Appears in the Proceedings of the 29th International Symposium on Computer Architecture

The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays

M.S. Hrishikesh* Norman P. Jouppi+ Keith I. Farkas+
Doug Burger† Stephen W. Keckler† Premkishore Shivakumar†

*Department of Electrical and Computer Engineering
†Department of Computer Sciences
The University of Texas at Austin
http://www.cs.utexas.edu/users/cart

+Western Research Lab
Compaq Computer Corporation
http://research.compaq.com/wrl/

Abstract

Microprocessor clock frequency has improved by nearly 40% annually over the past decade. This improvement has been provided, in equal measure, by smaller technologies and deeper pipelines. From our study of the SPEC 2000 benchmarks, we find that for a high-performance architecture implemented in 100nm technology, the optimal clock period is approximately 8 fan-out-of-four (FO4) inverter delays for integer benchmarks, comprised of 6 FO4 of useful work and an overhead of about 2 FO4. The optimal clock period for floating-point benchmarks is 6FO4. We find these optimal points to be insensitive to latch and clock skew overheads. Our study indicates that further pipelining can at best improve performance of integer programs by a factor of 2 over current designs. At these high clock frequencies it will be difficult to design the instruction issue window to operate in a single cycle. Consequently, we propose and evaluate a high-frequency design called a segmented instruction window.

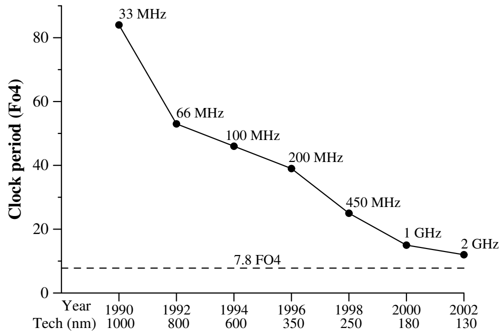


Figure 1: The year of introduction, clock frequency and fabrication technologies of the last seven generations of Intel processors. Logic levels are measured in fan-out-of-four delays (FO4). The broken line shows the optimal clock period for integer codes.

1 Introduction

Improvements in microprocessor performance have been sustained by increases in both instruction per cycle (IPC) and clock frequency. In recent years, increases in clock frequency have provided the bulk of the performance improvement. These increases have come from both technology scaling (faster gates) and deeper pipelining of designs (fewer gates per cycle). In this paper, we examine for how much further reducing the amount of logic per pipeline stage can improve performance. The results of this study have significant implications for performance scaling in the coming decade.

Figure 1 shows the clock periods of the Intel family of x86 processors on the y-axis. The x-axis shows the year of introduction and the feature size used to fabricate each processor. We computed the clock period by dividing the nominal frequency of the processor by the delay of one FO4 at the corresponding technology[1]. The graph shows that clock frequency has increased by approximately a factor of 60 over the past twelve years. During this period process technology has been scaled from 1000nm to 130nm, contributing an 8-fold improvement in clock frequency. The amount of logic per pipeline stage decreased from 84 to 12 FO4, contributing to the increase in clock frequency by a factor of 7. So far, both technology scaling and reduction in logic per stage have contributed roughly equally to improvements in clock frequency.

However, decreasing the amount of logic per pipeline stage increases pipeline depth, which in turn reduces IPC due to increased branch misprediction penalties and functional unit latencies. In addition, reducing the amount of logic per pipeline stage reduces the amount of useful work per cycle while not affecting overheads associated with latches, clock skew and jitter. Therefore, shorter pipeline stages cause the overhead to become a greater fraction of the clock period, which reduces the effective frequency gains.

Processor designs must balance clock frequency and IPC to achieve ideal performance. Previously, Kunkel and Smith examined this trade-off [9] by investigating the pipelining of a CRAY 1-S supercomputer to determine the number of lev-

[1] We measure the amount of logic per pipeline stage in terms of fan-out-of-four (FO4) – the delay of one inverter driving four copies of itself. Delays measured in FO4 are technology independent. The data points in Figure 1 were computed assuming that 1 FO4 roughly corresponds to 360 picoseconds times the transistor's drawn gate length in microns [6].

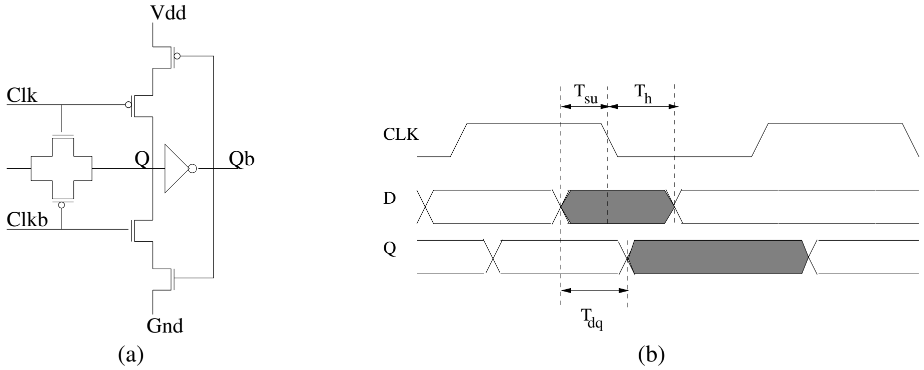


Figure 2: Circuit and timing diagrams of a basic pulse latch. The shaded area in Figure 2b indicates that the signal is valid.

els of logic per pipeline stage that provides maximum performance. They assumed the use of Earle latches between stages of the pipeline, which were representative of high-performance latches of that time. They concluded that, in the absence of latch and skew overheads, absolute performance increases as the pipeline is made deeper. But when the overhead is taken into account, performance increases up to a point beyond which increases in pipeline depth reduce performance. They found that maximum performance was obtained with 8 gate levels per stage for scalar code and with 4 gate levels per stage for vector code, which, using the equivalence we develop in Appendix A, is approximately 10.9 and 5.4 FO4 respectively.

In the first part of this paper, we re-examine Kunkel and Smith's work in a modern context to determine the optimal clock frequency for current-generation processors. Our study investigates a superscalar pipeline designed using CMOS transistors and VLSI technology, and assumes low-overhead pulse latches between pipeline stages. We show that maximum performance for integer benchmarks is achieved when the logic depth per pipeline stage corresponds to 7.8 FO4—6 FO4 of useful work and 1.8 FO4 of overhead. The dashed line in Figure 1 represents this optimal clock period. Note that the clock periods of current-generation processors already approach the optimal clock period. In the second portion of this paper, we identify a microarchitectural structure that will limit the scalability of the clock and propose methods to pipeline it at high frequencies. We propose a new design for the instruction issue window that divides it into sections. We show that although this method reduces the IPC of integer benchmarks by 11% and that of floating-point benchmarks by 5%, it allows significantly higher clock frequencies.

The remainder of this paper is organized in the following fashion. To determine the ideal clock frequency we first quantify latch overhead and present a detailed description of this methodology in Section 2. Section 3 describes the methodology to find the ideal clock frequency, which entails experiments with varied pipeline depths. We present the results of this study in Section 4. We examine specific microarchitectural structures in Section 5 and propose new designs that can be clocked at high frequencies. Section 6 discusses related work, and Section 7 summarizes our results and presents the conclusions of this study.

2 Estimating Overhead

The clock period of the processor is determined by the following equation

$$\phi = \phi_{logic} + \phi_{latch} + \phi_{skew} + \phi_{jitter} \qquad (1)$$

where ϕ is the clock period, ϕ_{logic} is useful work performed by logic circuits, ϕ_{latch} is latch overhead, ϕ_{skew} is clock skew overhead and ϕ_{jitter} is clock jitter overhead. In this section, we describe our methodology for estimating the overhead components, and the resulting values.

A pipelined machine requires data and control signals at each stage to be saved at the end of every cycle. In the subsequent clock cycle this stored information is used by the following stage. Therefore, a portion of each clock period, called *latch overhead*, is required by latches to sample and hold values. Latches may be either edge triggered or level sensitive. Edge-triggered latches reduce the possibility of *race through*, enabling simple pipeline designs, but typically incur higher latch overheads. Conversely, level-sensitive latches allow for design optimizations such as "slack-passing" and "time borrowing" [2], techniques that allow a slow stage in the pipeline to meet cycle time requirements by borrowing unused time from a neighboring, faster stage. In this paper we model a level-sensitive pulse latch, since it has low overhead and power consumption [4]. We use SPICE circuit simulations to quantify the latch overhead.

Figure 2a shows the circuit for a pulse latch consisting of a transmission gate followed by an inverter and a feed-back path. Data values are sampled and held by the latch as follows. During the period that the clock pulse is high, the transmission gate of the latch is on, and the output of the latch (Q) takes the same value as the input (D). When the clock signal changes to low, the transmission gate is turned off. However, the transistors along one of the two feedback paths turn on, completing the feedback loop. The inverter and the feedback loop retain the sampled data value until the following clock cycle.

The operation of a latch is governed by three parameters—setup time (T_{su}), hold time (T_h), and propagation delay

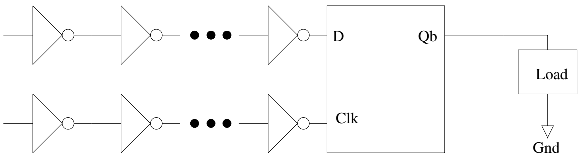


Figure 3: Simulation setup to find latch overhead. The clock and data signals are buffered by a series of six inverters and the output drives a similar latch with its transmission gate turned on.

(T_{dq}), as shown in Figure 2b. To determine latch overhead, we measured its parameters using the test circuit shown in Figure 3. The test circuit consists of a pulse latch with its output driving another similar pulse latch whose transmission gate is turned on. On-chip data and clock signals may travel through a number of gates before they terminate at a latch. To simulate the same effect, we buffer the clock and data inputs to the latch by a series of six inverters. The clock signal has a 50% duty cycle while the data signal is a simple step function. We simulated transistors at 100nm technology and performed experiments similar to those by Stojanović *et al.* [14], using the same P-transistor to N-transistor ratios. In our experiments, we moved the data signal progressively closer to the falling edge of the clock signal. Eventually when D changes very close to the falling edge of the Clk signal the latch fails to hold the correct value of D. Latch overhead is the smallest of the D-Q delays before this point of failure [14]. We estimated latch overhead to be 36ps (1 FO4) at 100nm technology. Since this delay is determined by the switching speed of transistors, which is expected to scale linearly with technology, its value in FO4 will remain constant at all technologies. Note that throughout this paper transistor feature sizes refer to the drawn gate length as opposed to the effective gate length.

In addition to latch overhead, clock skew and jitter also add to the total overhead of a clock period. A recent study by Kurd *et al.* [10] showed that, by partitioning the chip into multiple clock domains, clock skew can be reduced to less than 20ps and jitter to 35ps. They performed their studies at 180nm, which translates into 0.3 FO4 due to skew and 0.5 FO4 due to jitter. Many components of clock skew and jitter are dependent on the speed of the components, and those that are dependent on the transistor components should scale with technology. However, other terms, such as delay due to process variation, may scale differently, hence affecting the overall scalability. For simplicity we assume that clock skew and jitter will scale linearly with technology and therefore their values in FO4 will remain constant. Table 1 shows the values of the different overheads that we use to determine the clock frequency in Section 4. The sum of latch, clock skew and jitter overhead is equal to 1.8 FO4. We refer to this sum in the rest of the paper as $\phi_{overhead}$.

3 Methodology

To study the effect of deeper pipelining on performance, we varied the pipeline depth of a modern superscalar architecture similar to the Alpha 21264. This section describes our simulation framework and the methodology we used to perform this study.

Symbol	Definition	Overhead
ϕ_{latch}	Latch Overhead	1.0 FO4
ϕ_{skew}	Skew Overhead	0.3 FO4
ϕ_{jitter}	Jitter Overhead	0.5 FO4
$\phi_{overhead}$	Total	1.8 FO4

Table 1: Overheads due to latch, clock skew and jitter.

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

Table 2: SPEC 2000 benchmarks used in all simulation experiments. The benchmarks are further classified into vector and non-vector benchmarks.

3.1 Simulation Framework

We used a simulator developed by Desikan *et al.* that models both the low-level features of the Alpha 21264 processor [3] and the execution core in detail. This simulator has been validated to be within an accuracy of 20% of a Compaq DS-10L workstation. For our experiments, the base latency and capacities of on-chip structures matched those of the Alpha 21264, and the level-2 cache was configured to be 2MB. The capacities of the integer and floating-point register files alone were increased to 512 each, so that the performance of deep pipelines was not unduly constrained due to unavailability of registers. We modified the execution core of the simulator to permit the addition of more stages to different parts of the pipeline. The modifications allowed us to vary the pipeline depth of different parts of the processor pipeline, including the execution stage, the register read stage, the issue stage, and the commit stage.

Table 2 lists the benchmarks that we simulated for our experiments, which include integer and floating-point benchmarks taken from the SPEC 2000 suite. Some of the floating-point (FP) benchmarks operate on large matrices and exhibit strong vector-like behavior; we classify these benchmarks as vector floating-point benchmarks. When presenting simulation results, we show individual results for integer, vector FP, and non-vector FP benchmarks separately. All experiments skip the first 500 million instructions of each benchmark and simulate the next 500 million instructions.

		Branch	Rename	Issue	Register	Integer		FLoating Point			
ϕ_{logic} (FO4)	DL1	Predictor	Table	Window	File	Add	Mult	Add	Div	Sqrt	Mult
2	16	10	9	9	6	9	61	35	105	157	35
3	11	7	6	6	4	6	41	24	70	105	24
4	9	5	5	5	3	5	31	18	53	79	18
5	7	4	4	4	3	4	25	14	42	63	14
6	6	4	3	3	2	3	21	12	35	53	12
7	6	3	3	3	2	3	18	10	30	45	10
8	5	3	3	3	2	3	16	9	27	40	9
9	5	3	2	2	2	2	14	8	24	35	8
10	4	2	2	2	2	2	13	7	21	32	7
11	4	2	2	2	1	2	12	7	19	29	7
12	4	2	2	2	1	2	11	6	18	27	6
13	4	2	2	2	1	2	10	6	17	25	6
14	4	2	2	2	1	2	9	5	15	23	5
15	3	2	2	2	1	2	9	5	14	21	5
16	3	2	2	2	1	2	8	5	14	20	5
Alpha 21264 (17.4)	3	1	1	1	1	1	7	4	12	18	4

Table 3: Access latencies (clock cycles) of microarchitectural structures and integer and floating-point operations at 100nm technology (drawn gate length). The functional units are fully pipelined and new instructions can be assigned to them every cycle. The last row shows the latency of on-chip structures on the Alpha 21264 processor (180nm).

3.2 Microarchitectural Structures

We use Cacti 3.0 [12] to model on-chip microarchitectural structures and to estimate their access times. Cacti is an analytical tool originally developed by Jouppi and Wilton [7]. All major microarchitectural structures—data cache, register file, branch predictor, register rename table and instruction issue window—were modeled at 100nm technology and their capacities and configurations were chosen to match the corresponding structures in the Alpha 21264. We use the latencies of the structures obtained from Cacti to compute their access penalties (in cycles) at different clock frequencies.

3.3 Scaling Pipelines

We find the clock frequency that will provide maximum performance by simulating processor pipelines clocked at different frequencies. The clock period of the processor is determined by the following equation: $\phi = \phi_{logic} + \phi_{overhead}$. The overhead term is held constant at 1.8 FO4, as discussed in Section 2. We vary the clock frequency ($1/\phi$) by varying ϕ_{logic} from 2 FO4 to 16 FO4. The number of pipeline stages (clock cycles) required to access an on-chip structure, at each clock frequency, is determined by dividing the access time of the structure by the corresponding ϕ_{logic}. For example, if the access time of the level-1 cache at 100nm technology is 0.28ns (8 FO4), for a pipeline where ϕ_{logic} equals 2 FO4 (0.07ns), the cache can be accessed in 4 cycles.

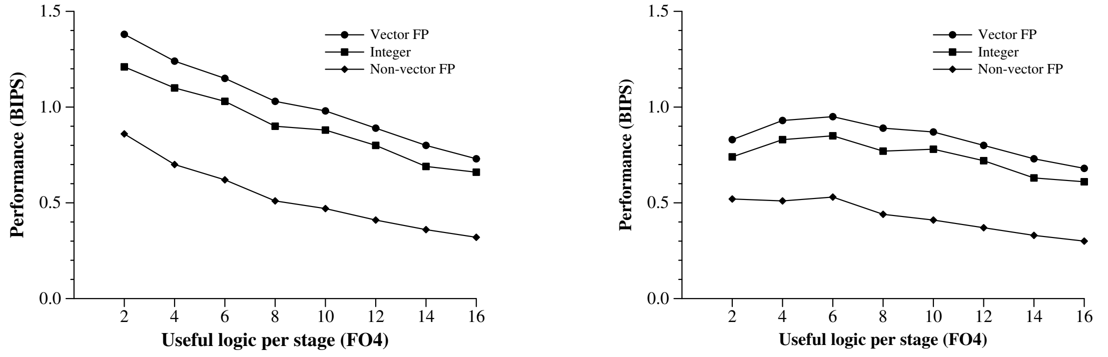
Though we use a 100nm technology in this study, the access latencies at other technologies in terms of the FO4 metric will remain largely unchanged at each corresponding clock frequency, since delays measured in this metric are technology independent. Table 3 shows the access latencies of structures at each ϕ_{logic}. These access latencies were determined by dividing the structure latencies (in pico seconds) obtained from the cacti model by the corresponding clock period. Table 3 also shows the latencies for various integer and floating-point operations at different clocks. To compute these latencies we determined ϕ_{logic} for the Alpha 21264 processor (800MHz, 180nm) by attributing 10% of its clock period to latch overhead (approximately 1.8 FO4). Using this ϕ_{logic} and the functional unit execution times of the Alpha 21264 (in cycles) we computed the execution latencies at various clock frequencies. In all our simulations, we assumed that results produced by the functional units can be fully bypassed to any stage between Issue and Execute.

In general, the access latencies of the structures increase as ϕ_{logic} is decreased. In certain cases the access latency remains unchanged despite a change in ϕ_{logic}. For example, the access latency of the register file is 0.39ns at 100nm technology. If ϕ_{logic} was 10 FO4 the access latency of the register file would be approximately 1.1 cycles. Conversely, if ϕ_{logic} was reduced to 6 FO4, the access latency would be 1.8 clock cycles. In both cases the access latency is rounded to 2 cycles.

By varying the processor pipeline as described above, we determine how deeply a high-performance design can be pipelined before overheads, due to latch, clock skew and jitter, and reduction in IPC, due to increased on-chip structure access latencies, begin to reduce performance.

4 Pipelined Architectures

In this section, we first vary the pipeline depth of an in-order issue processor to determine its optimal clock frequency. This in-order pipeline is similar to the Alpha 21264 pipeline except that it issues instructions in-order. It has seven stages—fetch, decode, issue, register read, execute, write back and commit. The issue stage of the processor is capable of issuing up to four instructions in each cycle. The execution stage consists of four



(a) (b)

Figure 4: In-order pipeline performance with and without latch overhead. Figure 4a shows that when there is no latch overhead performance improves as pipeline depth is increased. When latch and clock overheads are considered, maximum performance is obtained with 6 FO4 useful logic per stage (ϕ_{logic}), as shown in Figure 4b.

integer units and two floating-point units. All functional units are fully pipelined, so new instructions can be assigned to them at every clock cycle. We compare our results, from scaling the in-order issue processor, with the CRAY 1-S machine [9]. Our goal is to determine if either workloads or processor design technologies have changed the amount of useful logic per pipeline stage (ϕ_{logic}) that provides the best performance. We then perform similar experiments to find ϕ_{logic} that will provide maximum performance for a dynamically scheduled processor similar to the Alpha 21264. For our experiments in Section 4, we make the optimistic assumption that all microarchitectural components can be perfectly pipelined and be partitioned into an arbitrary number of stages.

4.1 In-order Issue Processors

Figure 4a shows the harmonic mean of the performance of SPEC 2000 benchmarks for an in-order pipeline, if there were no overheads associated with pipelining ($\phi_{overhead}$ = 0) and performance was inhibited by only the data and control dependencies in the benchmark. The x-axis in Figure 4a represents ϕ_{logic} and the y-axis shows performance in billions of instructions per second (BIPS). Performance was computed as a product of IPC and the clock frequency—equal to 1/ϕ_{logic}. The integer benchmarks have a lower overall performance compared to the vector floating-point (FP) benchmarks. The vector FP benchmarks are representative of scientific code that operate on large matrices and have more ILP than the integer benchmarks. Therefore, even though the execution core has just two floating-point units, the vector benchmarks out perform the integer benchmarks. The non-vector FP benchmarks represent scientific workloads of a different nature, such as numerical analysis and molecular dynamics. They have less ILP than the vector benchmarks, and consequently their performance is lower than both the integer and floating-point benchmarks. For all three sets of benchmarks, doubling the clock frequency does not double the performance. When ϕ_{logic} is reduced from 8 to 4 FO4, the ideal improvement in performance is 100%. However, for the integer benchmarks the improvement is only 18%. As ϕ_{logic} is further decreased, the improvement in performance deviates further from the ideal value.

Figure 4b shows performance of the in-order pipeline with $\phi_{overhead}$ set to 1.8 FO4. Unlike in Figure 4a, in this graph the clock frequency is determined by 1/(ϕ_{logic}+$\phi_{overhead}$). For example, at the point in the graph where ϕ_{logic} is equal to 8 FO4, the clock frequency is 1/(10 FO4). Observe that maximum performance is obtained when ϕ_{logic} corresponds to 6 FO4. In this experiment, when ϕ_{logic} is reduced from 10 to 6 FO4 the improvement in performance is only about 9% compared to a clock frequency improvement of 50%.

4.2 Comparison with the CRAY-1S

Kunkel and Smith [9] observed for the Cray-1S that maximum performance can be achieved with 8 gate levels of useful logic per stage for scalar benchmarks and 4 gate levels for vector benchmarks. If the Cray-1S were to be designed in CMOS logic today, the equivalent latency of one logic level would be about 1.36 FO4, as derived in Appendix A. For the Cray-1S computer this equivalent would place the optimal ϕ_{logic} at 10.9 FO4 for scalar and 5.4 FO4 for vector benchmarks. The optimal ϕ_{logic} for vector benchmarks has remained more or less unchanged, largely because the vector benchmarks have ample ILP, which is exploited sufficiently well by both the in-order superscalar pipeline and the Cray-1S. The optimal ϕ_{logic} for integer benchmarks has more than halved since the time of the Cray-1S processor, which means that a processor designed using modern techniques can be clocked at more than twice the frequency.

One reason for the decrease in the optimal ϕ_{logic} of integer benchmarks is that in modern pipelines average memory access latencies are lower, due to on-chip caches. The Alpha 21264 has a two-level cache hierarchy comprising of a 3-cycle, level-1 data cache and an off-chip unified level-2 cache. In the Cray-1S all loads and stores directly accessed a 12-cycle memory. Integer benchmarks have a large number of dependencies, and any instruction dependent on loads would stall the pipeline for 12 cycles. With performance bottlenecks in the memory system, increasing clock frequency by pipelining more deeply does not improve performance. We examined the

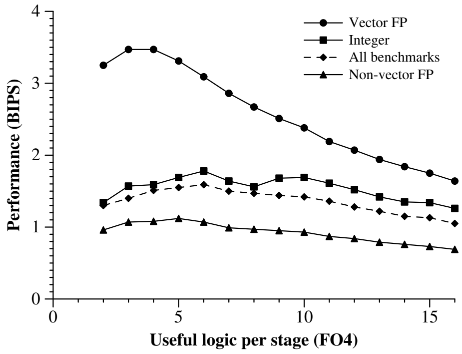


Figure 5: The harmonic mean of the performance of integer and floating point benchmarks, executing on an out-of-order pipeline, accounting for latch overhead, clock skew and jitter. For integer benchmarks best performance is obtained with 6 FO4 of useful logic per stage (ϕ_{logic}). For vector and non-vector floating-point benchmarks the optimal ϕ_{logic} is 4 FO4 and 5 FO4 respectively.

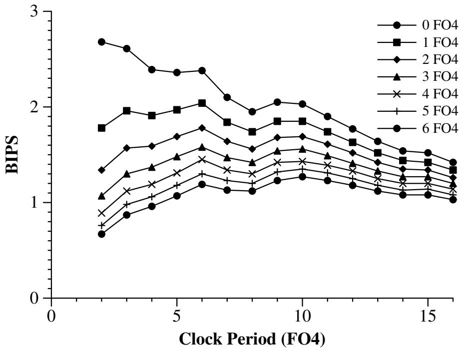


Figure 6: The harmonic mean of the performance of integer benchmarks, executing on an out-of-order pipeline for various values of $\phi_{overhead}$.

effect of scaling a superscalar, in-order pipeline with a memory system similar to the CRAY-1S (12 cycle access memory access, no caches) and found that the optimal ϕ_{logic} was 11 FO4 for integer benchmarks.

A second reason for the decrease in optimal ϕ_{logic} is the change in implementation technology. Kunkel and Smith assumed the processor was implemented using many chips at relatively small levels of integration, without binning of parts to reduce manufacturer's worst case delay variations. Consequently, they assumed overheads due to latches, data, and clock skew that were as much as 2.5 gate delays [9] (3.4 FO4). In contrast, modern VLSI microprocessors are comprised of circuits residing on the same die, so their process characteristics are more highly correlated than if they were from separate manufacturing runs fabricated perhaps months apart. Consequently, their speed variations and hence their relative skews are much smaller than in prior computer systems with lower levels of integration. Furthermore, the voltages and temperatures on one chip can be computed and taken into account at design time, also reducing the expected skews. These factors have reduced modern overhead to 1.8 FO4.

4.3 Dynamically Scheduled Processors

We performed similar experiments using a dynamically scheduled processor to find its optimal ϕ_{logic}. The processor configuration is similar to the Alpha 21264: 4-wide integer issue and 2-wide floating-point issue. We used a modified version of the simulator developed by Desikan *et al.* [3]. Figure 5 shows a plot of the performance of SPEC 2000 benchmarks when the pipeline depth of this processor is scaled. The performance shown in Figure 5 includes overheads represented by latch, clock skew and jitter ($\phi_{overhead}$). Figure 5 shows that overall performance of all three sets of benchmarks is significantly greater than for in-order pipelines. For a dynamically scheduled processor the optimal ϕ_{logic} for integer benchmarks is still 6 FO4. However, for vector and non-vector floating-point benchmarks the optimal ϕ_{logic} is 4 FO4 and 5 FO4 respectively. The dashed curve plots the harmonic mean of all three sets of benchmarks and shows the optimal ϕ_{logic} to be 6 FO4.

4.4 Sensitivity of ϕ_{logic} to $\phi_{overhead}$

Previous sections assumed that components of $\phi_{overhead}$, such as skew and jitter, would scale with technology and therefore overhead would remain constant. In this section, we examine performance sensitivity to $\phi_{overhead}$. Figure 6 shows a plot of the performance of integer SPEC 2000 benchmarks against ϕ_{logic} for different values of $\phi_{overhead}$. In general, if the pipeline depth were held constant (i.e. constant ϕ_{logic}), reducing the value of $\phi_{overhead}$ yields better performance. However, since the overhead is a greater fraction of their clock period, deeper pipelines benefit more from reducing $\phi_{overhead}$ than do shallow pipelines.

Interestingly, the optimal value of ϕ_{logic} is fairly insensitive to $\phi_{overhead}$. In section 2 we estimated $\phi_{overhead}$ to be 1.8 FO4. Figure 6 shows that for $\phi_{overhead}$ values between 1 and 5 FO4 maximum performance is still obtained at a ϕ_{logic} of 6 FO4.

4.5 Sensitivity of ϕ_{logic} to Structure Capacity

In previous sections we found the optimal ϕ_{logic} by varying the pipeline depth of a superscalar processor with structure capacities configured to match those of the Alpha 21264. However, at future clock frequencies the Alpha 21264 structure capacities may not yield maximum performance. For example, the data cache in the Alpha 21264 processor is 64KB and has a 3-cycle access latency. When the processor pipeline is scaled to higher frequencies, the cache access latency (in cycles) will increase and may unduly limit performance. In such a situation, a smaller capacity cache with a correspondingly lower access latency could provide better performance.

The capacity and latency of on-chip microarchitectural structures have a great influence on processor performance. These structure parameters are not independent and are closely tied together by technology and clock frequency. To identify the best capacity and corresponding latency for various on-chip structures, at each of our projected clock frequencies,

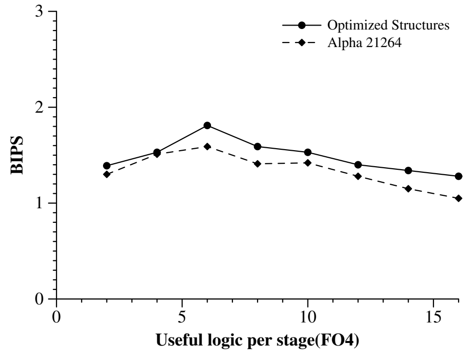


Figure 7: The harmonic mean of the performance of all SPEC 2000 benchmarks when optimal on-chip microarchitectural structure capacities are selected.

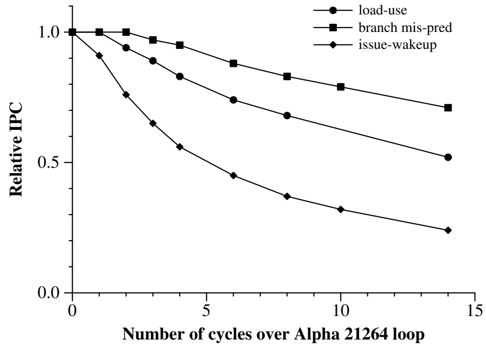


Figure 8: IPC sensitivity to critical loops in the data path. The x-axis of this graph shows the number of cycles the loop was extended over its length in the Alpha 21264 pipeline. The y-axis shows relative IPC.

we determined the sensitivity of IPC to the size and delay of each individual structure. We performed experiments independent of technology and clock frequency by varying the latency of each structure individually, while keeping its capacity unchanged. We measured how IPC changed with different latencies for each structure. We performed similar experiments to find the sensitivity of IPC to the capacity of each structure. We then used these two IPC sensitivity curves to determine, at each clock frequency, the capacity (and therefore latency) of every structure that will provide maximum performance. With that "best" configuration we simulated structures that were slightly larger/slower and smaller/faster to verify that the configuration was indeed optimal for that clock rate. At a clock with ϕ_{logic} of 6 FO4, the major on-chip structures have the following configuration: a level-1 data cache of 64KB, and 6 cycle access latency; a level-2 cache with 512KB, and 12 cycle access latency and a 64 entry instruction window with a 3 cycle latency. We assumed all on-chip structures were pipelined.

Figure 7 shows the performance of a pipeline with optimally configured microarchitectural structures plotting performance against ϕ_{logic}. This graph shows the harmonic mean of the performance (accounting for $\phi_{overhead}$) of all the SPEC 2000 benchmarks. The solid curve is the performance of a Alpha 21264 pipeline when the best size and latency is chosen for each structure at each clock speed. The dashed curve in the graph is the performance of the Alpha 21264 pipeline, similar to Figure 5. When structure capacities are optimized at each clock frequency, on the average, performance increases by approximately 14%. However, maximum performance is still obtained when ϕ_{logic} is 6 FO4.

4.6 Effect of Pipelining on IPC

Thus far we have examined scaling of the entire processor pipeline. In general, increasing overall pipeline depth of a processor decreases IPC because of dependencies within *critical loops* in the pipeline [2] [13]. These critical loops include issuing an instruction and waking its dependent instructions (issue-wake up), issuing a load instruction and obtaining the correct value (DL1 access time), and predicting a branch and resolving the correct execution path. For high performance it is important that these loops execute in the fewest cycles possible. When the processor pipeline depth is increased, the lengths of these critical loops are also increased, causing a decrease in IPC. In this section we quantify the performance effects of each of the above critical loops and in Section 5 we propose a technique to design the instruction window so that in most cases the issue-delay loop is 1 cycle.

To examine the impact of the length of critical loops on IPC, we scaled the length of each loop independently, keeping the access latencies of other structures to be the same as those of the Alpha 21264. Figure 8 shows the IPC sensitivity of the integer benchmarks to the branch misprediction penalty, the DL1 access time (load-use) and the issue-wake up loop. The x-axis of this graph shows the number of cycles the loop was extended over its length in the Alpha 21264 pipeline. The y-axis shows IPC relative to the baseline Alpha 21264 processor. IPC is most sensitive to the issue-wake up loop, followed by the load-use and branch misprediction penalty. The issue-wake up loop is most sensitive because it affects every instruction that is dependent on another instruction for its input values. The branch misprediction penalty is the least sensitive of the three critical loops because modern branch predictors have reasonably high accuracies and the misprediction penalty is paid infrequently. The floating-point benchmarks showed similar trends with regard to their sensitivity to critical loops. However, overall they were less sensitive to all three loops than integer benchmarks.

The results from Figure 8 show that the ability to execute dependent instructions back to back is essential to performance. Similar obsevations have been made in other studies [13] [1].

5 A Segmented Instruction Window Design

In modern superscalar pipelines, the instruction issue window is a critical component, and a naive pipelining strategy that prevents dependent instructions from being issued back to back would unduly limit performance. In this section we propose a method to pipeline the instruction issue window to enable clocking it at high frequencies.

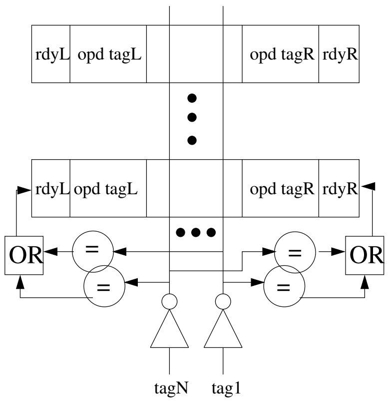


Figure 9: A high-level representation of the instruction window.

To issue new instructions every cycle, the instructions in the instruction issue window are examined to determine which ones can be issued (wake up). The instruction selection logic then decides which of the woken instructions can be selected for issue. Stark *et al.* showed that pipelining the instruction window, but sacrificing the ability to execute dependent instructions in consecutive cycles, can degrade performance by up to 27% compared to an ideal machine [13].

Figure 9 shows a high-level representation of an instruction window. Every cycle that a result is produced, the tag associated with the result (*destination tag*) is broadcast to all entries in the instruction window. Each instruction entry in the window compares the destination tag with the tags of its source operands (*source tags*). If the tags match, the corresponding source operand for the matching instruction entry is marked as ready. A separate logic block (not shown in the figure) selects instructions to issue from the pool of ready instructions. At every cycle, instructions in any location in the window can be woken up and selected for issue. In the following cycle, empty slots in the window, from instructions issued in the previous cycle, are reclaimed and up to four new instructions can be written into the window. In this section, we first describe and evaluate a method to pipeline instruction wake-up and then evaluate a technique to pipeline instruction selection logic.

5.1 Pipelining Instruction Wakeup

Palacharla *et al.* [11] argued that three components constitute the delay to wake up instructions: the delay to broadcast the tags, the delay to perform tag comparisons, and the delay to OR the individual match lines to produce the ready signal. Their studies show that the delay to broadcast the tags will be a significant component of the overall delay at feature sizes of 180nm and below. To reduce the tag broadcast latency, we propose organizing the instruction window into stages, as shown in Figure 10. Each stage consists of a fixed number of instruction entries and consecutive stages are separated by latches. A set of destination tags are broadcast to only one stage during a cycle. The latches between stages hold these tags so that they can be broadcast to the next stage in the following cycle. For example, if an issue window capable of holding 32 instructions is divided into two stages of 16 entries each, a set of tags are broadcast to the first stage in the first cycle. In the second cycle the same set of tags are broadcast to the next stage, while a new set of tags are broadcast to the first 16 entries. At every cycle, the entire instruction window can potentially be woken up by a different set of destination tags at each stage. Since each tag is broadcast across only a small part of the window every cycle, this instruction window can be clocked at high frequencies. However, the tags of results produced in a cycle can wake up instructions only in the first stage of the window during that cycle. Therefore, dependent instructions can be issued back to back only if they are in the first stage of the window.

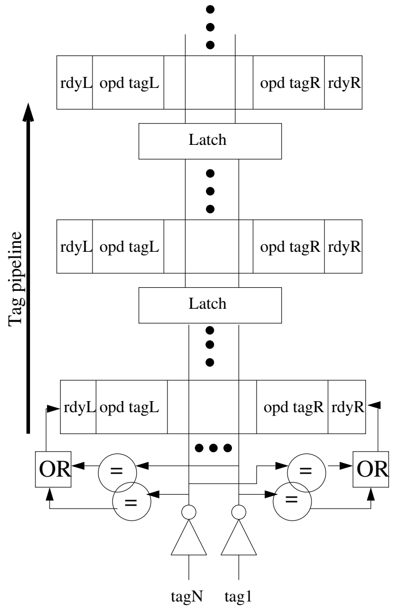


Figure 10: A segmented instruction window wherein the tags are broadcast to one stage of the instruction window at a time. We also assume that instructions can be selected from the entire window.

We evaluated the effect of pipelining the instruction window on IPC by varying the pipeline depth of a 32-entry instruction window from 1 to 10 stages. Figure 11 shows the results from our experiments when the number of stages of the window is varied from 1 to 10. Note that the x-axis on this graph is the pipeline depth of the wake-up logic. The plot shows that IPC of integer and vector benchmarks remain unchanged until the window is pipelined to a depth of 4 stages. The overall decrease in IPC of the integer benchmarks when the pipeline depth of the window is increased from 1 to 10 stages is approximately 11%. The floating-point benchmarks show a decrease of 5% for the same increase in pipeline depth. Note that this decrease is small compared to that of naive pipelining, which prevents dependent instructions from issuing consecutively.

5.2 Pipelining Instruction Select

In addition to wake-up logic, the selection logic determines the latency of the instruction issue pipeline stage. In a conventional processor, the select logic examines the entire in-

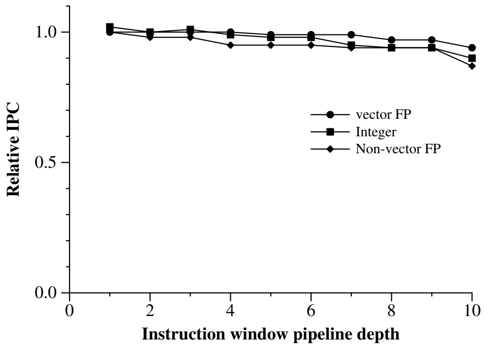


Figure 11: IPC sensitivity to instruction window pipeline depth, assuming all entries in the window can be considered for selection.

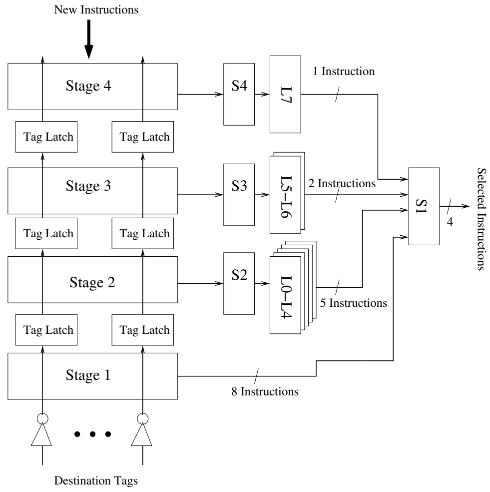


Figure 12: A 32-entry instruction window partitioned into four stages with a selection logic fan-in of 16 instructions

struction window to select instructions for issue. We propose to decrease the latency of the selection logic by reducing its fan-in. As with the instruction wake-up, the instruction window is partitioned into stages as shown in Figure 12. The selection logic is partitioned into two operations: *preselection* and *selection*. A preselection logic block is associated with all stages of the instruction window (S2-S4) except the first one. Each of these logic blocks examines all instructions in its stage and picks one or more instructions to be considered for selection. A selection logic block (S1) selects instructions for issue from among all ready instructions in the first section and the instructions selected by S2-S4. Each logic block in this partitioned selection scheme examines fewer instructions compared to the selection logic in conventional processors and can therefore operate with a lower latency.

Although several configurations of instruction window and selection logic are possible depending on the instruction window capacity, pipeline depth, and selection fan-in, in this study we evaluate the specific implementation shown in Figure 12. This instruction window consists of 32-entries partitioned into four stages and is configured so that the fan-in of S1 is 16. Since each stage in the window contains 8 instructions and all the instructions in Stage 1 are considered for selection by S1, up to 8 instructions may be pre-selected. Older instructions in the instruction window are considered to be more critical than younger ones. Therefore the preselection blocks are organized so that the stages that contain the older instructions have a greater share of the pre-selected instructions. The logic blocks S2, S3, and S4 pre-select instructions from the second, third, and fourth stage of the window respectively. Each select logic block can select from any instruction within its stage that is ready. However, S2 can pre-select a maximum of five instructions, S3 a maximum of 2 and S4 can pre-select only one instruction. The selection process works in the following manner. At every clock cycle, preselection logic blocks S2-S4 pick from ready instructions in their stage. The instructions pre-selected by these blocks are stored in latches L1-L7 at the end of the cycle. In the second cycle the select logic block S1 selects 4 instructions from among all the ready instructions in Stage 1 and those in L1-L7 to be issued to functional units.

With an instruction window and selection logic as described above, the IPC of integer benchmarks was reduced by only 4% compared to a processor with a single cycle, 32-entry, non-pipelined instruction window and select fan-in of 32. The IPC of floating-point benchmarks was reduced by only 1%. The rather small impact of pipelining the instruction window on IPC is not surprising. The floating-point benchmarks have fewer dependences in their instruction streams than integer codes, and therefore remain unaffected by the increased wake up 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 thus enables them to be woken up with less delay. This segmented window design will be capable of operating at greater frequencies than conventional designs at the cost of minimal degradation in IPC.

6 Related Work

Aside from the work of Kunkel and Smith [9] discussed in Section 4, the most relevant related work explores alternate designs for improving instruction window latencies. Stark *et al.* [13] proposed a technique to pipeline instruction wake up and select logic. In their technique, instructions are woken up "speculatively" when their *grandparents* are issued. The rationale behind this technique is that if an instruction's grandparents' tags are broadcast during the current cycle its *parents* will probably be issued the same cycle. While speculatively woken instructions can be selected, they cannot be issued until their parents have been issued. Although this technique reduces the IPC of the processor compared to a conventional 1-cycle instruction window, it enables the instruction window to function at a higher clock frequency.

Brown *et al.* [1] proposed a method to move selection logic off the critical path. In this method, wake-up and select are partitioned into two separate stages. In the first stage (wake-up) instructions in the window are woken up by producer tags, similar to a regular instruction window. All instructions that wake up speculate they will be selected for issue in the following cycle and assert their "available" signals. In the next cycle, the result tags of these instructions are broadcast to the window, as though all of them have been issued. However, the selection logic selects only a limited number of instructions from those that asserted their "available" signal. Instructions that do not get selected (*collision victims*) and any dependents that are woken up before they can be issued (*pileup victims*) are detected and re-scheduled. The authors show that this technique has an IPC within 3% of a machine with single-cycle scheduling logic.

7 Conclusion

In this paper, we measured the effects of varying clock frequency on the performance of a superscalar pipeline. We determined the amount of useful logic per stage (ϕ_{logic}) that will provide the best performance is approximately 6 FO4 inverter delays for integer benchmarks. If ϕ_{logic} is reduced below 6 FO4 the improvement in clock frequency cannot compensate for the decrease in IPC. Conversely, if ϕ_{logic} is increased to more than 6 FO4 the improvement in IPC is not enough to counteract the loss in performance resulting from a lower clock frequency. For vector floating-point benchmarks the optimal ϕ_{logic} was at 4 FO4. The clock period (ϕ_{logic} + $\phi_{overhead}$) at the optimal point is 7.8 FO4 for integer benchmarks, corresponding to a frequency of 3.6GHz at 100nm technology. For vector floating-point benchmarks the optimal clock period is 5.8 FO4 which corresponds to 4.8GHz at 100nm technology.

These optimal clock frequencies can be achieved only if on-chip microarchitectural structures can be pipelined to operate at high frequencies. We identified the instruction issue window as a critical structure, which will be difficult to scale to those frequencies. We propose a segmented instruction window design that will allow it to be pipelined to four stages without significant decrease in IPC. Scaling the pipeline depth of the window to 10 stages only decreases the IPC of SPEC 2000 integer benchmarks by 11% and floating-point benchmarks by 5%.

Although this study uses the parameters of a 100nm technology, our use of the technology-independent FO4 metric will permit our results to be translated to other technologies. We assume that 1 FO4 corresponds to 360 picoseconds times the transistor's drawn gate length. But, for highly tuned processes, such as the Intel 0.13-μm process, the drawn gate length and effective gate length may differ substantially [16]. However, our estimate of the optimal pipeline depth remains unchanged regardless of the exact value assigned to a FO4 delay though the actual cycle time will depend on the operating conditions and process technology specifications.

While we did not consider the effects of slower wires, they should not affect this study, which uses a fixed microarchitecture. To first order, wire delays remain constant as a fixed design is scaled to smaller feature sizes [15]. Although wire resistance increases, wire lengths decrease, thus preserving the absolute wire delay across technologies. However, long wires that arise as design complexity increases can have a substantial impact on the pipelining of the microarchitecture. For example, the high clock rate target of the Intel Pentium IV forced the designers to dedicate two pipeline stages just for data transportation [5]. We will examine the effects of wire delays on our pipeline models and optimal clock rate selection in future work.

Microprocessor performance has improved at about 55% per year for the last three decades, with much of the gains resulting from higher clock frequencies, due to process technology and deeper pipelines. However, our results show that pipelining can contribute *at most* another factor of two to clock rate improvements. Subsequently, in the best case, clock rates will increase at the rate of feature size scaling, which is projected to be 12-20% per year. Any additional performance improvements must come from increases in concurrency, whether they be instruction-level parallelism, thread-level parallelism, or a combination of the two. If the goal is to maintain historical performance growth rates, concurrency must start increasing at 33% per year and sustain a total of 50 IPC within the next 15 years. While this goal presents tremendous challenges, particularly in the face of increasing on-chip communication delays, rich opportunities for novel architectures lie ahead.

Acknowledgments

We thank the anonymous referees for their comments and suggestions. Thanks also to Mary Brown and members of the CART research group for their valuable suggestions. This research is supported by the Defense Advanced Research Projects Agency under contract F33615-01-C-1892, NSF CAREER grants CCR-9985109 and CCR-9984336, two IBM University Partnership awards, and a grant from the Intel Research Council.

References

[1] Mary D Brown, Jared Stark, and Yale N. Patt. Select-free instruction scheduling logic. In *Proceedings of the 34rd International Symposium on Microarchitecture*, pages 204–213, December 2001.

[2] Anantha Chandrakasan, William J. Bowhill, and Frank Fox ,editors. *Design of High-Performance Microprocessor Circuits*. IEEE Press, Piscataway, NJ, 2001.

[3] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 266–277, July 2001.

[4] Seongmoo Heo, Ronny Krashinsky, and Krste Asanović. Activity-sensitive flip-flop and latch selection for reduced energy. In *Conference on Advanced Research in VLSI*, pages 59–74, March 2001.

[5] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel. The microarchitecture of the pentium 4 processor. *Intel Technology Journal*, 1, February 2001.

[6] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[7] Norman P. Jouppi and Steven J. E. Wilton. An enhanced access and cycle time model for on-chip caches. Technical Report 93.5, Compaq Computer Corporation, July 1994.

[8] James S. Kolodzey. Cray-1 computer technology. *IEEE Transactions on Components, Hybrids, and Manufacturing Technology CHMT-4(2)*, 4(2):181–187, March 1981.

[9] Steven R. Kunkel and James E. Smith. Optimal pipelining in supercomputers. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 404–411, June 1986.

[10] Nasser A. Kurd, Javed S. Barkatullah, Rommel O. Dizon, Thomas D. Fletcher, and Paul D. Madland. Multi-GHz clocking scheme for Intel Pentium 4 microprocessor. In *Proceedings of the International Solid-state Circuits Conference*, pages 404–405, February 2001.

[11] Subbarao Palacharla, Norman P. Jouppi, and J.E. Smith. Complexity-effective superscalar processors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 206–218, June 1997.

[12] Premkishore Shivakumar and Norman P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical Report 2001/2, Compaq Computer Corporation, August 2001.

[13] Jared Stark, Mary D. Brown, and Yale N. Patt. On pipelining dynamic instruction scheduling logic. In *Proceedings of the 33rd International Symposium on Microarchitecture*, pages 57–66, December 2000.

[14] Vladimir Stojanović and Vojin G. Oklobdžija. Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems. *IEEE Journal of Solid-state Circuits*, 34(4):536–548, April 1999.

[15] Dennis Sylvester and Kurt Keutzer. Rethinking deep-submicron circuit design. *IEEE Computer*, 32(11):25–33, November 1999.

[16] S. Tyagi, M. Alavi, R. Bigwood, T. Bramblett, J. Brandenburg, W. Chen, B. Crew, M. Hussein, P. Jacob, C. Kenyon, C. Lo, B. Mcintyre, Z. Ma, P. Moon, P. Nguyen, L. Rumaner, R. Schweinfurth, S. Sivakumar, M. Stettler, S. Thompson, B. Tufts, J. Xu, S. Yang, and M. Bohr. A 130nm generation logic technology featuring 70nm transistors, dual vt transistors and 6 layers of cu interconnects. In *Proceedings of International Electronic Devices Meeting*, December 2000.

A ECL gate equivalent in FO4

The Cray-1S processor was designed in an ECL technology, using four and five input NAND gates [8] with eight gate levels at every pipeline stage. Because of its implementation from discrete ECL devices and the design of transmission lines for the wires connecting the chips, the latency of one wire and one gate delay were roughly equivalent. Furthermore, because of the transmission line effect of the wires, additional gate fanout loading can largely be ignored. The result is that the latency of a pipeline stage was approximately equal to the delay of 16 logic gates. Our CMOS equivalent of one Cray ECL gate circuit consists of a 4-input NAND driving a 5-input NAND, where the first accounts for gate delay and the second accounts for the wire delay. Figure 13 shows the test circuit we used to perform this measurement. SPICE simulations show that this one ECL gate equivalent has a latency equal to 1.36 FO4.

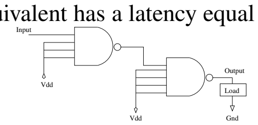


Figure 13: Circuit to measure the delay of CRAY-1S gates in terms of FO4.