB), register rename table, instruction issue window and re-order buffer (ROB). The structures were configured as shown in Table 2.

The instruction and data TLBs are modeled as content-accessible memories (CAM) with 56 bits wide tags - 48 bits of virtual address and 8 bits to represent the address space number. Each entry of the TLB is 44 bits wide to represent the physical address. The branch predictor in the Alpha 21264 [7] consists of a local predictor, a global predictor and a choice predictor. The local predictor is a 2-level predictor that uses the contents of a local history table of 1K entries index into a local predictor table of 1K entries. The local history table itself is indexed using 10 bits of the branch instruction PC. The global predictor is a table of 4K entries that is indexed by a global history of the 12 most recent branches. The choice predictor selects between the predictions made by the local and global predictors. All three predictors can be accessed in parallel, therefore the latency of the branch predictor is determined by the slowest of the three predictors. We found that the local predictor latency was always the component that determined the branch predictor latency across all technologies and clock frequencies.

The instruction issue window (IW) is modeled as a 8-bit wide CAM and a 40-bit wide direct mapped structure. The IW has 8 ports which are used to write 4 new instructions every cycle and to issue 4 instructions. The IW in the Alpha 21264 is a self collapsing window - it collapses empty slots at the beginning of the cycle to make room for new incoming instructions. Our model of the instruction window does not account for this functionality.

The rename table is organized as a CAM [8]. The CAM has one entry for each physical register. Each entry has two fields - a logical register designator and a valid bit. The valid bit is set if the current mapping is valid. Renaming is done by matching the logical register designator field. The rename table has 12 ports to enable renaming of up to four instructions in a cycle.

The access latency obtained from Cacti is used to scale the microarchitectural structures for the pipeline and capacity scaling experiments. We also assume that all of the structures can be perfectly pipelined.

3 Sensitivity Analysis

The size and access latency of microarchitectural structures have a direct impact on the performance of the processor. Large microarchitectural components result in better performance of the structures – in the case of branch predictors it results in better accuracy while in the case of caches and TLBs it results in better hit rates. However, increasing structure capacity also increases access latency of the structures. This increase in access latency could degrade overall performance.

In this section we examine the impact of structure size and latency on IPC. We performed two sets of experiments. In the first set of experiments the structure sizes are configured to be the same as the base case while the access latencies of these structures is varied. The second set of experiments vary the capacity of structures while holding their access latencies to be the same as the base case.

In each of the experiments either the access latency or the capacity of one structure is varied while all other structures are configured as in the base case. For our experiments we simulated the SPEC 2000 benchmark suite, for all experiments we report the harmonic mean of the IPC of the integer and floating-point benchmarks. All simulations were run for 500 million instructions. We used a modified version of the sim-alpha [4] simulator for our studies.

3.1 Effect of Structure Latency on IPC

In this section we study the impact of individual structure latencies on instruction throughput. For these simulations the capacity of the structures are the same as the base case and the latency is varied.

3.1.1 Instruction Cache

The instruction cache latency has a large impact on IPC. On every I-cache access the Alpha 21264 processor fetches four instructions, corresponding to its issue width. If the I-cache access time is increased to two cycles then a bubble is introduced in the pipeline on every fetch. This clearly hurts performance and is not a realistic design option. Designers can tolerate some I-cache latency by fetching a larger number of instructions on every fetch and buffering those instructions in a small memory structure (i.e. caching the I-cache). However, an exhaustive study of such front end design techniques is outside the scope of this report. While we realize that multi cycle I-caches are not a valid design option we include the study of the sensitivity of IPC to I-cache latency for completeness.

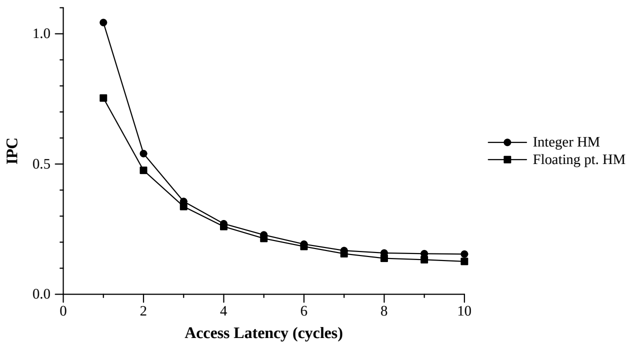


Figure 3: L1 Instruction Cache Latency Sensitivity

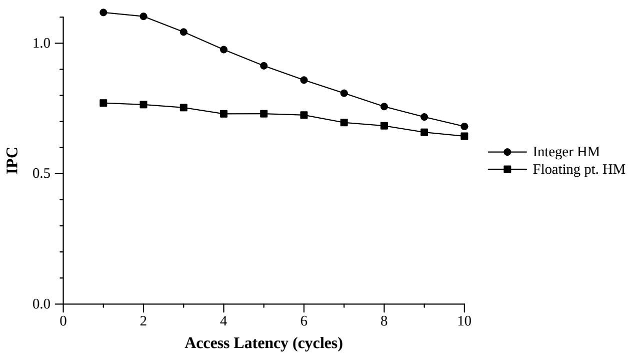


Figure 4: L1 Data Cache Latency Sensitivity

Figure 3 shows a plot of IPC versus instruction cache (IL1) latency. In this graph and all subsequent graphs in the sensitivity analysis section, we plot the harmonic mean of the integer benchmark IPCs as the integer curve, similarly the curve marked as floating-point is the harmonic mean of the floating-point benchmark IPCs. Both integer and floating-point benchmarks show the largest drop in IPC when IL1 latency increases from 1 to 2 cycles. The integer benchmarks drop in IPC by about 48% and the floating-point benchmarks show a drop of 37%. For both sets of benchmarks IL1 latencies beyond 6 cycles do not have a large impact on IPC.

3.1.2 Data Cache

Figure 4 shows a plot of IPC versus DL1 cache latency for integer and floating-point benchmarks. The integer benchmarks show an IPC drop of about 41% when the latency of the data cache is increased from 1 to 10 cycles while the floating-point benchmarks show a drop of around 25%. Both sets of benchmarks execute approximately the same number of load/store instructions. However, the floating-point benchmarks have greater instruction level parallelism and are therefore able to hide the memory access latencies by executing more instructions in parallel.

3.1.3 Translation Look-aside Buffer

In Figure 5 we display a plot of the IPC against increasing DTLB latency. The data cache we considered is a virtually indexed, physically tagged structure. Therefore the cache can be indexed and the appropriate data can be obtained while the TLB is being accessed to obtain the translation. In effect the latency of a hit in the DL1 is the maximum of the latencies to access the TLB and

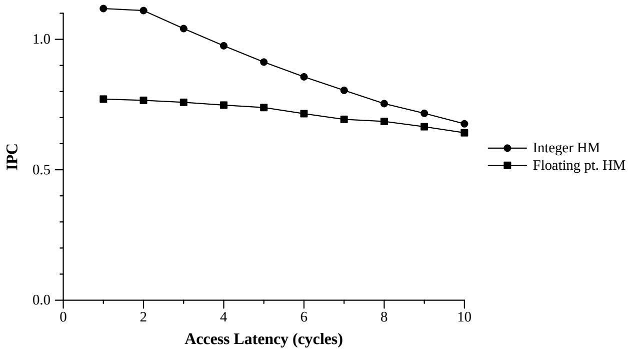


Figure 5: DTLB Latency Sensitivity

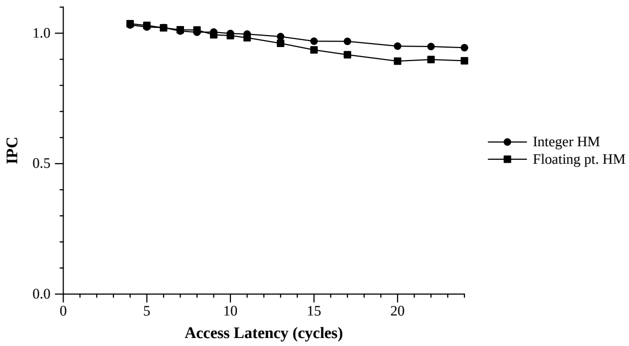


Figure 6: L2 Latency Sensitivity

the latency to access the data cache. Therefore it is not surprising that the sensitivity curve for the Data TLB looks exactly like the curve for the DL1 cache.

3.1.4 Level-2 Cache

Almost all the benchmarks have very low L1 cache miss rates and hardly access the L2 cache. Therefore, as can be seen from Figure 6 the L2 cache latency does not affect IPC significantly.

The level-2 cache is accessed only if an access to the level-1 cache results in a miss. The level-1 caches are 6KB large and make low miss rates, therefore the level-2 cache is accessed infrequently. As a result increasing the level-2 cache access latency does not have a significant impact on IPC. Figure 6 shows a plot of the harmonic mean of the IPCs of integer and floating point benchmarks against level-2 cache access latency. The integer benchmarks show a decrease in IPC of 8% when the L2 cache access latency is increased from 4 to 24 cycles. For the same increase in access latency the floating point benchmarks show a decrease in IPC of 13%.

3.1.5 Branch Predictor

Branch prediction occurs in the second stage of the Alpha 21264 pipeline. In the fetch stage of the pipeline a line predictor predicts the next cache line to be fetched. The line predictor provides a prediction in a single cycle. The branch predictor is accessed in the second stage of the pipeline. If the branch predictor agrees with the line predictor then no special action is taken. However, if the branch predictor disagrees with the line predictor then the instructions in the fetch stage are flushed, and in the next cycle the fetch engine begins to fetch instructions from the path predicted

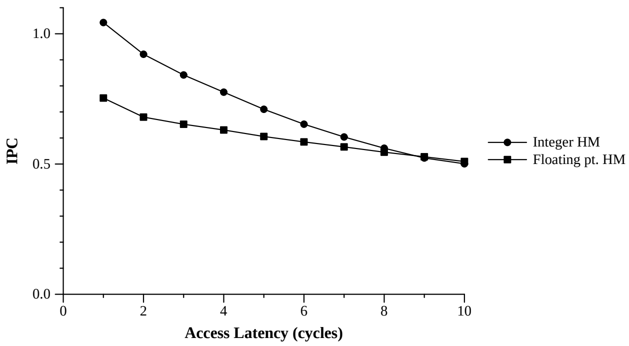
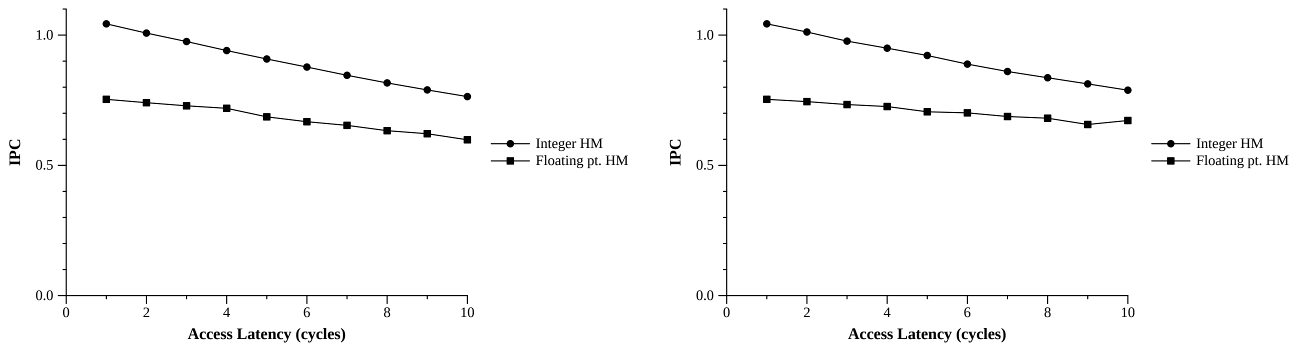


Figure 7: Branch Predictor Latency Sensitivity



A. Register File B. Rename Table

Figure 8: Register File and Rename Table Latency Sensitivities

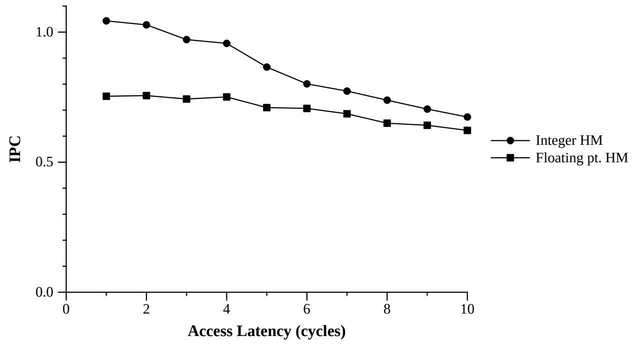
by the branch predictor. Whenever the branch predictor and the line predictor disagree a 1 cycle bubble is introduced in the pipeline.

For the sensitivity experiments, and in the subsequent pipeline and capacity scaling experiments, we do not scale the line predictor. When the branch predictor is configured to take multiple cycles to provide a prediction, the fetch engine continues to fetch instructions as long as the pipeline stages between the fetch stage and slot stage have sufficient buffer space to accommodate the new instructions.

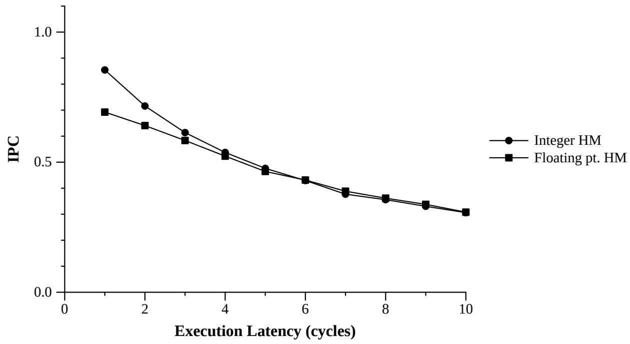
Figure 7 plots the harmonic mean of IPCs against access latency for integer and floating-point benchmarks. For the integer benchmarks IPC decreases by close to 55% when the latency of the branch predictor is increased from 1 to 10 cycles. For the floating-point benchmarks this decrease is noticeably smaller, for a similar increase in branch predictor latency the IPC decreases by only 6%. The effect of increasing the branch prediction latency is more pronounced on the integer benchmarks. This is to be expected because the integer SPEC benchmarks encounter a branch in about every 6 instructions as opposed to the floating-point benchmarks which encounter a branch every 55 instructions.

3.1.6 Register File

For the simulation of the register files we assume that register file access can be fully pipelined. In addition we also assume a full bypass from the execution units to the register read stages. Increasing the register read penalty extends the pipeline and adds to the branch mis-predict penalty. The effect of increasing register read latency from 1 to 10 cycles decreases IPC by about 26% for the integer



A. Re-order Buffer



B. Functional Unit

Figure 9: Latency Sensitivities

benchmarks and by about 20% for the floating-point benchmarks. The floating-point benchmarks have better branch prediction rates and therefore suffer the mis-prediction penalty less frequently than the integer benchmarks.

3.1.7 Rename Table

We model the rename table latency by increasing the number of stages between slot and map stage of the original Alpha 21264 pipeline. The only effect of increasing the rename table access latency has is increasing the branch mis-prediction penalty. Figure 8B shows that when the rename table latency is increased from 1 to 10 cycles the IPC of the integer benchmarks falls by around 25% while the IPC of the floating-point benchmarks falls by around 12%.

3.2 Re-order Buffer Latency

In Figure 9A we show the sensitivity of IPC to he re-order buffer (ROB) latency. Increasing the ROB latency increases the branch mis-prediction penalty, thereby reducing IPC. In addition, it also delays updates to the branch predictor which will lower branch prediction accuracy. The IPC of the integer benchmarks are more sensitive to ROB latency than the floating-point benchmarks due to the greater frequency of branches in integer code. As the ROB latency is increased from 1 to 10 cycles the integer benchmarks show a drop in IPC of about 35% while the floating-point benchmarks show a drop in IPC of about 18%.

3.2.1 Functional Unit

Functional units in many current processors are pipelined so that multiple instructions can be in different stages of execution in the same functional unit. However, if the execution penalty of functional units is more than one cycle dependent instructions cannot be issued back to back. Since the critical path of most programs is usually a stream of dependent instructions, increasing functional unit penalties will delay these paths and thereby reduce IPC. In addition, increasing the execution penalties of functional units increases the overall length of the pipeline and so increases branch mis-prediction penalty. These two factors combined make IPC more sensitive to functional unit latency that to either register read or rename table latencies.

In Figure 9B we show a plot of IPC against increasing functional unit latencies. The integer benchmarks show a drop of approximately 16% when the latency of functional units increase by 1

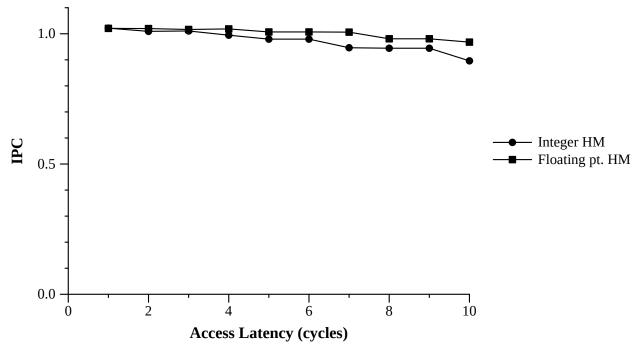


Figure 10: Instruction Issue Window Latency Sensitivity

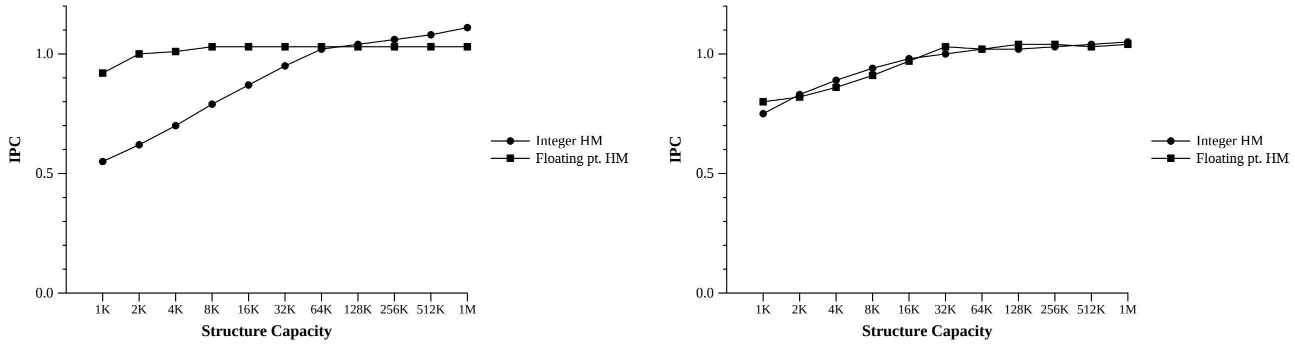
cycle over their baseline latencies. In general as the execution latencies of functional units increase the instruction throughput decreases. However, the rate at which the IPC decreases gradually reduces. For example, when latency increases from 3 to 4 cycles the corresponding reduction in IPC is only 13%. The floating point benchmarks also show a decrease in instruction throughput as the latencies of functional units are increased. However, the greater ILP of the floating point benchmarks enable them to tolerate the increase in functional unit latencies better. Consequently their IPCs decrease at a slower rate.

3.2.2 Instruction Window

The Alpha 21264 has two instruction issue windows – one for integer instructions and one for floating-point instructions. The function of the instruction window is to wake up instructions, thereby marking them as ready to be selected, and selecting from the instructions that are awake. Every cycle the tags of the operands being committed are compared with the tags of the source operands of every instruction in the instruction window. If the tags match the source operand for the corresponding instruction is marked as ready. An instruction is woken up if both its source operands are ready. Instructions to be issued are then selected from among those that are awake based on the availability of functional units. A more detailed description of wakeup-select logic is presented by Palacharla, et al. [8].

For our simulations we assumed the issue window is fully pipelined. For example, if an issue window was capable of holding 20 instructions with a 2 cycle access latency. A set of tags are compared with the first ten entries in the first cycle. In the second cycle the same set of tags are compared with the next ten entries, while a new set of tags are compared with the first ten entries. In every cycle potentially the entire instruction window could be woken up.

In Figure 10 we show a plot of the IPC and instruction issue window latency. The integer benchmarks show a 12% decrease in IPC when the instruction window latency is increased from 1 to 10 cycles while the floating-point benchmarks show a decrease of around 5%. The relatively small impact of instruction window access penalty on IPC is not surprising. The floating point benchmarks have very little dependency in their instruction stream and therefore remain unaffected by the increased penalties. For the integer benchmarks most of the dependent instructions are fairly close to the instructions that produce their source values. Also, the instruction window adjusts its contents at the beginning of every cycle so that the older instructions collect to one end of the window. This feature causes dependent instructions to eventually collect at the "bottom" of the window and so enables them to be woken up earlier.



A. Instruction

B. Data

Figure 11: L1 Cache Sensitivities

3.3 Effect of Structure Capacity on Performance

In this section we examine the impact of structure capacity on instruction throughput. For these simulations the access latencies of the structures are configured to be the same as the base case while their capacities are varied. For each experiment the capacity of one individual structure is varied, the capacities of other structures are held constant.

3.3.1 Instruction Cache

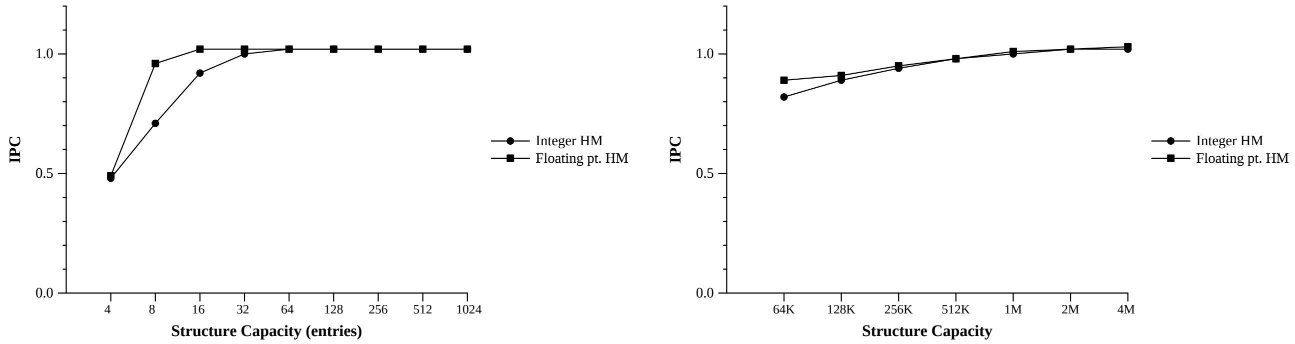
To measure the sensitivity of instruction throughput with respect to the instruction cache we conducted an experiment varying the I-cache size from 1M Byte to 1K Byte. As the I-cache size is decreased the miss rate of the cache increases. The decrease in IPC is a direct result of the increasing miss rate. As shown in Figure 11a for the integer benchmarks the IPC dropped by about 50% and for the floating point benchmarks the drop was just 11%. One of the reasons for the surprisingly small reductions in IPC is that we do not execute the programs to completion. However, since we do not scale the IL1 cache for the pipeline or capacity scaling experiments the conclusions drawn from those experiments are unaffected by this effect.

3.3.2 Data Cache

In Figure 11b we show the sensitivity of IPC to data cache size. When the data cache size is varied from 1MByte to 1KByte the IPC drops by about 25% for both the integer and floating-point benchmarks. For a cache size of just 1K the miss rate is about 45% for both sets of benchmarks. The miss rate progressively decreases as the cache size is increased. After about 128K of data cache memory we see no change in the miss rate when cache sizes are further increased. However, the IPC does not show any improvement for cache sizes beyond 64KB despite the decrease in miss rates. The latency of these extra misses is tolerated due to out of order execution and the inherent parallelism in the instruction stream.

3.3.3 Translation Look-aside Buffer

In the Alpha 21264 the L1 instruction cache is virtually indexed and virtually tagged and therefore does not require a translation to be accessed. The L1 data cache is virtually indexed and physically tagged and would require the translated address to perform tag compares. We varied the capacity of the data TLB from 4 entries to 1024 entries, Figure 12A shows a plot of IPC against the DTLB



A. DTLB B. L2 Cache

Figure 12: DTLB and L2 Capacity Sensitivities

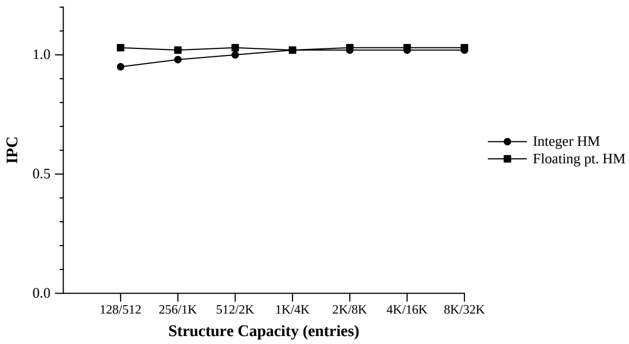


Figure 13: Branch Predictor Capacity Sensitivity

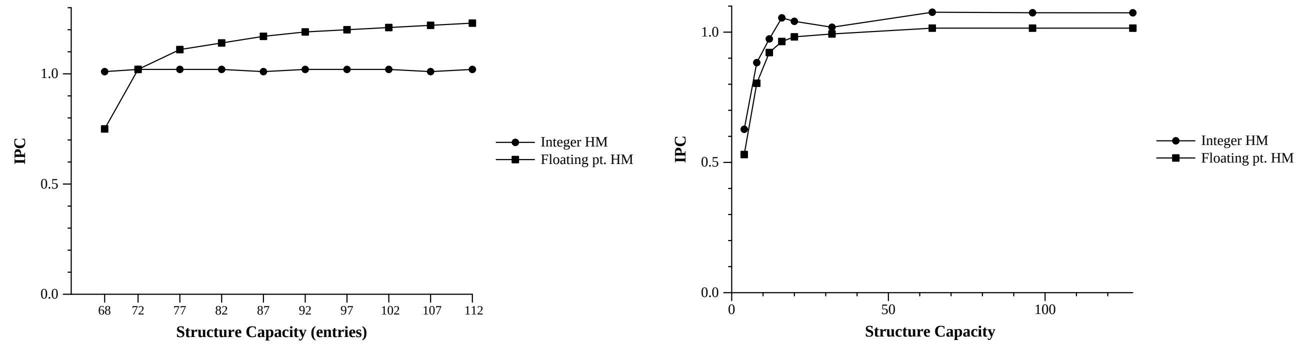
capacity. For both sets of benchmarks the IPC is extremely poor for small DTLB capacities between 4-16 entries due to a large number of DTLB misses. For DLTB capacities of 32-128 entries the IPC shows a marginal increase. Increasing the capacity beyond 128 entries shows no benefit in instruction throughput.

3.3.4 Level-2 Cache

In Figure 12B we plot the sensitivity of IPC to level-2 cache size. The level-2 cache size is varied from 64KB up to 4MB. For both sets of benchmarks IPC improves by 15% when the cache capacity is increased from 64KB to 2MB. The relatively small effect of the L2 cache capacity on IPC is explained by the fact that the L1 caches have high hit rates. The majority of memory requests are satisfied by the L1 caches. At about 2MB the working set of all the benchmarks is captured in the L2 cache and there is no improvement in IPC for larger cache sizes.

3.3.5 Branch Predictor

The Alpha 21264 branch predictor consists of a 1K entry local predictor and 4K entry global and choice predictors. In scaling the capacity of the branch predictor we kept the ratio between the sizes of the local and global/choice predictor the same. The size of the local predictor was varied between 128-8K entries and correspondingly the size of the global and choice predictors were varied between 512-32K entries. Figure 13 shows the sensitivity of IPC to branch predictor capacity. The integer benchmarks are more sensitive to the branch predictor capacity than the floating-point



A. Register File B. Issue Window

Figure 14: Register File and Instruction Issue Window Capacity Sensitivities

benchmarks because they have poorer branch behavior. The improvements in IPC saturate for a branch predictor capacity of - 1K local entries, 4K global entries.

3.3.6 Register File

The Alpha 21264 provides 32 integer architectural registers and the same number of floating-point architectural registers. In the map stage of the pipeline the operands of every instruction are renamed, this way conventional superscalar architectures remove false dependencies. Once the destination operand on an instruction is renamed all subsequent instructions that depend on the current instruction use the new renamed value.

One of the issues related to register renaming is when a physical register name is released. The physical register cannot be released when the corresponding instruction is committed as there could be dependent instructions in-flight. When an instruction is committed the current physical register is not released, instead the physical register that was previously mapped to the committing architectural register is released. For this mechanism to work the number of physical registers has to be at least twice as many as architectural registers.

Figure 14A shows the sensitivity of IPC to the number of registers. The integer and floating point register file capacity is increased from 68 to 112. The register file capacity has negligible impact on the instruction throughput of integer benchmarks. The floating-point benchmarks on the other hand show an improvement in IPC of close to 40% when the register file capacity is increased from 68 to 112.

3.3.7 Instruction Window

The sensitivity of IPC to the capacity of the instruction issue window is shown in Figure 14B. For this experiment the capacity of the re-order buffer was configured to match the capacity of the issue window. IPC increases as the capacity of the structure is increased and peaks at an issue window capacity of 64. Thereafter there is no significant change in the IPC for both the integer and floating point benchmarks.

4 Pipeline Scaling

Performance is a function of two parameters - instruction throughput and clock frequency. Improvements in microprocessor performance in the past has come about by improvements to both

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	16	11	89	10	7	9	9	6
3	11	7	60	7	5	6	6	4
4	9	6	45	5	4	5	5	3
5.5	7	4	33	4	3	4	4	2
6	6	4	30	4	3	3	3	2
7	6	3	26	3	2	3	3	2
8	5	3	23	3	2	3	3	2
9	5	3	20	3	2	2	2	2
10	4	3	18	2	2	2	2	2
11	4	2	17	2	2	2	2	1
12	4	2	15	2	2	2	2	1
13	4	2	14	2	1	2	2	1
14	4	2	13	2	1	2	2	1
15	3	2	12	2	1	2	2	1
16	3	2	12	2	1	2	2	1

Table 3: Pipeline scaling structure latencies at 100nm technology

these parameters. One method designers use to obtain higher clock frequencies is by reducing the amount of work done in every stage of the pipeline and thereby increasing the depth of the pipeline. However, this improvement in clock frequency comes at a price. Longer pipelines increase the penalty of mis-predicted branches and therefore tend to reduce instruction throughput (IPC). Furthermore, in pipelined processors the entire clock period cannot be not used to perform useful work. A fraction of the period is required by the latches between the pipeline stages to sample and hold data values. This time is termed as latch overhead. As the amount of work at every pipeline stage is reduced the overhead becomes an increasing fraction of the clock period. Since a smaller fraction of the clock period is used to perform useful work in deep pipelines each operation will take longer time to complete.

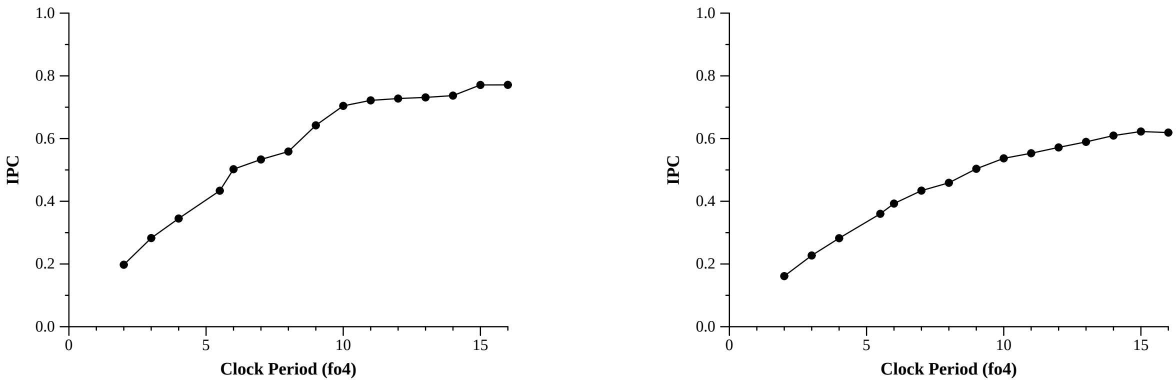
Deeper pipelines result in improvements to clock frequency but also degrade IPC due to greater mis-predict penalties, greater structure access penalties and increased latch overhead. In this section we study the tradeoff between clock frequency and IPC for deeply pipelined processors. For this purpose we consider an Alpha 21264 like processor core. The capacities of the microarchitectural structures such as caches, TLBs, branch predictors etc. are configured to be the same as in the Alpha 21264. The structure access penalties are varied corresponding to the clock frequency. We consider clock periods ranging from 2 fanout-of-four to 16 fanout-of-four (FO4). In addition, to examine how this tradeoff scales in the future we extend our experiments across seven technologies – 250nm, 180nm, 130nm, 100nm, 70nm, 50nm and 35nm.

As explained in section 2.2 we use Cacti to model the microarchitectural components at the different technologies. Table 3 shows microarchitectural structures we considered and their access penalties at 100nm technology for each of the clock frequencies we studied. In addition to microarchitectural components the latencies of functional units were also scaled. Table 4 shows the latencies of the functional units at 100nm technology in terms of clock cycles. In both tables the column designated as Clock represents the amount of useful work per pipeline stage. The access penalties of structures and functional units at other technologies are listed in Appendix A.

Figure 15 shows a plot of the harmonic mean of IPC values, measured across the SPEC 2000 integer and floating point benchmarks, against clock period. The x-axis plots the clock period (not counting latch overhead) increasing from 2 FO4 to 16 FO4. In general, as the clock period increases the IPCs of both integer and floating point benchmarks increase up to a point. This improvement in

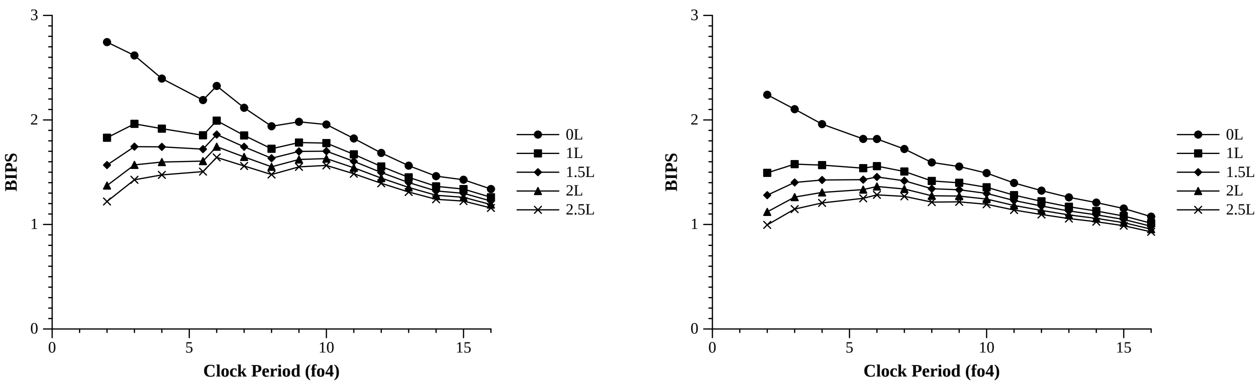
Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 4: Pipeline scaling operation latencies at 100nm technology



Integer Floating pt.

Figure 15: 100nm Technology Pipeline Scaling IPC Trend



Integer Floating pt.

Figure 16: 100nm Technology Pipeline Scaling Performance Trend

IPC is due to reduced structure access penalties and branch mis-prediction penalties at higher clock periods. For both the integer and floating point benchmarks the frequency of memory operations is greater than the frequency of branches. In addition, the branch mis-prediction penalty is paid infrequently due to high branch prediction accuracy. Therefore IPC is more sensitive to the level-1 data cache access penalty than the branch mis-prediction penalty.

The IPC of both benchmarks show a sharp rise for the clock periods between 2 FO4 and 6 FO4. This sharp rise is due to the decrease in the access penalty of multiple structures. Though the reduced penalties of all structures contribute to the improvement the level-1 data cache has the most significant effect, since on average 1 in 5 instructions is a memory operation. The integer benchmark IPC shows a smaller improvement for clock periods between 6 FO4 to 8 FO4. These periods correspond to locations in Table 3 and Table 4 where the penalty of only one structure or functional unit decreases. The integer IPC shows a smaller improvement for the clock periods between 6 FO4 and 8 FO4. Table 3 shows that at these clock periods the access penalty of just one structure or functional unit decrease. Correspondingly the improvement in IPC is also reduced. For clock periods between 8 FO4 and 11 FO4 the access penalties of multiple structures and the most commonly used integer unit (Add) decrease. Therefore IPC shows a much larger improvement for this interval. After the 11 FO4 clock period the access penalties of the various microarchitectural structures and integer functional units remain more or less unchanged. Consequently the IPC of the integer benchmarks remains almost constant. A small increase in IPC between 14 FO4 and 15 FO4 clock periods occurs due to a 1 cycle decrease in DL1 access penalty at 15 FO4.

The floating point benchmarks have a lot more parallelism as compared the the integer benchmarks and fewer branches. Therefore their IPCs are inherently less sensitive to structure structure access penalties and mis-predictions. Furthermore, the Alpha 21264 architecture is capable of issuing only two floating point instructions every cycle which limits their absolute IPC. Accordingly, the impact of varying pipeline depth and structure access penalties has a significantly smaller effect on the IPCs of floating point benchmarks. A combination of the above factors contributes to a fairly smooth IPC curve for the floating point benchmarks.

Instruction throughput is only one part of the equation governing performance, the other half is clock frequency. A reduction in instruction throughput can be tolerated if the corresponding increase in clock frequency results in improved performance.

A processor's clock frequency is determined by the amount of logic in it's slowest pipeline stage and the latch overhead. Latch overhead varies between 1-2.5 FO4 depending on the design of the latch that is used. In Figure 16 we plot the absolute performance of the SPEC 2000 benchmarks at different latch overheads. The y-axis on this graph shows performance in terms of billions of instructions per second. The x-axis shows the clock period (not counting latch overhead) in units of FO4. The graph plots five performance curves corresponding to five different latch overheads (0 FO4 - 2 FO4).

For the curve assuming perfect latches with no overheads, the clock frequency increases faster than the rate at which IPC decreases. Therefore overall performance increases as the clock period decreases. However, perfect latches are not a realistic assumption. Real latches have setup and hold times that have to be adhered to in addition to the propagation delay through the latch itself. The curves that assume latch overheads ranging from 1 to 2.5 FO4 show that the best performance is achieved with 6 FO4 useful logic at every pipeline stage. At clock periods smaller than 6 FO4 the decrease in IPC combined with the latch overhead is too large and the improvement in clock frequency is not enough to improve performance. At clock periods larger than 6 FO4 the improvement in IPC is not enough to account for the loss in performance due to lower clock frequency.

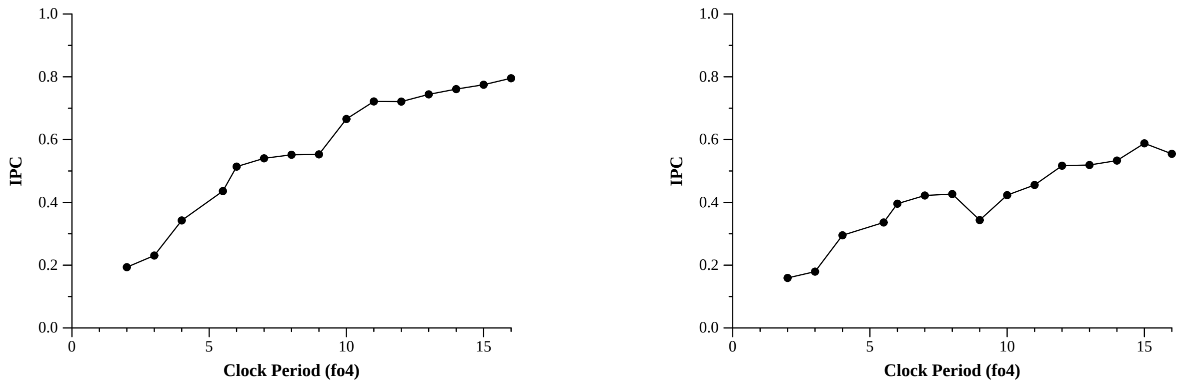


Figure 17: 100nm Technology Capacity Scaling IPC Trend

The integer benchmarks show a small dip in performance over the period between 6-8 FO4. As explained earlier, the slope of the IPC curve for integer benchmarks between 6-8 FO4 is lower than the slope between 8-11 FO4. This reduced slope accounts for the small drop in performance. In contrast, the floating point benchmark performance curve is very flat. The best performance is at 6 FO4 but the performance at adjacent clock periods is comparable. This behavior is due to the smooth, almost linear increase in the IPC curve of the floating point benchmarks.

Appendix B shows the instruction throughput and performance graphs for the experiments conducted at other technologies. The IPC and the performance graphs show the same trend across technologies for both sets of benchmarks. The optimal performance across all technologies is achieved with 6 FO4 levels of useful logic at each stage.

5 Capacity Scaling

Deep pipelines tend to reduce IPC due to large branch mis-prediction penalties. One design option is to restrict the capacity of microarchitectural structures so that their access penalties, and therefore overall pipeline depth, remain small. Reducing the capacities of microarchitectural structures would decrease their performance and accuracy. For example, reducing the capacity of the branch predictor reduces the accuracy of the predictions. While shallow pipelines benefit IPC due to the smaller mis-prediction penalties, the reduced capacities of microarchitectural structures will have an adverse impact on IPC. In this section we study the tradeoff between microarchitectural structure capacities and pipeline depths.

We consider six clock frequencies for our studies. The latencies of the various structures were found using Cacti. In Table 5 and Table 6 we describe the structure sizes used in these experiments at 100nm technology. The structure latencies are shown in parenthesis. All structure capacities are specified in terms of the number of entries except for the caches. For the branch predictor we show two sizes representing the capacities of the local and global predictors. In selecting these capacities we attempted to find the largest possible size for each structure that could be accessed with the same latency as the corresponding structure in the Alpha 21264. In situations where we could not reach a reasonably sized structure we increased the access latency permitting a structure large enough to be useful. Appendix C describes the sizes and latencies of structures used for other technologies.

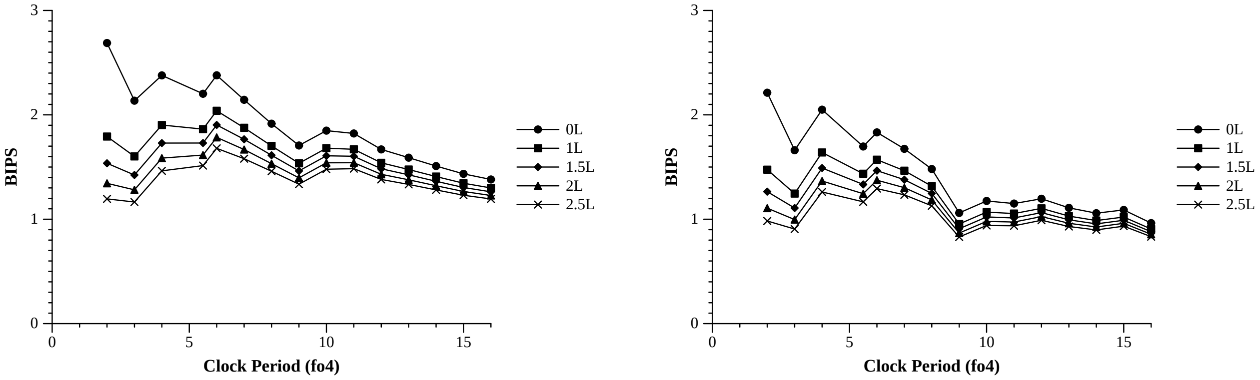
In Figure 17 we show the harmonic mean of the IPCs across the SPEC 2000 benchmark suite

Clock (FO4)	DL1(Bytes)	DTLB	ITLB	L2(Bytes)	Branch Pred.	ROB
2	8192(13)	128(10)	128(10)	32768(10)	256,1024 (8)	16(5)
3	8192(9)	512(7)	512(7)	32768(7)	128,512 (5)	8(3)
4	8192(7)	128(5)	128(5)	32768(5)	256,1024 (4)	64(3)
5.5	32768(6)	1024(4)	1024(4)	131072(6)	256,1024 (3)	32(2)
6	8192(5)	1024(4)	1024(4)	131072(6)	512,2048 (3)	64(2)
7	32768(5)	512(3)	512(3)	131072(5)	2048,8192 (3)	128(2)
8	8192(4)	1024(3)	1024(3)	131072(4)	256,1024 (2)	128(2)
9	16384(4)	1024(3)	1024(3)	262144(6)	512,2048 (2)	8(1)
10	65536(4)	128(2)	128(2)	262144(5)	2048,8192 (2)	16(1)
11	65536(4)	1024(2)	1024(2)	262144(5)	4096,16384 (2)	32(1)
12	8192(3)	1024(2)	1024(2)	524288(6)	4096,16384 (2)	64(1)
13	16384(3)	1024(2)	1024(2)	524288(6)	8192,32768 (2)	96(1)
14	32768(3)	1024(2)	1024(2)	524288(6)	8192,32768 (2)	128(1)
15	65536(3)	1024(2)	1024(2)	524288(5)	128,512 (1)	128(1)
16	65536(3)	1024(2)	1024(2)	524288(5)	256,1024 (1)	128(1)

Table 5: Capacity scaling structure sizes and latencies at 100nm technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	28(9)	8(8)	100(6)	100(9)
3	28(6)	28(6)	100(4)	100(6)
4	28(5)	8(4)	100(3)	100(5)
5.5	28(3)	12(3)	80(2)	80(4)
6	28(3)	28(3)	100(2)	100(3)
7	28(3)	28(3)	100(2)	100(3)
8	28(3)	8(2)	100(2)	100(3)
9	28(2)	28(2)	100(2)	100(2)
10	28(2)	28(2)	100(2)	100(2)
11	28(2)	28(2)	80(1)	80(2)
12	28(2)	28(2)	100(1)	100(2)
13	28(2)	28(2)	100(1)	100(2)
14	28(2)	28(2)	100(1)	100(2)
15	28(2)	28(2)	100(1)	100(2)
16	28(2)	8(1)	100(1)	100(2)

Table 6: Capacity scaling structure sizes and latencies at 100nm technology



Integer Floating pt.

Figure 18: 100nm Technology Capacity Scaling Performance Trend

at different clock periods (not including latch overhead). The graph shows that in general, as the clock period increases IPC of the integer and floating point benchmarks increase. The improvement in IPC at the lower frequencies is due to the reduced access penalties and increase in capacity of microarchitectural structures. Overall the IPC of the integer benchmarks reduces by 76% when the clock period is reduced from 16 FO4 to 2 FO4. The IPC of floating point benchmarks reduces by approximately 82% over the same period.

The integer benchmarks show an increase in IPC till a clock period of 6 FO4. For clocks between 7 FO4 and 9 FO4 IPC remains almost constant. At these clock periods Table 5 shows that the access penalty of level-1 data cache, the branch predictor and the ROB decrease but their capacities also decrease. The benefits of reduced structure access penalties are lost due to a corresponding reduction of their capacities. For clock periods 7-8 FO4 the floating point benchmarks also show no change. At 9 FO4 their IPC decreases sharply due to a decrease in the capacity of the ROB. The the integer benchmarks are not as sensitive to the ROB capacity as the floating point benchmarks. Therefore their IPC does not decrease when the ROB capacity is decreased.

A comparison of the capacity and pipeline scaling results in Figure 17 and Figure 15 shows that both techniques yield almost identical performance. The capacity scaling technique results in shorter pipelines and therefore lower branch mis-prediction penalties. However, this benefit is offset by reduced structure capacities. Similarly, for the pipeline scaling technique the benefit of increased structure capacities are offset by a higher branch mis-prediction penalty.

In Figure 18 we show the performance of the SPEC 2000 benchmarks at the different clocks. As before we consider five different latch overheads. The best performance for both sets of benchmarks is obtained at a 2 FO4 clock if latch overhead is not considered. However, when latch overhead is taken into account the integer benchmarks show the best performance when the amount of useful logic per stage corresponds to a delay of 6 FO4. For the floating point benchmarks optimal performance is obtained with 4 FO4 amount of useful delay per stage with small latch overhead assumptions. For latch overheads of 1.5 FO4 and greater optimal performance is reached with a delay of 6 FO4 per stage. In Appendix D we show throughput and performance graphs for other technologies. We see similar trends at the other technologies we considered.

6 Conclusions

The latency and capacity of microarchitectural structures have a strong influence on instruction throughput. From the sensitivity analysis we see that among microarchitectural structures the latency of the branch predictor has the largest impact on IPC for the integer benchmarks. The SPEC2000 integer benchmarks are control intensive, approximately one in every six instructions is a branch therefore branch predictor latency has a large impact on IPC. The floating-point benchmarks, in contrast have much fewer branches and therefore they are not very sensitive to branch predictor latency. Besides the branch predictor latency, IPC is most sensitive to data cache latencies. For the integer benchmarks IPC falls by 41% when the DL1 latency is increased from 1 to 10 cycles, the floating-point benchmarks show a drop in IPC of about 25%. For both sets of benchmarks branch prediction accuracy is crucial to obtain high instruction throughput at long pipeline depths. When structure capacities are increased, for almost all structures the IPC rapidly increases with capacity for the smaller structure sizes. For small structure sizes the hit rate or accuracy of the structures is extremely poor yielding very low IPCs. As the structure size is increased the hit rate of structure gradually improves resulting in better IPC. However, beyond a certain capacity the performance of the structure remains a constant, thereafter IPC shows no improvement. The capacity of the re-order buffer and the register file have a greater impact on the IPCs of floating-point benchmarks than the integer benchmarks. The floating-point benchmarks are representative of scientific code and have greater instruction parallelism. Increasing the capacity of the register file or the re-order buffer enables greater exploitation of this parallelism.

We presented two strategies to scale micro-architectures – pipeline and capacity scaling. In the pipeline scaling strategy we hold structure capacities to be constant while allowing their latencies to be scaled according to technology and clock frequency. As clock frequencies are increased the latencies of the structures, and therefore the overall pipeline length, increases. Large pipeline depths cause greater branch mis-prediction penalties resulting in lower IPCs. In addition, increasing functional unit latencies delay the execution of dependent instructions. These factors cause IPC to decrease. Our studies show that, across technologies, maximum performance is obtained when the amount of useful work per stage is close to 6 FO4.

For the capacity scaling strategy we attempted to keep the access penalties of structures to be as close to baseline penalties. At high clock frequencies the structure capacities are small and they have large access penalties. The small structures are unable to store enough program state and therefore the structure performance (i.e. hit rate or prediction accuracy) is poor. This poor structure performance results in very low instruction throughput. As the clock frequency is reduced larger structures can be accessed resulting in better structure performance. Therefore with decreasing clock frequency IPC increases. For the capacity scaling strategy also the best performance is achieved when the amount of useful logic per stage is 6 FO4.

References

[1] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar, "The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pp. 14–24, May 2002.

[2] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate vs. ipc : The end of the road for conventional microprocessors," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 248–259, June 2000.

[3] "The international technology roadmap for semiconductors." Semiconductor Industry Association, 1999.

[4] R. Desikan, D. Burger, and S. W. Keckler, "Measuring experimental error in microprocessor simulation," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[5] S. J. Wilton and N. P. Jouppi, "An enhanced access and cycle time model for on-chip caches," Tech. Rep. WRL-TR-93.5, Compaq Western Research Lab, July 1993.

[6] V. Agarwal, S. W. Keckler, and D. Burger, "The effect of technology scaling on microarchitectural structures," Tech. Rep. TR2000-02, Computer Architecture and Technology Lab, November 2000.

[7] "Compiler writer's guide for the alpha 21264," Tech. Rep. EC-RJ66A-TE, Compaq Computer Corporation, May 2000.

[8] S. Palacharla, N. Jouppi, and J. Smith, "Complexity-effective superscalar processors," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.

A Pipeline Scaling: Access Penalties of Structures and Functional Units

This section shows the access latencies of microarchitectural structures at different processes technologies for clock periods (not counting latch overhead)ranging from 3.5 FO4 to 17.5 FO4. The capacities of these structures are shown in Table 2, in Section 2.1. In all the tables the column titled "Clock" represents the amount of useful work per pipeline stage (ie. the latch overhead of 1.5 FO4 is not included).

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	15	10	87	8	6	8	8	6
3	10	7	58	6	4	6	5	4
4	8	5	44	4	3	4	4	3
5.5	6	4	32	3	3	3	3	2
6	6	4	29	3	2	3	3	2
7	5	3	25	3	2	3	3	2
8	5	3	22	2	2	2	2	2
9	4	3	20	2	2	2	2	2
10	4	2	18	2	2	2	2	2
11	4	2	16	2	2	2	2	1
12	4	2	15	2	1	2	2	1
13	4	2	14	2	1	2	2	1
14	3	2	13	2	1	2	2	1
15	3	2	12	2	1	2	1	1
16	3	2	11	1	1	1	1	1

Table 7: Pipeline scaling structure latencies at 250nm technology

Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 8: Pipeline scaling operation latencies at 250nm technology

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	17	11	75	9	6	9	9	6
3	12	7	50	6	4	6	6	4
4	9	6	38	5	3	5	5	3
5.5	7	4	28	4	3	4	4	2
6	7	4	25	3	2	3	3	2
7	6	3	22	3	2	3	3	2
8	5	3	19	3	2	3	3	2
9	5	3	17	2	2	2	2	2
10	5	3	15	2	2	2	2	2
11	4	2	14	2	2	2	2	1
12	4	2	13	2	1	2	2	1
13	4	2	12	2	1	2	2	1
14	4	2	11	2	1	2	2	1
15	4	2	10	2	1	2	2	1
16	3	2	10	2	1	2	2	1

Table 9: Pipeline scaling structure latencies at 180nm technology

Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 10: Pipeline scaling operation latencies at 180nm technology

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	16	11	82	9	7	9	9	6
3	11	7	55	6	5	6	6	4
4	9	6	41	5	4	5	5	3
5.5	7	4	30	4	3	4	4	2
6	6	4	28	3	3	3	3	2
7	6	3	24	3	2	3	3	2
8	5	3	21	3	2	3	3	2
9	5	3	19	2	2	2	2	2
10	4	3	17	2	2	2	2	2
11	4	2	15	2	2	2	2	1
12	4	2	14	2	2	2	2	1
13	4	2	13	2	1	2	2	1
14	4	2	12	2	1	2	2	1
15	3	2	11	2	1	2	2	1
16	3	2	11	2	1	2	2	1

Table 11: Pipeline scaling structure latencies at 130nm technology

Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 12: Pipeline scaling operation latencies at 130nm technology

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	16	11	89	10	7	9	9	6
3	11	7	60	7	5	6	6	4
4	9	6	45	5	4	5	5	3
5.5	7	4	33	4	3	4	4	2
6	6	4	30	4	3	3	3	2
7	6	3	26	3	2	3	3	2
8	5	3	23	3	2	3	3	2
9	5	3	20	3	2	2	2	2
10	4	3	18	2	2	2	2	2
11	4	2	17	2	2	2	2	1
12	4	2	15	2	2	2	2	1
13	4	2	14	2	1	2	2	1
14	4	2	13	2	1	2	2	1
15	3	2	12	2	1	2	2	1
16	3	2	12	2	1	2	2	1

Table 13: Pipeline scaling structure latencies at 100nm technology

Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 14: Pipeline scaling operation latencies at 100nm technology

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	15	10	82	10	7	8	9	6
3	10	7	55	7	5	6	6	4
4	8	5	41	5	4	4	5	3
5.5	6	4	30	4	3	3	3	3
6	6	4	28	4	3	3	3	2
7	5	3	24	3	2	3	3	2
8	5	3	21	3	2	2	3	2
9	4	3	19	3	2	2	2	2
10	4	2	17	2	2	2	2	2
11	4	2	15	2	2	2	2	2
12	4	2	14	2	2	2	2	1
13	4	2	13	2	1	2	2	1
14	3	2	12	2	1	2	2	1
15	3	2	11	2	1	2	2	1
16	3	2	11	2	1	1	2	1

Table 15: Pipeline scaling structure latencies at 70nm technology

Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 16: Pipeline scaling operation latencies at 70nm technology

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	16	11	96	10	7	9	9	6
3	11	7	64	7	5	6	6	4
4	9	6	48	5	4	5	5	3
5.5	7	4	35	4	3	3	4	3
6	6	4	32	4	3	3	3	2
7	6	3	28	3	2	3	3	2
8	5	3	24	3	2	3	3	2
9	5	3	22	3	2	2	2	2
10	4	3	20	2	2	2	2	2
11	4	2	18	2	2	2	2	2
12	4	2	16	2	2	2	2	1
13	4	2	15	2	2	2	2	1
14	4	2	14	2	1	2	2	1
15	3	2	13	2	1	2	2	1
16	3	2	12	2	1	2	2	1

Table 17: Pipeline scaling structure latencies at 50nm technology

Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 18: Pipeline scaling operation latencies at 50nm technology

Clock (FO4)	DL1	DTLB	L2	Branch Pred.	ROB	Rename Table	Issue Window	Reg. File
2	17	11	108	10	8	9	9	7
3	12	7	72	7	5	6	6	5
4	9	6	54	5	4	5	5	4
5.5	7	4	40	4	3	4	4	3
6	7	4	36	4	3	3	3	3
7	6	3	31	3	3	3	3	2
8	5	3	27	3	2	3	3	2
9	5	3	24	3	2	2	2	2
10	5	3	22	2	2	2	2	2
11	4	2	20	2	2	2	2	2
12	4	2	18	2	2	2	2	2
13	4	2	17	2	2	2	2	1
14	4	2	16	2	2	2	2	1
15	4	2	15	2	1	2	2	1
16	3	2	14	2	1	2	2	1

Table 19: Pipeline scaling structure latencies at 35nm technology

Clock (FO4)	Int. Add	Int. Mult.	Fp. Add	Fp. Div.	Fp. Sqrt	Fp. Mult.	Fp. Cmp.
2	9	61	35	105	157	35	35
3	6	41	24	70	105	24	24
4	5	31	18	53	79	18	18
5.5	4	23	13	38	57	13	13
6	3	21	12	35	53	12	12
7	3	18	10	30	45	10	10
8	3	16	9	27	40	9	9
9	2	14	8	24	35	8	8
10	2	13	7	21	32	7	7
11	2	12	7	19	29	7	7
12	2	11	6	18	27	6	6
13	2	10	6	17	25	6	6
14	2	9	5	15	23	5	5
15	2	9	5	14	21	5	5
16	2	8	5	14	20	5	5

Table 20: Pipeline scaling operation latencies at 35nm technology

B Pipeline Scaling: Instruction Throughput and Performance Graphs

This section shows the results of the pipeline scaling experiments, described in Section 4, for technology points ranging between 35-250nm. The plots in this section show the harmonic mean of IPC values, measured across the SPEC 2000 integer and floating point benchmarks, against clock period for a range of process technologies. We also show plots of the absolute performance (billions of instructions per second) against clock period. The microarchitectural structure latencies used for these experiments are described in Appendix A.

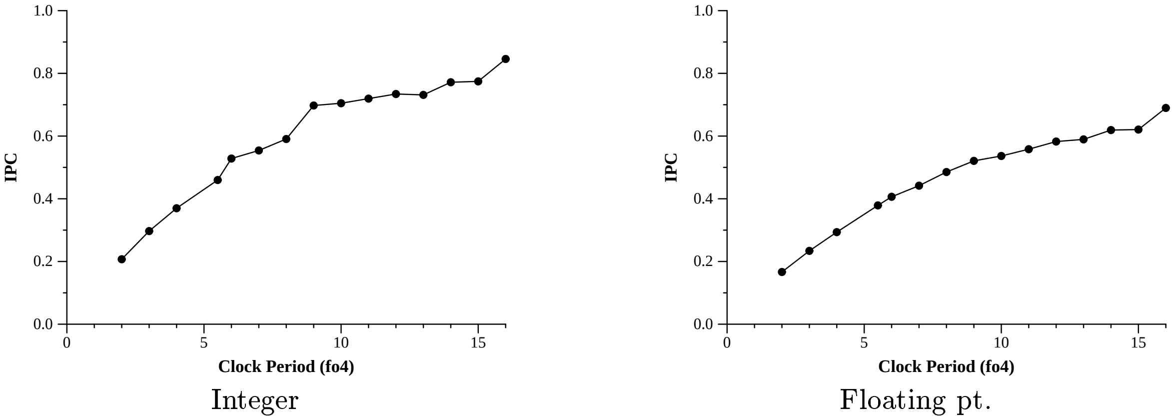


Figure 19: 250nm Technology Pipeline Scaling IPC Trend

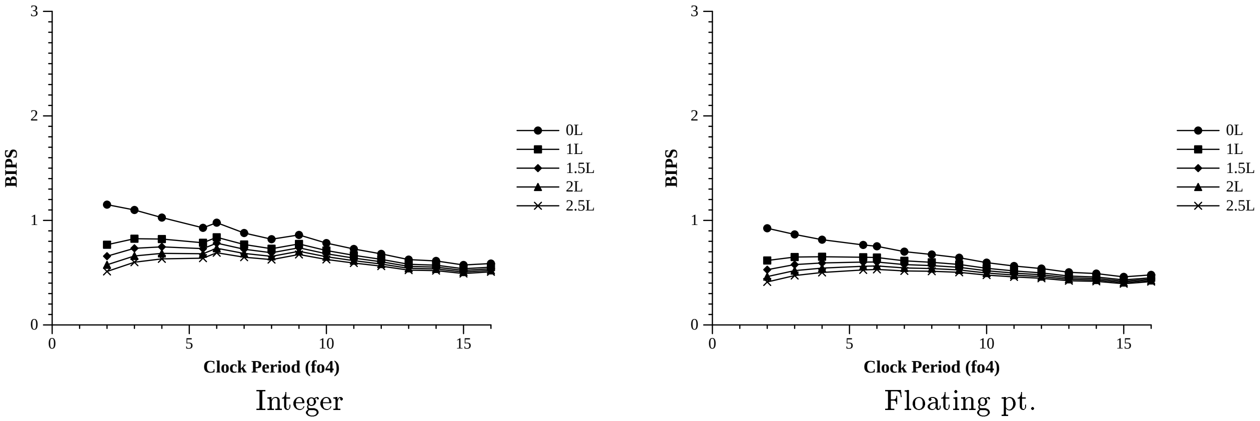
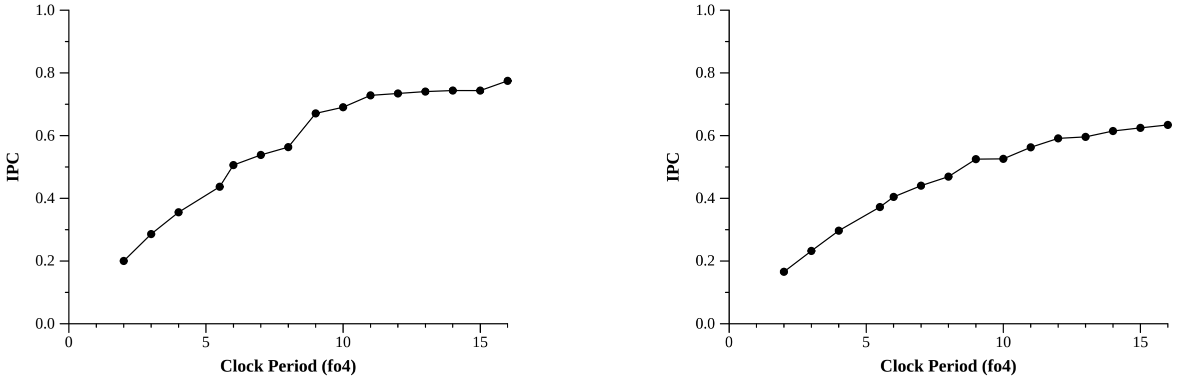
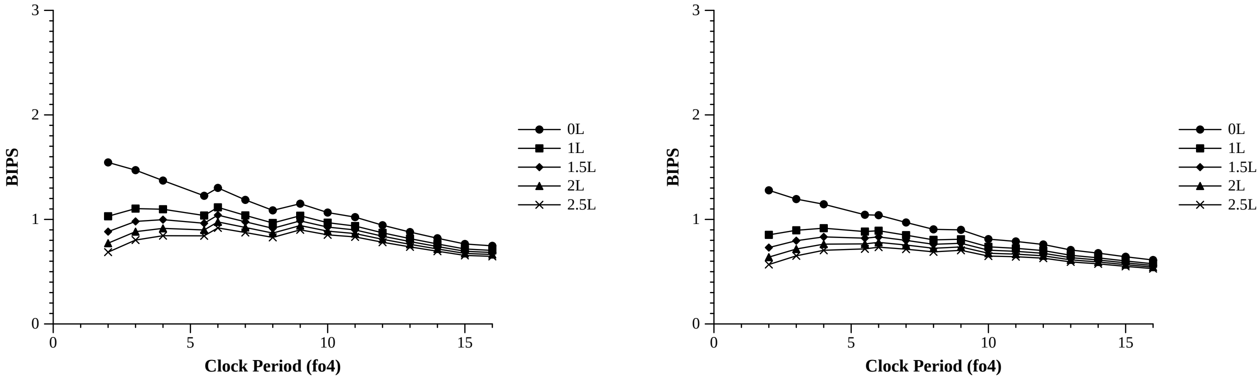


Figure 20: 250nm Technology Pipeline Scaling Performance Trend



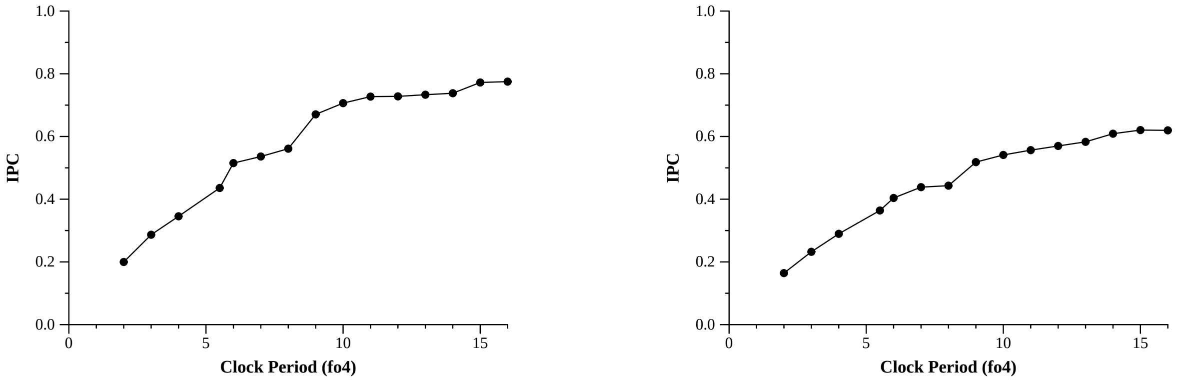
Integer Floating pt.

Figure 21: 180nm Technology Pipeline Scaling IPC Trend



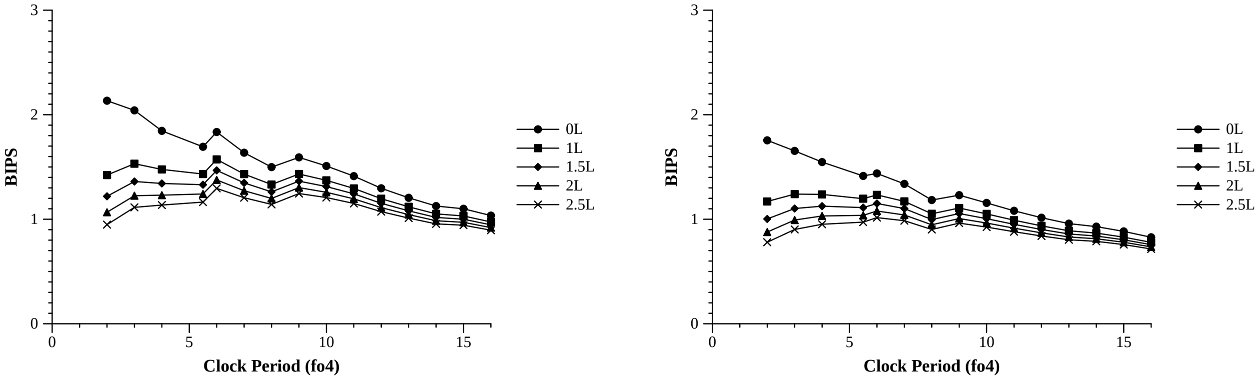
Integer Floating pt.

Figure 22: 180nm Technology Pipeline Scaling Performance Trend



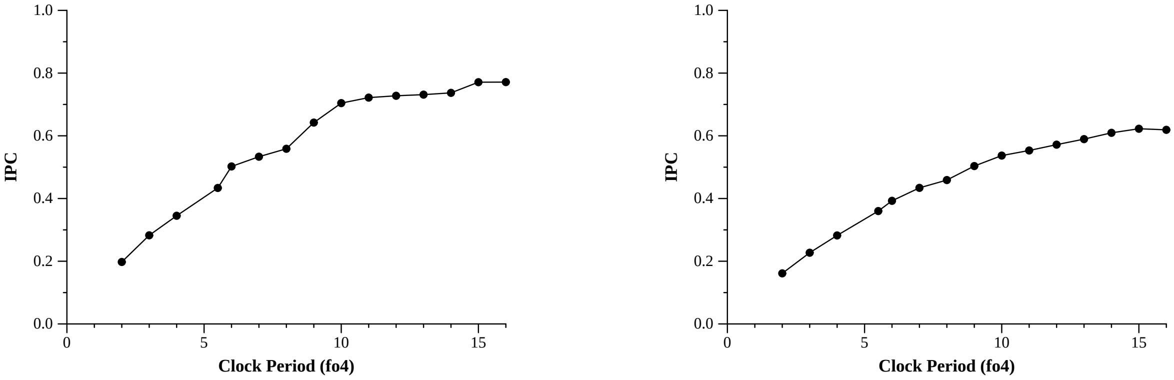
Integer Floating pt.

Figure 23: 130nm Technology Pipeline Scaling IPC Trend



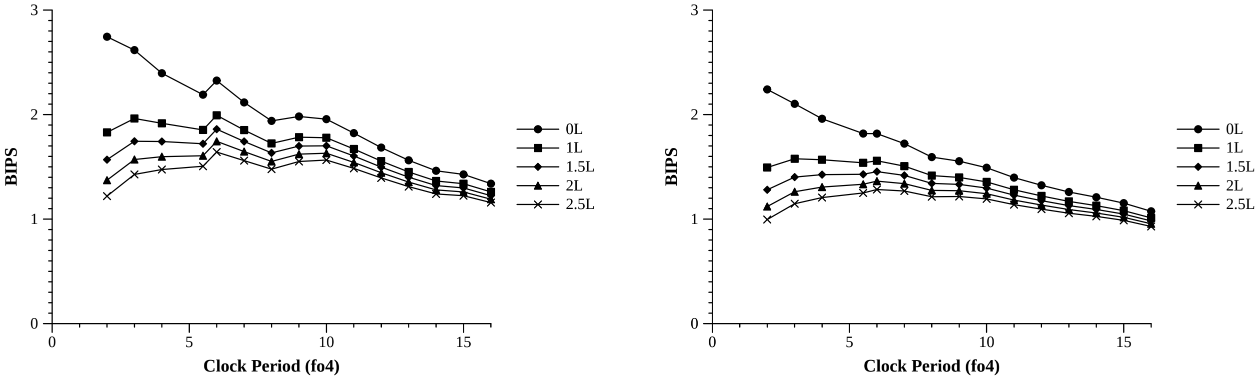
Integer Floating pt.

Figure 24: 130nm Technology Pipeline Scaling Performance Trend



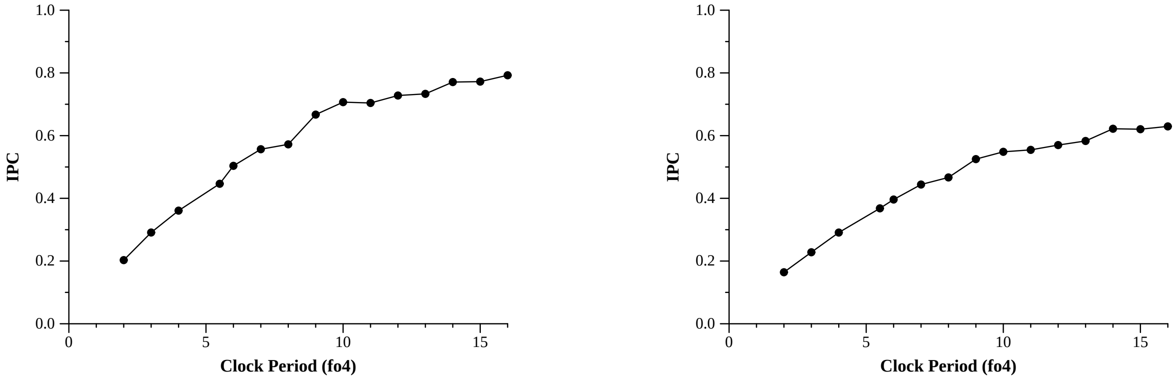
Integer Floating pt.

Figure 25: 100nm Technology Pipeline Scaling IPC Trend



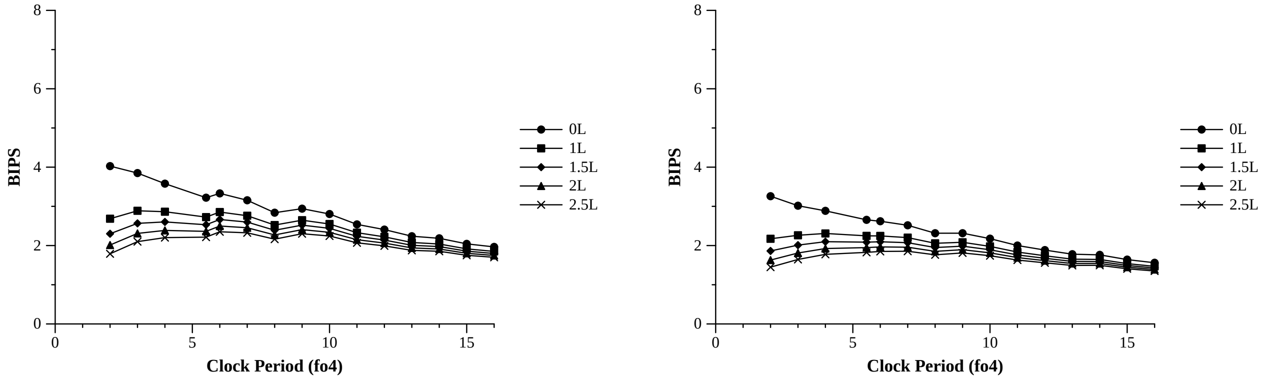
Integer Floating pt.

Figure 26: 100nm Technology Pipeline Scaling Performance Trend



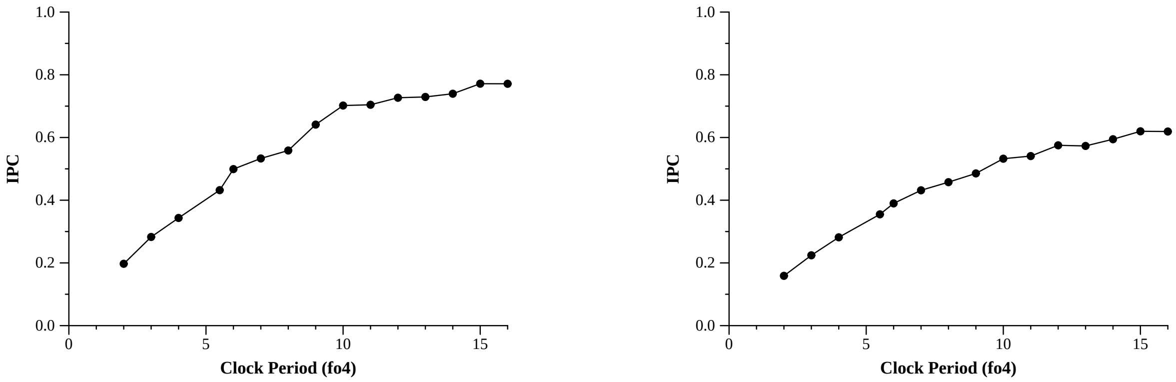
Integer Floating pt.

Figure 27: 70nm Technology Pipeline Scaling IPC Trend



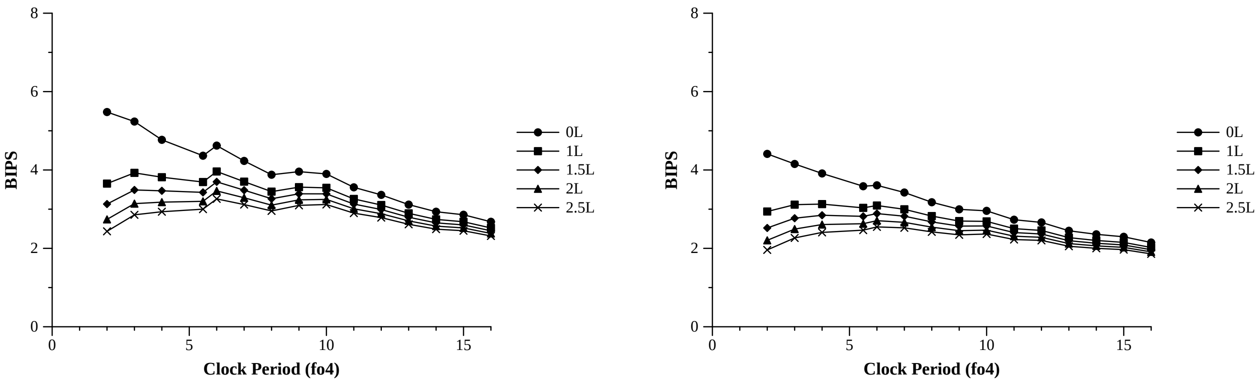
Integer Floating pt.

Figure 28: 70nm Technology Pipeline Scaling Performance Trend



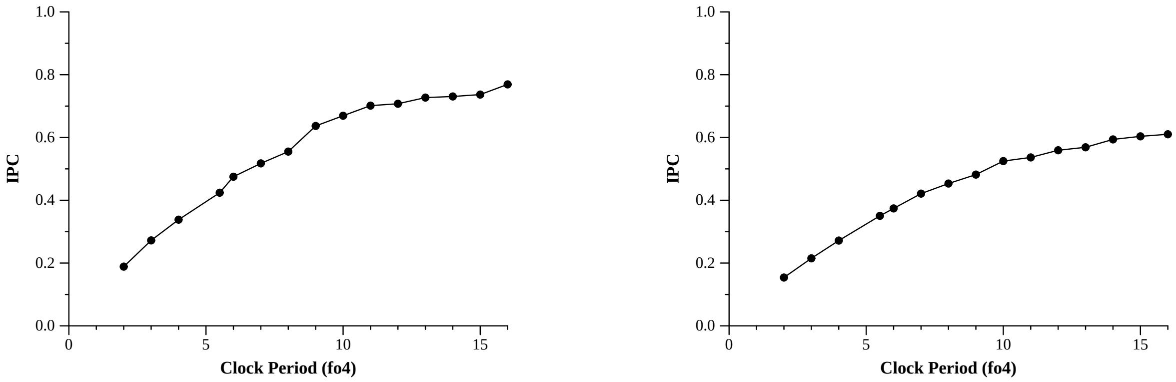
Integer Floating pt.

Figure 29: 50nm Technology Pipeline Scaling IPC Trend



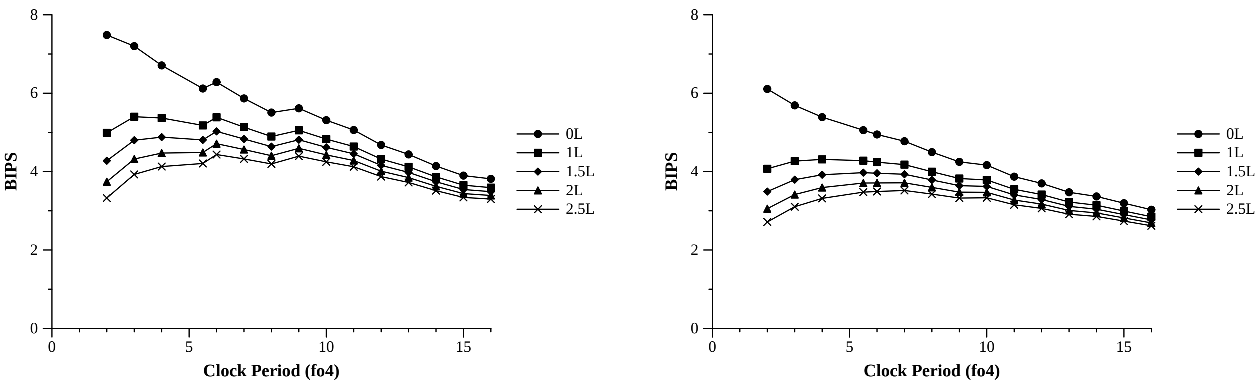
Integer Floating pt.

Figure 30: 50nm Technology Pipeline Scaling Performance Trend



Integer Floating pt.

Figure 31: 35nm Technology Pipeline Scaling IPC Trend



Integer Floating pt.

Figure 32: 35nm Technology Pipeline Scaling Performance Trend

C Capacity Scaling: Access Penalties and Capacities of Structures

This section shows the structure sizes and access latencies of microarchitectural structures used in capacity scaling experiments described in Section 5 for technologies ranging from 35-250nm. In all the tables the column titled "Clock" represents the amount of useful work per pipeline stage (ie. the latch overhead of 1.5 FO4 is not included).

Clock (FO4)	DL1(Bytes)	DTLB	L2(Bytes)	Branch Pred.	ROB
2	8192(11)	8(8)	16384(7)	256,1024 (7)	8(4)
3	8192(8)	512(6)	32768(6)	512,2048 (5)	16(3)
4	8192(6)	8(4)	65536(6)	1024,4096 (4)	8(2)
5.5	16384(5)	16(3)	131072(6)	2048,8192 (3)	75(2)
6	32768(5)	512(3)	131072(5)	4096,16384 (3)	96(2)
7	8192(4)	1024(3)	262144(6)	256,1024 (2)	128(2)
8	32768(4)	8(2)	262144(6)	1024,4096 (2)	8(1)
9	65536(4)	512(2)	262144(5)	4096,16384 (2)	16(1)
10	8192(3)	1024(2)	262144(5)	8192,32768 (2)	32(1)
11	16384(3)	1024(2)	524288(6)	8192,32768 (2)	75(1)
12	32768(3)	1024(2)	524288(6)	8192,32768 (2)	96(1)
13	32768(3)	1024(2)	524288(6)	128,512 (1)	128(1)
14	65536(3)	1024(2)	524288(5)	256,1024 (1)	128(1)
15	65536(3)	1024(2)	524288(5)	512,2048 (1)	128(1)
16	65536(3)	8(1)	524288(5)	1024,4096 (1)	128(1)

Table 21: Capacity scaling structure sizes and latencies at 250nm Technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	8(7)	8(7)	76(5)	76(8)
3	24(5)	28(5)	100(4)	100(6)
4	28(4)	28(4)	100(3)	100(4)
5.5	28(3)	28(3)	100(2)	100(3)
6	28(3)	28(3)	100(2)	100(3)
7	8(2)	8(2)	100(2)	100(3)
8	28(2)	28(2)	100(2)	100(2)
9	28(2)	28(2)	100(2)	100(2)
10	28(2)	28(2)	76(1)	76(2)
11	28(2)	28(2)	100(1)	100(2)
12	28(2)	28(2)	100(1)	100(2)
13	28(2)	28(2)	100(1)	100(2)
14	8(1)	8(1)	100(1)	100(2)
15	24(1)	28(1)	100(1)	100(2)
16	28(1)	28(1)	100(1)	100(1)

Table 22: Capacity scaling structure sizes and latencies at 250nm Technology

Clock (FO4)	DL1(Bytes)	DTLB	L2(Bytes)	Branch Pred.	ROB
2	1024(13)	64(10)	32768(9)	512,2048 (8)	24(5)
3	1024(9)	512(7)	32768(6)	256,1024 (5)	12(3)
4	1024(7)	64(5)	65536(6)	512,2048 (4)	80(3)
5.5	16384(6)	512(4)	131072(6)	512,2048 (3)	48(2)
6	1024(5)	1024(4)	131072(5)	1024,4096 (3)	80(2)
7	16384(5)	512(3)	262144(6)	4096,16384 (3)	128(2)
8	1024(4)	1024(3)	262144(6)	512,2048 (2)	128(2)
9	16384(4)	1024(3)	262144(5)	1024,4096 (2)	12(1)
10	32768(4)	64(2)	262144(5)	4096,16384 (2)	24(1)
11	65536(4)	512(2)	524288(6)	4096,16384 (2)	48(1)
12	1024(3)	1024(2)	524288(6)	8192,32768 (2)	80(1)
13	8192(3)	1024(2)	524288(5)	8192,32768 (2)	128(1)
14	16384(3)	1024(2)	524288(5)	8192,32768 (2)	128(1)
15	32768(3)	1024(2)	524288(5)	256,1024 (1)	128(1)
16	65536(3)	1024(2)	1048576(6)	512,2048 (1)	128(1)

Table 23: Capacity scaling structure sizes and latencies at 180nm Technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	24(9)	28(9)	100(6)	100(10)
3	24(6)	28(6)	100(4)	100(7)
4	28(5)	28(5)	100(3)	100(5)
5.5	8(3)	8(3)	92(2)	92(4)
6	24(3)	28(3)	100(2)	100(4)
7	28(3)	28(3)	100(2)	100(3)
8	28(3)	28(3)	100(2)	100(3)
9	24(2)	28(2)	100(2)	100(3)
10	28(2)	28(2)	100(2)	100(2)
11	28(2)	28(2)	92(1)	92(2)
12	28(2)	28(2)	100(1)	100(2)
13	28(2)	28(2)	100(1)	100(2)
14	28(2)	28(2)	100(1)	100(2)
15	28(2)	28(2)	100(1)	100(2)
16	28(2)	28(2)	100(1)	100(2)

Table 24: Capacity scaling structure sizes and latencies at 180nm Technology

Clock (FO4)	DL1(Bytes)	DTLB	L2(Bytes)	Branch Pred.	ROB
2	8192(13)	128(10)	32768(10)	128,512 (7)	24(5)
3	8192(9)	512(7)	32768(7)	256,1024 (5)	8(3)
4	8192(7)	128(5)	65536(6)	256,1024 (4)	75(3)
5.5	32768(6)	1024(4)	131072(6)	512,2048 (3)	48(2)
6	8192(5)	1024(4)	131072(5)	1024,4096 (3)	75(2)
7	32768(5)	512(3)	131072(5)	128,512 (2)	128(2)
8	8192(4)	1024(3)	262144(6)	256,1024 (2)	128(2)
9	16384(4)	1024(3)	262144(5)	1024,4096 (2)	8(1)
10	65536(4)	128(2)	262144(5)	3072,12288 (2)	24(1)
11	65536(4)	1024(2)	524288(6)	4096,16384 (2)	48(1)
12	8192(3)	1024(2)	524288(6)	8192,32768 (2)	75(1)
13	16384(3)	1024(2)	524288(6)	8192,32768 (2)	96(1)
14	32768(3)	1024(2)	524288(5)	128,512 (1)	128(1)
15	65536(3)	1024(2)	524288(5)	256,1024 (1)	128(1)
16	65536(3)	1024(2)	524288(5)	256,1024 (1)	128(1)

Table 25: Capacity scaling structure sizes and latencies at 130nm Technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	28(9)	8(8)	100(6)	100(9)
3	28(6)	28(6)	100(4)	100(6)
4	28(5)	8(4)	100(3)	100(5)
5.5	8(3)	12(3)	92(2)	92(4)
6	28(3)	28(3)	100(2)	100(3)
7	28(3)	28(3)	100(2)	100(3)
8	28(3)	8(2)	100(2)	100(3)
9	28(2)	28(2)	100(2)	100(2)
10	28(2)	28(2)	100(2)	100(2)
11	28(2)	28(2)	92(1)	92(2)
12	28(2)	28(2)	100(1)	100(2)
13	28(2)	28(2)	100(1)	100(2)
14	28(2)	28(2)	100(1)	100(2)
15	28(2)	28(2)	100(1)	100(2)
16	28(2)	8(1)	100(1)	100(2)

Table 26: Capacity scaling structure sizes and latencies at 130nm Technology

Clock (FO4)	DL1(Bytes)	DTLB	L2(Bytes)	Branch Pred.	ROB
2	8192(13)	128(10)	32768(10)	256,1024 (8)	16(5)
3	8192(9)	512(7)	32768(7)	128,512 (5)	8(3)
4	8192(7)	128(5)	32768(5)	256,1024 (4)	64(3)
5.5	32768(6)	1024(4)	131072(6)	256,1024 (3)	32(2)
6	8192(5)	1024(4)	131072(6)	512,2048 (3)	64(2)
7	32768(5)	512(3)	131072(5)	2048,8192 (3)	128(2)
8	8192(4)	1024(3)	131072(4)	256,1024 (2)	128(2)
9	16384(4)	1024(3)	262144(6)	512,2048 (2)	8(1)
10	65536(4)	128(2)	262144(5)	2048,8192 (2)	16(1)
11	65536(4)	1024(2)	262144(5)	4096,16384 (2)	32(1)
12	8192(3)	1024(2)	524288(6)	4096,16384 (2)	64(1)
13	16384(3)	1024(2)	524288(6)	8192,32768 (2)	96(1)
14	32768(3)	1024(2)	524288(6)	8192,32768 (2)	128(1)
15	65536(3)	1024(2)	524288(5)	128,512 (1)	128(1)
16	65536(3)	1024(2)	524288(5)	256,1024 (1)	128(1)

Table 27: Capacity scaling structure sizes and latencies at 100nm Technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	28(9)	8(8)	100(6)	100(9)
3	28(6)	28(6)	100(4)	100(6)
4	28(5)	8(4)	100(3)	100(5)
5.5	8(3)	12(3)	80(2)	80(4)
6	28(3)	28(3)	100(2)	100(3)
7	28(3)	28(3)	100(2)	100(3)
8	28(3)	8(2)	100(2)	100(3)
9	28(2)	28(2)	100(2)	100(2)
10	28(2)	28(2)	100(2)	100(2)
11	28(2)	28(2)	80(1)	80(2)
12	28(2)	28(2)	100(1)	100(2)
13	28(2)	28(2)	100(1)	100(2)
14	28(2)	28(2)	100(1)	100(2)
15	28(2)	28(2)	100(1)	100(2)
16	28(2)	8(1)	100(1)	100(2)

Table 28: Capacity scaling structure sizes and latencies at 100nm Technology

Clock (FO4)	DL1(Bytes)	DTLB	L2(Bytes)	Branch Pred.	ROB
2	2048(11)	8(9)	16384(9)	128,512 (7)	12(5)
3	8192(8)	8(6)	16384(6)	128,512 (5)	64(4)
4	2048(6)	512(5)	32768(5)	256,1024 (4)	64(3)
5.5	16384(5)	1024(4)	131072(6)	256,1024 (3)	32(2)
6	32768(5)	8(3)	131072(6)	512,2048 (3)	64(2)
7	8192(4)	1024(3)	131072(5)	128,512 (2)	128(2)
8	32768(4)	1024(3)	262144(6)	256,1024 (2)	128(2)
9	65536(4)	8(2)	262144(6)	512,2048 (2)	128(2)
10	2048(3)	512(2)	262144(5)	2048,8192 (2)	12(1)
11	16384(3)	1024(2)	262144(5)	4096,16384 (2)	32(1)
12	32768(3)	1024(2)	524288(6)	4096,16384 (2)	64(1)
13	32768(3)	1024(2)	524288(6)	8192,32768 (2)	80(1)
14	65536(3)	1024(2)	524288(5)	128,512 (1)	128(1)
15	65536(3)	1024(2)	524288(5)	128,512 (1)	128(1)
16	65536(3)	1024(2)	524288(5)	256,1024 (1)	128(1)

Table 29: Capacity scaling structure sizes and latencies at 70nm Technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	16(8)	16(8)	100(6)	100(9)
3	28(6)	28(6)	100(4)	100(6)
4	16(4)	16(4)	100(3)	100(5)
5.5	24(3)	24(3)	72(2)	72(3)
6	28(3)	28(3)	100(2)	100(3)
7	28(3)	28(3)	100(2)	100(3)
8	16(2)	16(2)	100(2)	100(3)
9	28(2)	28(2)	100(2)	100(2)
10	28(2)	28(2)	100(2)	100(2)
11	28(2)	28(2)	72(1)	72(2)
12	28(2)	28(2)	100(1)	100(2)
13	28(2)	28(2)	100(1)	100(2)
14	28(2)	28(2)	100(1)	100(2)
15	28(2)	28(2)	100(1)	100(2)
16	16(1)	16(1)	100(1)	100(2)

Table 30: Capacity scaling structure sizes and latencies at 70nm Technology

Clock (FO4)	DL1(Bytes)	DTLB	L2(Bytes)	Branch Pred.	ROB
2	1024(11)	8(9)	16384(9)	256,1024 (8)	8(5)
3	4096(8)	8(6)	16384(6)	128,512 (5)	48(4)
4	1024(6)	128(5)	32768(6)	256,1024 (4)	48(3)
5.5	8192(5)	1024(4)	65536(5)	256,1024 (3)	24(2)
6	16384(5)	8(3)	131072(6)	512,2048 (3)	48(2)
7	4096(4)	512(3)	131072(5)	2048,8192 (3)	96(2)
8	16384(4)	1024(3)	131072(5)	256,1024 (2)	128(2)
9	32768(4)	8(2)	131072(4)	512,2048 (2)	128(2)
10	1024(3)	128(2)	262144(6)	1024,4096 (2)	8(1)
11	8192(3)	1024(2)	262144(5)	2048,8192 (2)	24(1)
12	16384(3)	1024(2)	262144(5)	4096,16384 (2)	48(1)
13	32768(3)	1024(2)	524288(6)	8192,32768 (2)	64(1)
14	32768(3)	1024(2)	524288(6)	8192,32768 (2)	96(1)
15	65536(3)	1024(2)	524288(6)	128,512 (1)	128(1)
16	65536(3)	1024(2)	524288(5)	256,1024 (1)	128(1)

Table 31: Capacity scaling structure sizes and latencies at 50nm Technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	8(8)	8(8)	92(6)	92(9)
3	28(6)	28(6)	92(4)	92(6)
4	8(4)	8(4)	92(3)	92(5)
5.5	16(3)	20(3)	100(3)	100(3)
6	28(3)	28(3)	92(2)	92(3)
7	28(3)	28(3)	100(2)	100(3)
8	8(2)	8(2)	100(2)	100(3)
9	28(2)	28(2)	100(2)	100(2)
10	28(2)	28(2)	100(2)	100(2)
11	28(2)	28(2)	100(2)	100(2)
12	28(2)	28(2)	92(1)	92(2)
13	28(2)	28(2)	100(1)	100(2)
14	28(2)	28(2)	100(1)	100(2)
15	28(2)	28(2)	100(1)	100(2)
16	8(1)	8(1)	100(1)	100(2)

Table 32: Capacity scaling structure sizes and latencies at 50nm Technology

Clock (FO4)	DL1(Bytes)	DTLB	L2(Bytes)	Branch Pred.	ROB
2	2048(12)	128(10)	16384(9)	256,1024 (8)	8(5)
3	8192(9)	512(7)	16384(6)	128,512 (5)	32(4)
4	8192(7)	128(5)	32768(6)	256,1024 (4)	32(3)
5.5	2048(5)	1024(4)	65536(6)	256,1024 (3)	16(2)
6	8192(5)	1024(4)	65536(5)	512,2048 (3)	32(2)
7	32768(5)	512(3)	131072(6)	2048,8192 (3)	75(2)
8	8192(4)	1024(3)	131072(5)	256,1024 (2)	128(2)
9	32768(4)	1024(3)	131072(5)	512,2048 (2)	128(2)
10	32768(4)	128(2)	131072(4)	1024,4096 (2)	8(1)
11	2048(3)	1024(2)	262144(6)	2048,8192 (2)	16(1)
12	8192(3)	1024(2)	262144(6)	4096,16384 (2)	32(1)
13	16384(3)	1024(2)	262144(5)	8192,32768 (2)	48(1)
14	32768(3)	1024(2)	262144(5)	8192,32768 (2)	75(1)
15	32768(3)	1024(2)	524288(6)	128,512 (1)	96(1)
16	65536(3)	1024(2)	524288(6)	256,1024 (1)	128(1)

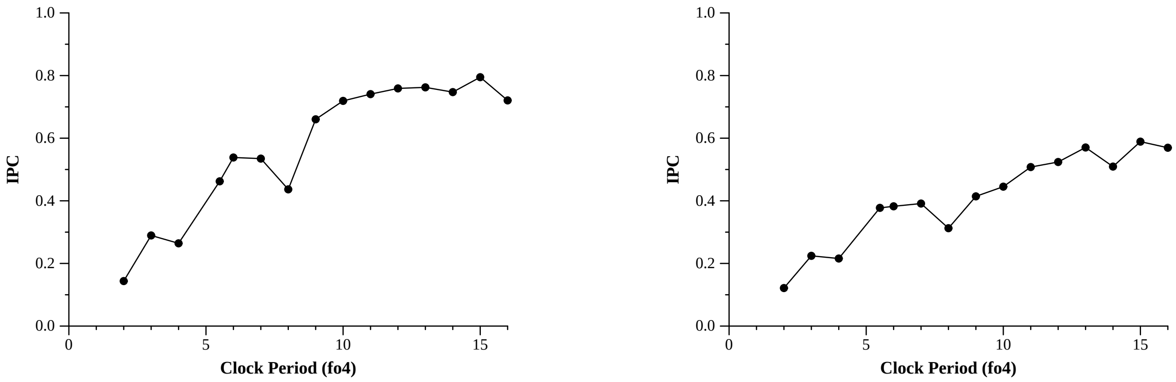
Table 33: Capacity scaling structure sizes and latencies at 35nm Technology

Clock (FO4)	Issue Window (Int)	Issue Window (Fp)	Reg. File	Rename Table
2	8(8)	8(8)	76(6)	76(9)
3	28(6)	28(6)	76(4)	76(6)
4	8(4)	8(4)	76(3)	76(5)
5.5	12(3)	16(3)	100(3)	100(4)
6	28(3)	28(3)	76(2)	76(3)
7	28(3)	28(3)	100(2)	100(3)
8	8(2)	8(2)	100(2)	100(3)
9	28(2)	28(2)	100(2)	100(2)
10	28(2)	28(2)	100(2)	100(2)
11	28(2)	28(2)	100(2)	100(2)
12	28(2)	28(2)	76(1)	76(2)
13	28(2)	28(2)	100(1)	100(2)
14	28(2)	28(2)	100(1)	100(2)
15	28(2)	28(2)	100(1)	100(2)
16	8(1)	8(1)	100(1)	100(2)

Table 34: Capacity scaling structure sizes and latencies at 35nm Technology

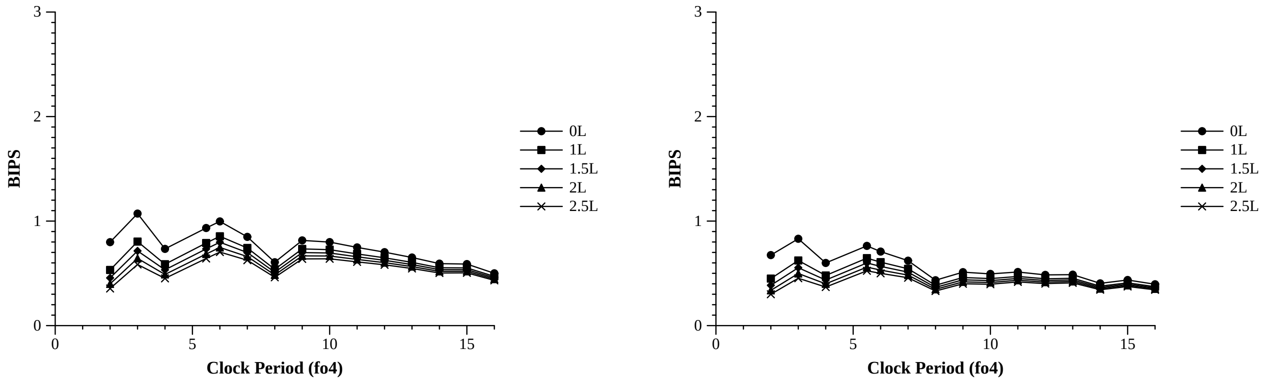
D Capacity Scaling: Instruction Throughput and Performance Graphs

This section shows the results of the capacity scaling experiments, described in Section 5, for technology points ranging between 35-250nm. The plots in this section show the harmonic mean of IPC values, measured across the SPEC 2000 integer and floating point benchmarks, against clock period (not counting latch overhead) for a range of process technologies. We also show plots of the absolute performance (billions of instructions per second) against clock period. The microarchitectural structure latencies used for these experiments are described in Appendix C.



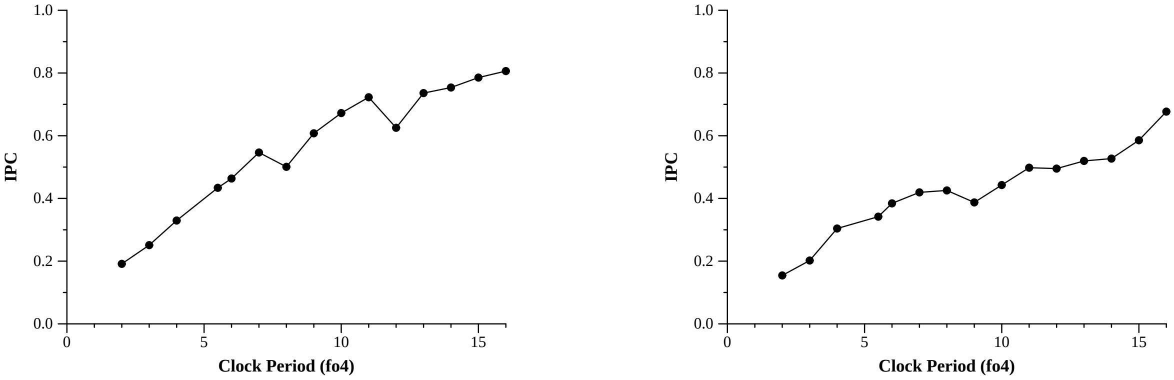
Integer Floating pt.

Figure 33: 250nm Technology Capacity Scaling IPC Trend



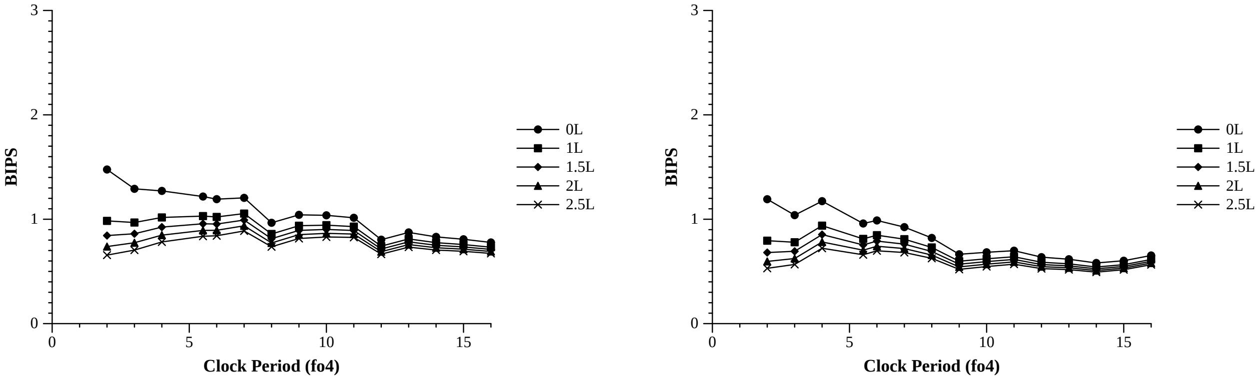
Integer Floating pt.

Figure 34: 250nm Technology Capacity Scaling Performance Trend



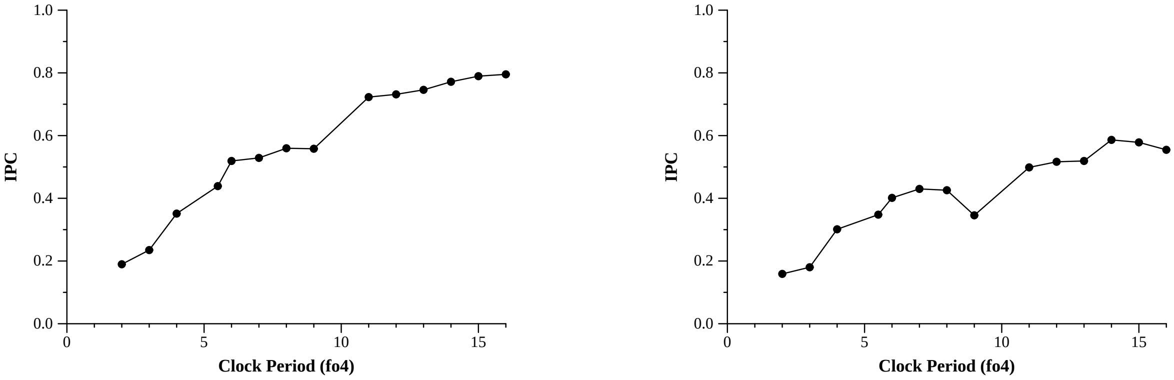
Integer Floating pt.

Figure 35: 180nm Technology Capacity Scaling IPC Trend



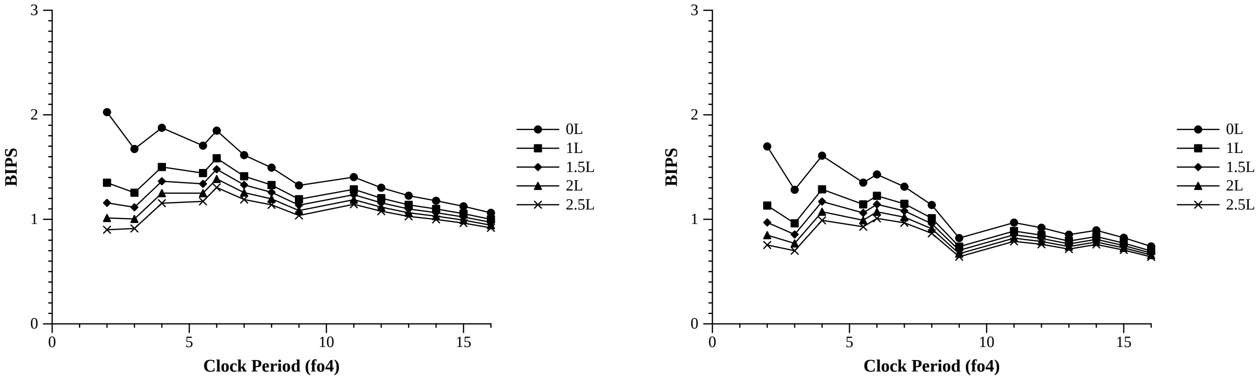
Integer Floating pt.

Figure 36: 180nm Technology Capacity Scaling Performance Trend



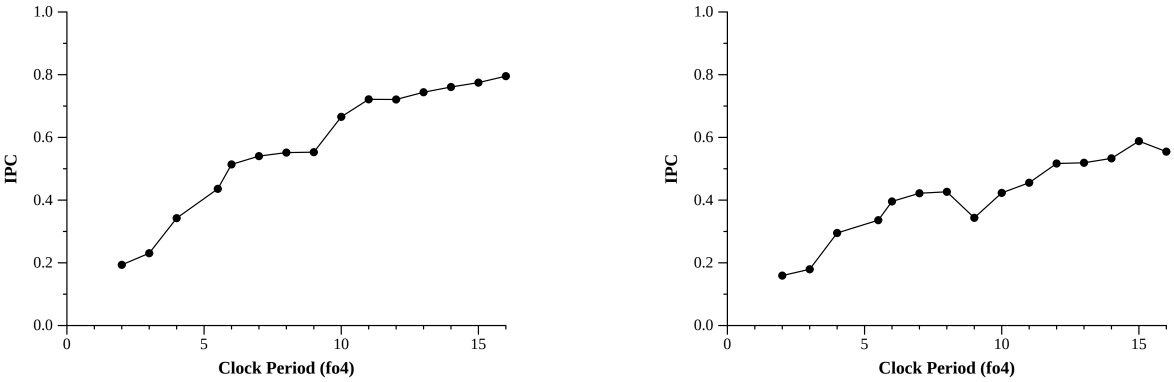
Integer Floating pt.

Figure 37: 130nm Technology Capacity Scaling IPC Trend



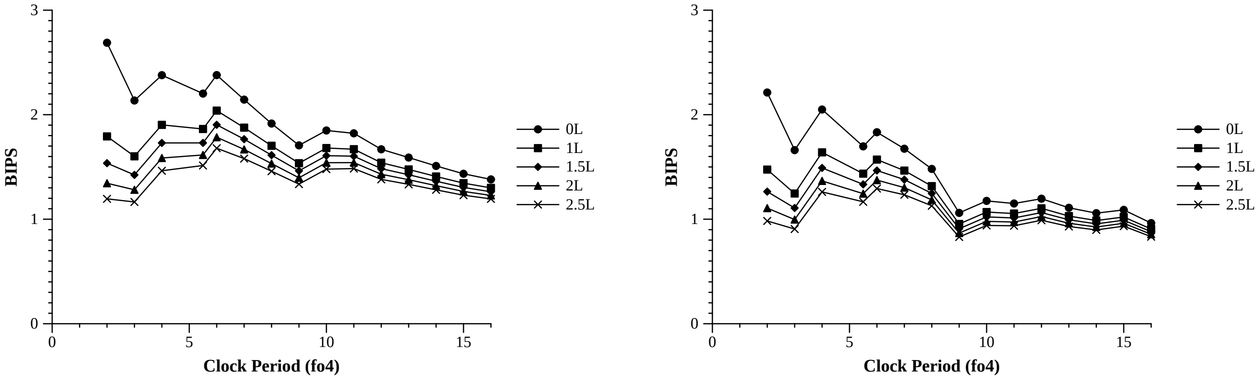
Integer Floating pt.

Figure 38: 130nm Technology Capacity Scaling Performance Trend



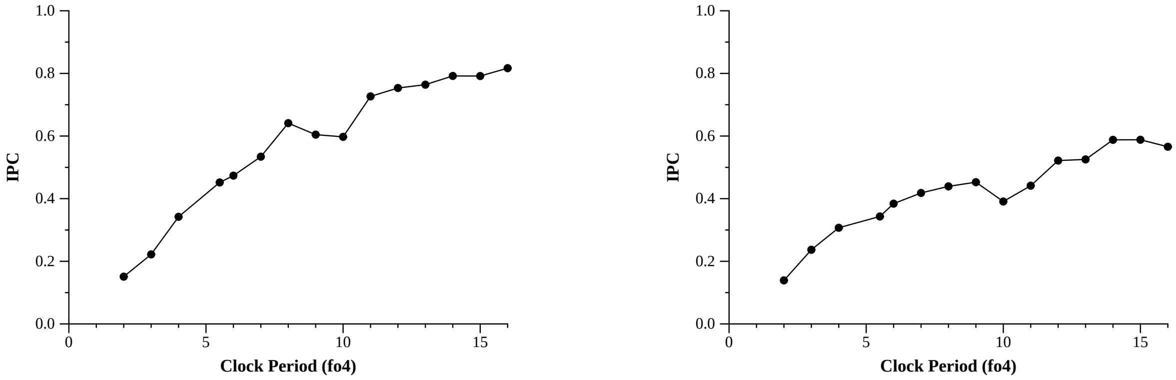
Integer Floating pt.

Figure 39: 100nm Technology Capacity Scaling IPC Trend



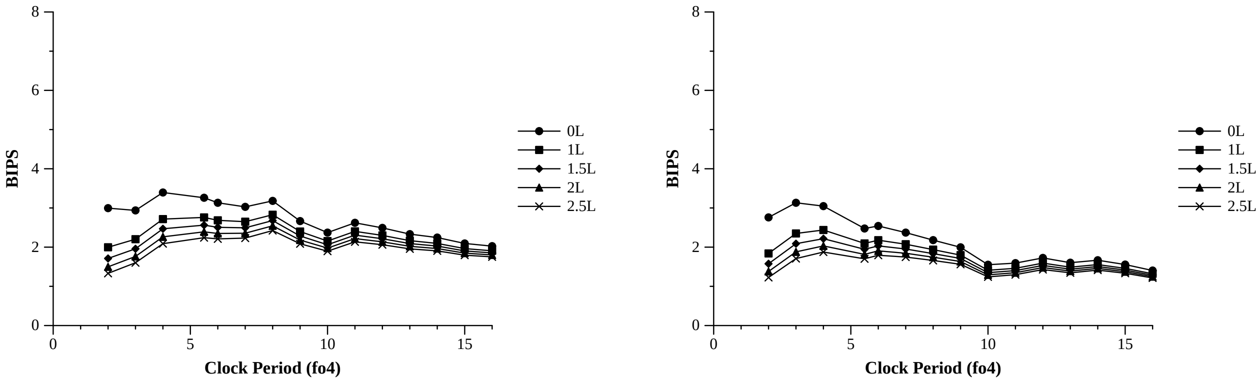
Integer Floating pt.

Figure 40: 100nm Technology Capacity Scaling Performance Trend



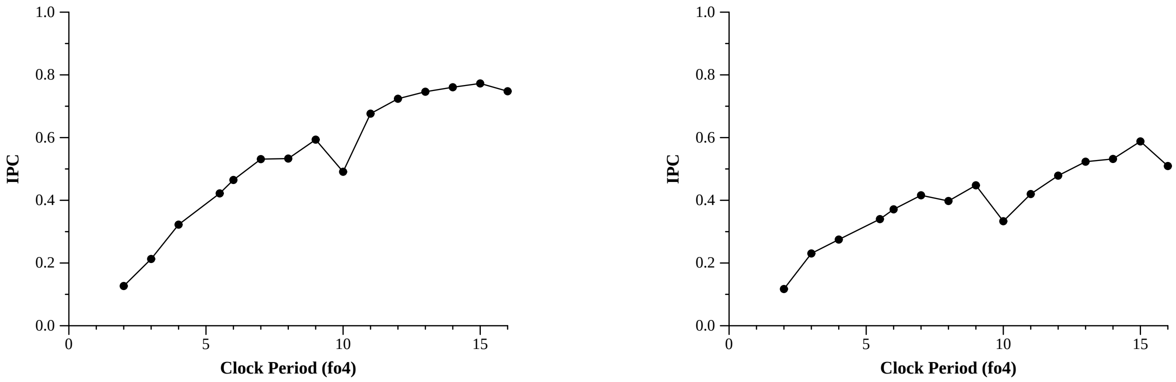
Integer Floating pt.

Figure 41: 70nm Technology Capacity Scaling IPC Trend



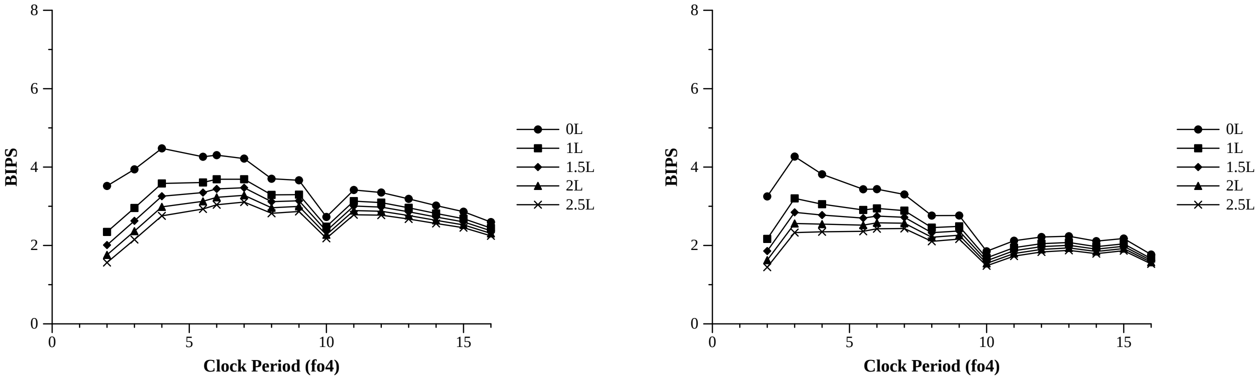
Integer Floating pt.

Figure 42: 70nm Technology Capacity Scaling Performance Trend



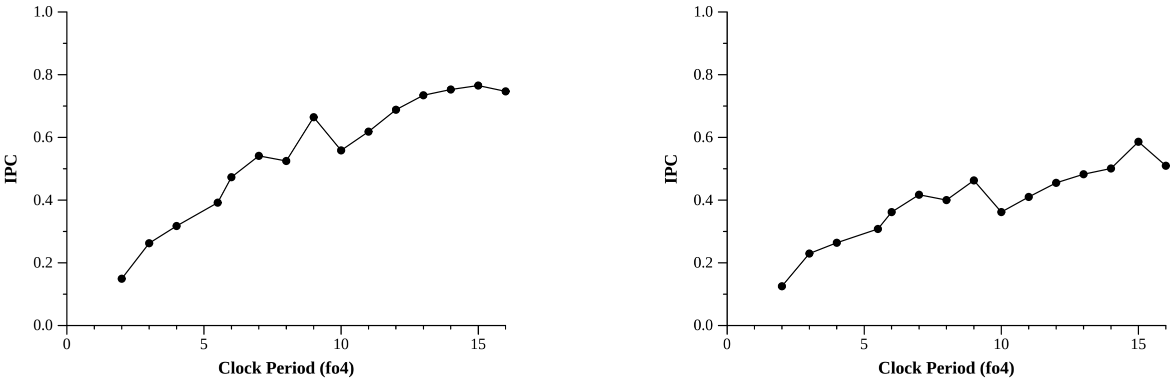
Integer Floating pt.

Figure 43: 50nm Technology Capacity Scaling IPC Trend



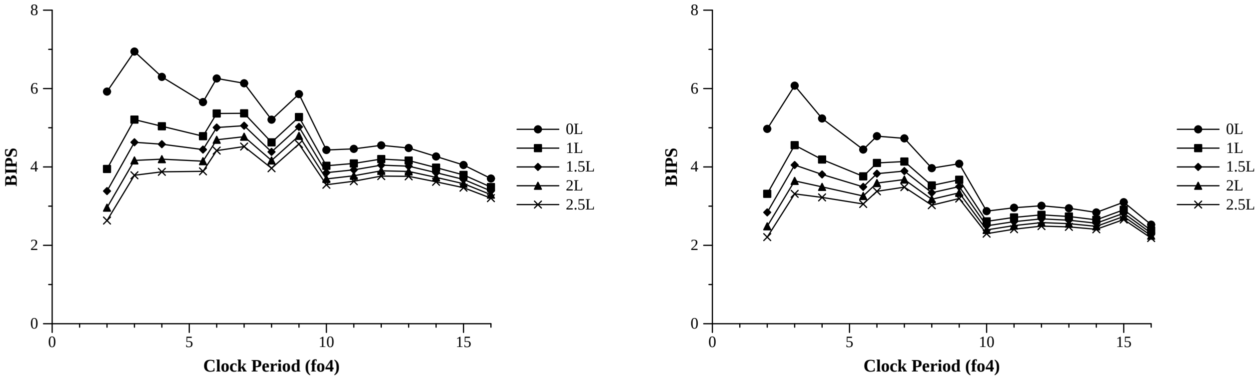
Integer Floating pt.

Figure 44: 50nm Technology Capacity Scaling Performance Trend



Integer Floating pt.

Figure 45: 35nm Technology Capacity Scaling IPC Trend



Integer Floating pt.

Figure 46: 35nm Technology Capacity Scaling Performance Trend