Report Number: WUCS-94-32

1994-01-01

Pipelined and Superscalar Architectures in Clocked and Asynchronous Environments

Mark A. Franklin and Tienyo Pan

In this paper, a set of simple, general, yet practical performance models for RISC architectures are developed. These models apply to a wide range of systems that include both pipelined and superscalar systems operating in either clocked or asynchronous environments. The models permit quantitative evaluation of various design choices (e.g., the number of pipelines in the system, the pipeline depth, and the choice between clocked and asynchronous methodologies) as functions of technology parameters, environmental operating parameters, and pipeline function characteristics. Design curves are presented indicating optimal pipeline depth and number of pipelines to employ under various conditions.

... Read complete abstract on page 2.

Pipelined and Superscalar Architectures in Clocked and Asynchronous Environments

Mark A. Franklin and Tienyo Pan

Complete Abstract:

In this paper, a set of simple, general, yet practical performance models for RISC architectures are developed. These models apply to a wide range of systems that include both pipelined and superscalar systems operating in either clocked or asynchronous environments. The models permit quantitative evaluation of various design choices (e.g., the number of pipelines in the system, the pipeline depth, and the choice between clocked and asynchronous methodologies) as functions of technology parameters, environmental operating parameters, and pipeline function characteristics. Design curves are presented indicating optimal pipeline depth and number of pipelines to employ under various conditions.

Pipelined and Superscalar Architectures in Clocked and Asynchronous Environments

Mark A. Franklin and Tienyo Pan

WUCS-94-32

November 1994

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

This research has been sponsored in part by funding from the National Science Foundation Grant CCR-9021041 and ARPA Contract DABT-93-C0057.

PIPELINED and SUPERSCALAR ARCHITECTURES

in

CLOCKED and ASYNCHRONOUS ENVIRONMENTS [*]

Mark A. Franklin and Tienyo Pan
Computer and Communications Research Center
Campus Box 1115
Washington University
St. Louis, Missouri 63130-4899

jbf@wuccrc.wustl.edu pan@wuccrc.wustl.edu
(314) 935-6107 (314) 935-8562
FAX: (314)935-7302

Key Words
pipelined, superscalar, clocked, asynchronous, instruction-level parallelism

Abstract

In this paper, a set of simple, general, yet practical performance models for RISC architectures are developed. These models apply to a wide range of systems that include both pipelined and superscalar systems operating in either clocked or asynchronous environments. The models permit quantitative evaluation of various design choices (e.g., the number of pipelines in the system, the pipeline depth, and the choice between clocked and asynchronous methodologies) as functions of technology parameters, environmental operating parameters, and pipeline function characteristics. Design curves are presented indicating optimal pipeline depth and number of pipelines to employ under various conditions.

[*] This research has been sponsored in part by funding from the National Science Foundation Grant CCR-9021041 and ARPA Contract DABT-93-C0057.

Pipelined and Superscalar Architectures in Clocked and Asynchronous Environments[1]

Mark A. Franklin and Tienyo Pan
Computer and Communication Research Center
Washington University
St. Louis, MO 63130

August 1, 1994

1 Introduction

Effective instruction pipeline design is central to achieving high microprocessor performance. This paper examines the design of instruction pipelines based on two alternative control and synchronization methodologies: clocked (synchronous) and self-timed (asynchronous). Our goal is to develop a set of general performance models which will permit the designer to determine the optimal pipeline design for a given set of technology, environmental and design operating ranges, instruction set execution assumptions, and the selected synchronization methodology.

Typical early RISC machines consisted of a single, five-stage pipeline [11]. In more advanced machines, performance has been improved by either increasing the pipeline depth (i.e., *superpipelined*) or employing multiple pipelines (i.e., *superscalar*). Superpipelined techniques divide each instruction into finer segments thus reducing the cycle time. Superscalar techniques, on the other hand, issue several instructions into parallel pipelines and thus increase the average number of instructions being processed per cycle. In this paper, instruction throughput is modeled as a function of the number of pipelines (P) and the number of stages in each pipeline (S). Once other specific technology and design parameters are chosen, the best combination of S and P that yields the highest instruction throughput can be determined for each of the control/synchronization methods.

[1]This research has been sponsored in part by funding from NSF Grant CCR-9021041 and ARPA Contract DABT-93-C0057

Most commercial machines are currently clocked, however, asynchronous design has attracted more attention in recent years due to several reasons. First, as clocked systems increase in size, clock skew increases and inevitably limits clock rate. The equivalent delay is not present in asynchronous systems (although other delays are present) [16]. Second, hierarchical, modular design techniques are generally ill suited to handling global design constraints such as clock distribution. Asynchronous techniques permit one to focus on the functional and logical sequencing aspects of design and not on such global issues thus making the design task more manageable. Third, there is some indication that using the CMOS technology, asynchronous systems require less power than clocked systems since power consumption is not directly related to clock frequency (i.e., unused modules in asynchronous systems do not require charging/discharging on each clock cycle [9]). Finally, the clock period in a clocked system is generally based on the worst case time for component functional units. In asynchronous systems, however, average function delays may govern overall throughput rates thus potentially resulting in higher performance.

Due to the above reasons, although asynchronous methodology currently have disadvantages in chip area and design complexity, it is a design alternative worthy of consideration. At least one company is currently marketing an asynchronous RISC processor [8] while it appears that several other companies are investigating the design of asynchronous versions of popular microprocessor designs [15].

In this research, a set of simple, general, yet practical performance models for pipelined systems are developed. These models apply to a wide range of systems that include both pipelined and superscalar systems operating in either clocked or asynchronous environments. Applying the models (with appropriate parameters values), computer designers have the opportunity to estimate the optimal pipeline design before detailed simulations are performed. Computer architects can use them to evaluate the tradeoffs between superpipelined and superscalar architectures. In addition, when a design is to be switched from clocked to asynchronous, these models will be helpful in estimating potential performance gains (or losses).

Section 2 of this paper includes a background discussion of instruction pipeline throughput and a review of previous research in this area. In Section 3 pipeline performance, in terms of IPC (Instructions Per Cycle), is modeled as a function of pipeline depth and the number of pipelines present. Cycle times associated with the clocked and asynchronous design methodologies are determined in Sections 4 and 5 respectively. In Section 6 the models are then applied to a given design environment using typical parameter values. Resulting design curves are shown and compared. Section 7 presents conclusions and suggestions for future research in this area.

2 Background

Instruction throughput of a computer system can be expressed as:

$$Throughput = IPC \,/\, t_{cycle} \tag{1}$$

IPC is the average number of Instructions completed Per Cycle. In a clocked system, a cycle is well defined and corresponds to the *clock cycle time* which is generally a fixed time period designated as t_{cycle}. In an asynchronous system, however, a cycle is of variable length determined by a host of factors. For example, the time it takes for a given pipeline stage to complete its operation will depend on the instruction type being executed at that stage and the operand values assumed [7]. In addition, pipeline stages generally have different completion times. This will be defined more precisely later, for now however, t_{cycle} will be taken to be an *average* value of a random variable and the IPC for this asynchronous case is the number of instructions completed on average in this t_{cycle}.

Assume that with a non-pipelined processor the $IPC = 1$ and for this case take $t_{cycle} = 1$. Then, with an *ideal* S-stage pipeline $t_{cycle} = 1/S$ and an S-fold increase in throughput is obtained. For an *ideal* (i.e., no overheads or delays due to stalling etc.) superscalar machine that consists of P pipelines, its IPC would increase P times. If each pipeline in the superscalar system has S stages, performance of this ideal superscalar machine would then be $P * S$ times better than its non-pipelined, non-superscalar counterpart. However, instruction pipelines generally do not achieve the ideal performance (i.e., are underutilized) for several reasons:

- **Unbalanced workload:** Partitioning instruction execution into stages of equal time duration is difficult. Typically the slowest stage limits pipeline throughput and leads to under-utilization of some stages.
- **Non-Ideal IPC:** Control (branching) and data dependencies may cause flushing or stalling of a pipeline, thus reducing pipeline utilization. Also, in superscalar systems, class conflicts in a group of instructions may cause idle cycles in some pipelines.
- **Synchronization overhead:** Associated with each stage computation completion and transfer of results to the next stage, there is a synchronization overhead. This includes delays attributable to clock skew, latching time, intentional padding (for single phase clocked systems), and completion-detection and handshaking delay (for asynchronous systems). These delays have the effect of further reducing pipeline utilization and thus overall throughput.

If the pipeline(s) of a system are fully utilized (i.e., an ideal system), throughput maximization would be achieved simply by maximizing the number of stages (S) and/or the number of pipelines (P). As those numbers increase, however, many of the above problems are exacerbated and system performance suffers. Thus, for each of the synchronization methodologies, technology, and instruction execution parameters, there is some optimal design. One problem, therefore, is to determine how performance varies with S and P for each of the synchronization methodologies. The two methodologies can then be compared based on their optimal performance.

There have been a number of studies which have considered the timing constraints and circuit level performance associated with pipeline design [3, 10, 5]. Based on their work, Kunkel and Smith [13] have studied the optimal pipeline design in the context of CRAY-1S architecture. The first 14 Lawrence Livermore Loops are selected in their research to help determine the IPC, and the clock cycle time t_{cycle} is modeled by a set of timing constraint equations for single-phase clocks. Due to synchronization overheads and the non-ideal nature of the pipeline, it has been shown that eight to ten gate levels per pipeline segment lead to optimal overall performance. Deeper pipelines (fewer gate-levels per stage) may cause performance degradation. This research was extended and generalized by Dubey and Flynn [4]. In their work, IPC is expressed as a function of pipeline depth, so instruction throughput can be written in a single variable equation. Once parameters of this equation are selected by examining target machines and applications, the optimal pipeline depth can be calculated. Franklin and Pan [6] further extended this research to include analysis of asynchronous design methodologies. Performance equations of clocked and asynchronous (single) pipelines have been developed so that, for a given set of parameter and application values, optimal pipeline depths can be obtained for each of the two strategies and a comparison can then be made.

The work presented in this paper significantly extends the research cited above in the following ways.

- Dubey and Flynn [4] have modeled IPC as a second-order equation, $IPC = IPC_{max} - rS^2 - vS$, where S is the pipeline depth, and r and v are coefficients that can be empirically determined. While their model successfully describes the decrease in IPC as pipeline depth increases, it becomes negative and fails when S is greater than a certain value (depending on r and v). Furthermore it does not take into account superscalar systems. In this paper, a new IPC model is presented which corrects these deficiencies. The model is validated by realistic cases in Section 6.
- Jouppi and Wall [12] have compared instruction-level parallelism for superscalar and superpipelined machines. In their paper, a *base machine* is defined as a four-stage pipeline. The superscalar architectures discussed are all based upon this base machine which has a fixed pipeline depth (i.e., $S = 4$). The models presented in this paper cover any combination of S and P.
- This is the first study we know of that examines the performance of asynchronous superscalar machines.

3 IPC Model Development

Pipelined and superscalar architectures considered in this paper have a basic form shown in Figure 1. The number of pipelines in a system is denoted as P, and the number of stages in each pipeline is denoted as S. The example given in this figure, therefore, can be called a $(P, S) = (3, 4)$ superscalar machine. Architectures with $P = 1$ *and* $S > 1$ are usually called pipelined (or superpipelined) machines. Each pipeline stage (shown as a box in Figure 1) can hold either an instruction or a *STall*. A stage is utilized when it is filled by an instruction and it is wasted or unutilized when filled by a stall. The IPC can be expressed as:

$$IPC = P\,/\,(1 + A) \qquad (2)$$

where A is defined as the average number of stalls generated by each instruction. Under ideal conditions, there are no stalls in the system, and thus IPC equals the number of pipelines in the

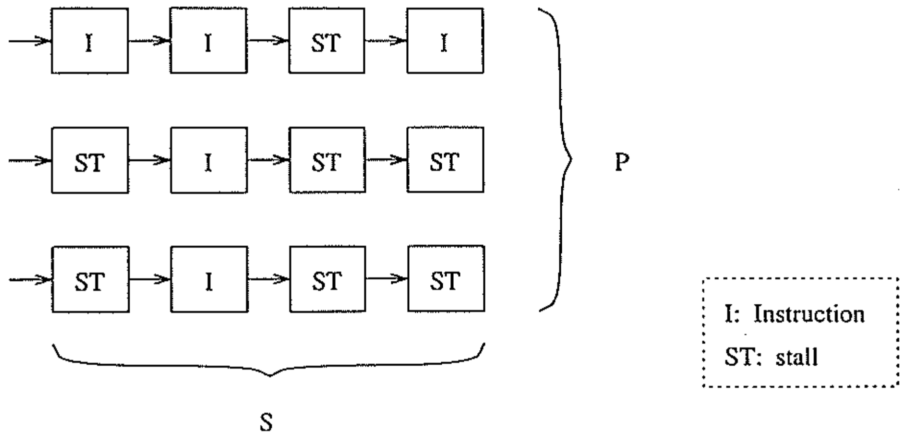

Figure 1: An $(P, S) = (3, 4)$ superscalar machine

system. For example, if Figure 1 represents the average contents of the pipelines, then with 7 stalls generated by 5 instructions, $A = 7/5$. In this case $IPC = 5/12 * P$ and 5/12 corresponds to the overall utilization (the number of instructions over the total number of stages) of the three pipelines.

Under the assumption that out-of-order instruction execution is not permitted, stalls exist in instruction pipelines for the following principal reasons:

- **Multiple cycle operations:** Some operations cannot be completed in one cycle. For example, a floating-point calculation or a cache-miss usually makes an instruction stay in a stage for multiple cycles. Since other instructions ahead of this multiple-cycle operation keep moving forward along the pipeline at the speed of one stage per cycle, stalls are generated.
- **Control and data hazards:** An instruction involved in branching (control) or data dependency (hazard) has to freeze the pipelines until the hazard is resolved.
- **Class conflicts:** In each cycle, a group of P instructions are issued. However, sometimes more than one instruction in the group belongs to the same type and requests the same pipeline (e.g., two instructions want to use the same hardware resources). That leaves some pipeline(s) unused by this group of instructions effectively creating pipeline stalls.

To determine the average number of stalls generated by each instruction, all of the above sources must be considered. First, assume that an instruction requiring multiple cycle operations at some pipeline stage requires this instruction to stay in this stage for X cycles. In order to preserve in-order instruction sequencing, all P pipelines have to be frozen at this stage, and therefore $X * P$ stalls are generated under this condition. To approximate X, note that many multiple cycle operations, such as a cache miss, require a (relatively) fixed service time. Since deeper pipelines (larger S) yield smaller cycle times, such operations generally need more cycles as pipeline depth increases and therefore X can be taken to be roughly proportional to S. The average number of stalls generated by a multiple cycle operation can thus be expressed as:

$$A_1 = aPS \tag{3}$$

Second, when an instruction is involved in a control or data hazard, all P pipelines have to be frozen until the hazard is resolved (to insure in-order execution). The number of cycles needed to resolve a hazard is estimated as proportional to $S - 1$. This follows the generally accepted notion that longer pipelines potentially generate more hazards, while dealing with the case that non-pipelined machines ($S = 1$) have no control and data hazards. The average number of stalls generated in this case is estimated as:

$$A_2 = bP(S - 1) \tag{4}$$

Finally, when class conflicts occur, some instructions will be rejected, thus causing some pipelines to idle for a cycle. Generally one can expect such conflicts to grow with P, however, with a single pipe ($P = 1$) there are no such conflicts, hence the average number of stalls generated by class conflicts can be approximated as:

$$A_3 = c(P - 1) \tag{5}$$

Summing the above sources of stalls, A can be written as:

$$A = A_1 + A_2 + A_3 = (a + b)PS + (c - b)P - c \tag{6}$$

Performing simple parameter substitution yields:

$$IPC = P \,/\, (1 + A) = 1 \,/\, (\alpha S + \beta + \gamma/P) \tag{7}$$

Two general approaches are available for obtaining parameter values. The first is to obtain instruction level statistics relating to the various sources of stalls (e.g., cache misses, branching probability, etc.) and from this data determine the values of a, b and c. The second is to obtain empirical data on IPC values for benchmark programs running of different machines, and then estimate the parameters α, β and γ through curve fitting techniques. This latter approach is pursued in Section 6.

At this point, some interesting preliminary results can be derived from the simple IPC model given above. First, assuming that $\beta = 0$ and an ideal system is present where the cycle time $t_{cycle} = 1/S$, the system throughput can be written as:

$$Throughput = IPC \,/\, t_{cycle} \;=\; 1\,/\,(\alpha + \gamma/SP) \tag{8}$$

This expression shows that under a specific condition ($\beta = 0$), S and P make equivalent contribution to the throughput (i.e., doubling pipeline depth achieves the same performance gain as doubling the number of pipelines). Thus the IPC model nicely reflects the fact that superscalar and superpipelined machines are roughly equivalent ways to exploit instruction-level parallelism [12].

Note also that for a single-pipeline architecture ($P = 1$) with $\lambda = \beta + \gamma$, the IPC equation can be simplified as:

$$IPC = 1\,/\,(\alpha S + \lambda) \tag{9}$$

This is a reasonable description of the relation between IPC and S in single pipelined machines. Furthermore, assigning $\alpha = 1/v$ and $\lambda = 1 - 1/v$, Equation 9 becomes:

$$IPC = v\,/\,(v + S - 1) \tag{10}$$

This is standard utilization equation of a vector pipeline, where v is the vector length and $S - 1$ is the pipeline fill time [1].

4 Clocked Pipeline Model Development

As indicated earlier, instruction pipelines do not achieve ideal performance for three principal reasons: unbalanced workload per stage, non-ideal IPC, and synchronization overhead. Workload imbalance is highly dependent on design techniques and skills available as well as on the properties of computation functions to be implemented. In order to develop models which are general, however, it is assumed that computation at each pipeline stage can be modeled as the evaluation

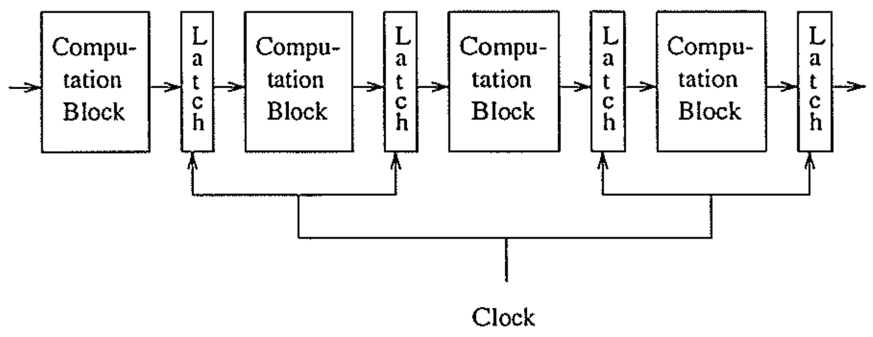

Figure 2: Basic clocked pipeline model

of a combinational logic function which can be partitioned into any integer number of equal subfunctions as necessary. Thus, each stage of the pipeline can be made to have equal computation time (on average).

The second reason pipelines do not achieve ideal performance relates to the *non-ideal* IPC. This has been studied in Section 3 and the expressions developed there reflect the performance penalties associated with various multiple cycle instructions, hazards and class conflicts. In this and the next sections, synchronization methodologies, overhead, and cycle times will be discussed.

Most pipelines found in commercial systems are clocked and can be modeled as variants of Figure 2. Each computation block is made up of combinational logic which performs some part of the overall function (instruction) to be achieved. Following each computation block is a latch which, on the occurrence of a clock event, samples the outputs of the block and holds the sample at its own output until the next clock event occurs. A computation block and its latch constitute a stage or a segment of the pipeline.

A clock cycle includes three components. First, data must propagate through the computation block (i.e., be processed). Then, computation block results must be latched (controlled by clock signals). Finally, the maximum clock skew must be accounted for during in this period. Thus, clock cycle time can be written as:

$$t_{cycle-clk} = t_{comp} + t_{latch} + t_{skew} \tag{11}$$

The instruction to be pipelined is assumed to require t_{all} processing time in its non-pipelined

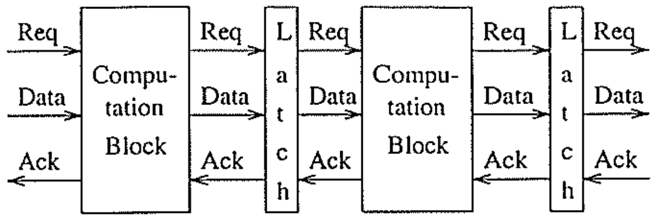

Figure 3: Basic asynchronous pipeline model

version and, since perfect balancing among the stages is possible, the computation time, t_{comp}, of each stage is equal to t_{all}/S. A multiple-phase clocking scheme is assumed to be used and the latching delay is taken to be constant independent of S and P [2]

The clock skew, t_{skew}, is determined by differences in clock signal line lengths; differences in line parameters (e.g., line time constant); differences in delays through active line elements; and differences in latch threshold voltages. Assuming a simple linear pipeline layout and with both P and S taken as powers of 2, the first source of skew can be eliminated by using a binary tree distribution (Figure 2). The skew associated with the remaining sources can be estimated as being some fraction of the average overall delay through the tree. For a (P, S) system, there are $P * S$ sets of stage latches, and therefore $\log_2 PS$ tree levels are needed. Overall delay (including delays of metal lines and drivers) is proportional to the number of tree levels. Assuming all elements on the chip (active and passive) have the same minimum to maximum delay ratio denoted as r, clock skew is the difference between the maximum tree delay ($t_k \log_2 PS$) and minimum tree delay ($rt_k \log_2 PS$), where t_k is a time constant that can be empirically determined. The clock cycle time (Equation 11) therefore can be rewritten as:

$$t_{cycle-clk} = t_{all} \, / \, S + t_{latching} + t_k(1-r)\log_2 PS \tag{12}$$

5 Asynchronous Pipeline Model Development

Figure 3 shows the basic model of an asynchronous pipeline. At a functional level, the computation

[2]Note some high performance vector machines have employed single-phase, single-level-latch clocking schemes where, to avoid race conditions, an intentional (padding) delay is sometimes added to the computation. In this case latching/padding delays can be dependent on S [3, 5, 13].

block is assumed to be the same as in the clocked case (Figure 2)[3]. The timing of data transitions between stages is controlled by a handshaking protocol. Blocks are self-timed so that a stage sends a request signal to its successor stage when computation is completed and data has been placed on the data lines. If the successor stage can accept the data, an acknowledge signal is returned, and the job is passed to next stage in the pipeline. Otherwise, the job waits in the current stage until the successor stage is empty. Though not shown here, multiple buffers or latches may be present between the computational blocks to help increase average block utilization and overall throughput.

Finding the throughput of an asynchronous pipeline is difficult, because the pipeline is elastic in the sense that the time to get through a stage may be variable and dependent not only on the vacancy of succeeding stages, but on the actual data being processed in a given stage. However, two cases, one optimistic and one pessimistic, may be used to approximate the throughput.

In the optimistic case, assume that a buffer with sufficient capacity is present between any two adjacent stages so that there is always an empty buffer available for intermediate results. Assume also that a new instruction is available at the input to each pipeline whenever the first stage of the pipeline becomes free. The cycle time at stage j of the i^{th} pipeline consists of the computation time and synchronization time, and is denoted by the random variable $t_{i,j}$ with expected value $\mathrm{E}(t_{i,j})$. The overall throughput of the system will be bounded from below by the longest of these expected cycle times. Thus, for this optimistic case the overall mean cycle time is:

$$t_{cycle-async-op} = \max[\mathrm{E}(t_{1,1}), \mathrm{E}(t_{1,2}), \ldots, \mathrm{E}(t_{P,S})] \tag{13}$$

A second bound (pessimistic) on the cycle time can be obtained by assuming that there is a zero-delay AND gate gathering the completion signals from all stages of all pipelines. Assertion of this gate's output indicates that computation over all computation blocks is complete. At this point the AND gate output enables each stage to pass its results to the succeeding stage and accept results from a prior stage thus starting the next cycle. Thus, every stage is in lock-step

[3]This assumption is made for simplicity of analysis. It will not hold in a number of situations where there may be no advantage, for example, in providing early completion logic when a clocked design is employed.

with the stage which has the maximum cycle time, and the average cycle time may be taken to be the mean of the maximum cycle times over all stages and pipelines.

$$t_{cycle-async-pe} = \mathrm{E}[\max(t_{1,1}, t_{1,2}, \ldots, t_{P,S})] \tag{14}$$

Since all stages are assumed identical in cycle time, once $t_{i,j}$ is decided, optimistic and pessimistic performance of an asynchronous superscalar machine can be evaluated. $t_{i,j}$ has two components: computation time and synchronization overhead that includes delays associated with completion detection and handshaking. Therefore, $t_{i,j}$ can be expressed as:

$$t_{i,j} = t_{comp-async} + t_{sync-oh} \tag{15}$$

Compared with clocked designs, one potential advantage of asynchronous designs is the speed increase associated with eliminating the bound on clock cycle time which is tied to global maximum stage computation time. This advantage derives from the following sources:

- **Operand variation:** The asynchronous approach can take advantage of computation time differences which are related to input operands that may cause certain function evaluations to proceed faster than others.
- **Instruction variation:** In instruction pipelines, a large variance in stage completion times can be expected due to the differing characteristics of various instructions. The asynchronous approach can take advantage of this instruction time variance.
- **Fabrication variation:** Fabrication processes are not always uniform. Clocked cycle time is set to tolerate certain "worst" case fabrication conditions. Asynchronous designs can respond to average conditions.
- **Environmental variation:** To ensure correct operation, clocked designs must be based on worst case assumptions concerning environmental operating conditions. This introduces additional delays which must be built into the clocked design. For example, the clock period must be set for worst case temperature conditions even though, in general, operating temperatures are not worst case. Asynchronous systems can take advantage of the increased speed associated with nominal operating environments.

The overall instruction execution time for a non-pipelined design has been defined for a clocked environment as t_{all} and represents the *worst-case* execution time. To use t_{all} in developing asynchronous performance models certain modifications must be made.

First, a completion time parameter w ($0 \leq w \leq 1$) is introduced to reflect the possibility of early completions due to operand and instruction variations. For asynchronous pipelines, the maximum computation time in a stage is t_{all}/S (i.e., the worst-case computation delay of a single stage), but the minimum computation time is expressed as wt_{all}/S. Second, it is assumed that all elements on the chip (active and passive) have the same minimum to maximum delay ratio, r, due to fabrication variation. The minimum value of the cycle time (including computation time and synchronization overhead) should therefore include r as a multiplicand. Finally, an environmental parameter e ($0 \leq e \leq 1$) is introduced to reflect the actual (nominal) operating environment. For clocked designs, worst-case environment assumptions require setting $e = 1$. Note that this environmental factor applies to both the maximum and minimum values of $t_{i,j}$ since it is assumed that environment across all stages is the same (fabrication variations can occur between stages). Considering all of the variations discussed above, the value of $t_{i,j}$ is distributed between its maximum and minimum values as:

$$t_{i,j-max} = e(t_{all} \,/\, S + t_{sync-oh}) \tag{16}$$

$$t_{i,j-min} = e(wrt_{all} \,/\, S + rt_{sync-oh}) \tag{17}$$

The distribution of $t_{i,j}$ appears too complex to be modeled since it depends on all of the delay variations described above. For simplicity we assume that all $t_{i,j}$'s (i.e., $i = 1, 2, \ldots, P$ and $j = 1, 2, \ldots, S$) are identical, independent, and uniformly distributed between the minimum and maximum values given above. Using Equations 13 and 14, the optimistic and pessimistic solutions for the average cycle time of asynchronous systems are:

$$\begin{aligned} t_{cycle-async-op} &= (t_{i,j-max} + t_{i,j-min}) \,/\, 2 \\ &= e[(1 + wr)t_{all}/S + (1 + r)t_{sync-oh}] \,/\, 2 \end{aligned} \tag{18}$$

$$\begin{aligned} t_{cycle-async-pe} &= (PSt_{i,j-max} + t_{i,j-min}) \,/\, (PS + 1) \\ &= e[(PS + wr)t_{all}/S + (PS + r)t_{sync-oh}] \,/\, (PS + 1) \end{aligned} \tag{19}$$

The PS terms in Equation 19 result from obtaining the *maximum* of the expected values of a set of $P * S$ identical, independently distributed uniform random variables.

The optimistic solution (Equation 18) represents the performance of an aggressive asynchronous design where the instruction sequencing can be violated (i.e., instructions can move along the pipelines at their highest speeds unless resource, data, or control hazards are detected). For this aggressive design, a scoreboard is needed to keep track of the instructions flow. In addition, well designed buffers (small delay) with sufficient capacity are needed to make an asynchronous processor close to this performance upper bound. The pessimistic solution (Equation 19), on the other hand, represents the performance of a conservative asynchronous design where the pipeline stages are locked step, similar to its clocked counterpart. A conservative asynchronous design does not need a scoreboard and can be easily transformed from a clocked design.

6 Parameter Selections and Result Curves

Models developed in previous sections are summarized as follows:

- Throughput (from Equation 1):

$$Throughput = IPC \,/\, t_{cycle} \tag{20}$$

- IPC (from Equation 7):

$$IPC = 1 \,/\, (\alpha S + \beta + \gamma / P) \tag{21}$$

- Clocked cycle time (from Equation 12):

$$t_{cycle-clk} = t_{all} \,/\, S + t_{latching} + t_k(1-r)\log_2 PS \tag{22}$$

- Asynchronous cycle time (from Equations 18 and 19):

$$t_{cycle-async-op} = e[(1+wr)t_{all}/S + (1+r)t_{sync-oh}] \,/\, 2 \tag{23}$$

$$t_{cycle-async-pe} = e[(PS+wr)t_{all}/S + (PS+r)t_{sync-oh}] \,/\, (PS+1) \tag{24}$$

6.1 IPC Evaluation

Ideally, one would like to have a large number of data points available to parameterize the IPC model and obtain α, β and γ. Unfortunately, only a limited number of pipelined and superscalar designs have been fabricated and only limited performance data published. Given these constraints, coefficients in the IPC model have been obtained in two steps. First, α and $\beta + \gamma$ are evaluated by examining single-pipelined designs (i.e., $P = 1$) and second, the values of β and γ are be obtained individually by studying a selected superscalar design.

Three points were selected to decide the single-pipelined IPC curve.

- IPC of a 1-stage (non-pipelined) processor of the DLX [11] type is given as 91%. This is under the assumption that 5% of the instructions are floating point and 5% of the instructions have cache misses, and each of these instructions causes one cycle penalty (Clocks Per Instruction, CPI = 1.1).
- IPC of a 5-stage processor is 58%. This is based on [2] where CPI (the inverse of IPC), measured using the SPEC benchmark, of a 5-stage MIPS R3000 is reported as 1.71.
- IPC of an 8-stage processor is 43%. This is based on [14] where CPI, also measured using SPEC benchmark, of an 8-stage R4000 (primary-cache only) is reported as 2.3.

Performing a least squares fit of IPC values (given above) for $S = 1$, 5, and 8 yields the values for the constant α and $\beta + \gamma$ and results in the function which is shown in Figure 4:

$$IPC = 1\,/\,(0.17S + 0.91) \tag{25}$$

This curve not only matches the above three empirical points but also corresponds to the super-pipelined performance curve (for $S >= 4$) presented by Jouppi and Wall [12].

The superscalar performance curve presented by Jouppi and Wall is employed to obtain values of β and γ. In their paper, superscalar machines of $S = 4$ are simulated, and performance is presented as a function of P. Fitting to their results, the overall superscalar IPC equation can be quantified as:

$$IPC = 1\,/\,(0.17S - 0.05 + 0.96/P) \tag{26}$$

Figure 5 shows IPC curves for $P = 1$, 2, 4, and 8. Note that the *ideal* IPC is equal to the number

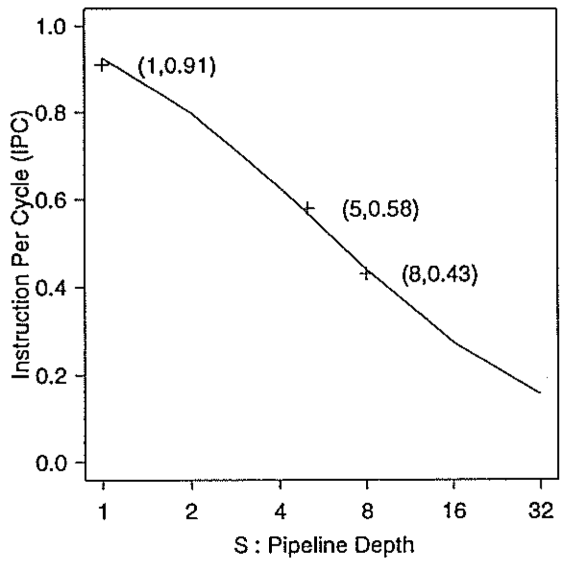

Figure 4: IPC as a function of S
($P = 1$)

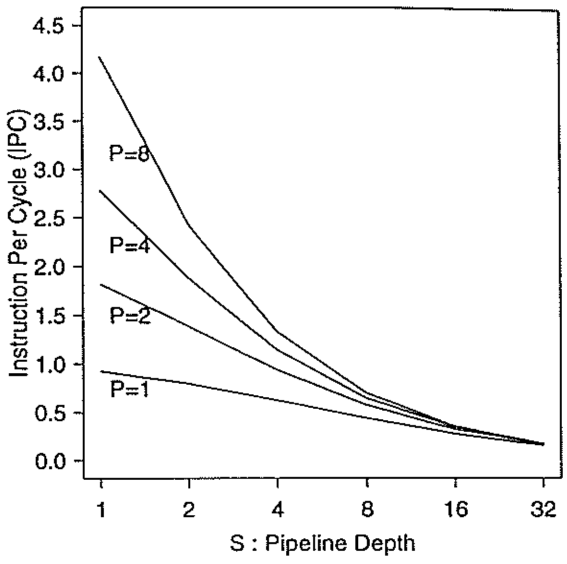

Figure 5: IPC as a function of S
($P = 1, 2, 4, 8$)

of pipelines (P) and independent of pipeline depth (S). Thus, Figure 5 shows how the various factors discussed earlier reduces the IPC from is potential maximum.

6.2 Parameter selections and performance comparison

Table 1 contains variable definitions associated with the models developed. All parameters in Equations 20 to 24 and their typical values (except S and P) are included. The maximum computation delay in the non-pipelined design (t_{all}) is selected as 35 ns. This roughly corresponds to

Table 1: Variable definitions and values

Variable	Definition	Value
S	number of pipeline stages	
P	number of pipelines in the system	
t_{all}	maximum computation delay of non-pipelined version	35 ns
t_{latch}	maximum latching delay	1.4 ns
$t_{sync-oh}$	synchronization overhead in asynchronous design	4 ns
t_k	clock delay time constant	1 ns
r	minimum to maximum propagation delay ratio	0.9
w	completion time parameter	0.5
e	operating-environment parameter	0.8

a 100 logic gate delay (i.e., 0.35 ns per gate) associated with using a 0.5-μ technology process with a 3.3-V supply. Assume that two-phase clocking scheme is employed in our clocked model. Each stage needs two-level latches (master and slave), and each latch has two gate levels. Maximum latching delay (t_{latch}) is selected as 1.4 ns. Synchronization overhead in the asynchronous design ($t_{sync-oh}$) is taken as 4 ns which roughly corresponds 12 gate delays being used for completion detection and a full handshake. Clock skew time constant is given as 1 ns. Applied to Equation 22, an 8-stage pipelined machine yields about 0.3 ns clock skew.

Fabrication derived variation in the design (i.e., the difference between minimum and maximum propagation delays for gates of the same type) is taken to be 10% ($r = 0.9$). The completion time parameter, w, which reflects operand and instruction variations is given as 0.5. Thus, the effective computation time of the simplest operation takes 50% of that of the most complex operation. Some instructions actually take very small or even zero time at certain pipeline stages, and the average delay of some combinational logic, such as an adder, is much shorter than its worst-case delay [7]. Therefore, the value (i.e., $w = 0.5$) chosen here is rather conservative. Finally, the environmental parameter e is assumed to be 0.8, which means a gate delay under the typical environment is 80% of that under the worst-case environment that is tolerated by the clocked design [9].

Using the selected parameter values and the models in Equations 20 to 24, performance curves of clocked, optimistic asynchronous, and pessimistic asynchronous designs are shown in Figures 6 to 8.

Figure 6 shows that the optimum number of stages for a clocked, single-pipelined processor is about 8 which roughly corresponds to the number being used in the instruction pipelines of some contemporary RISC processors (e.g., MIPS R4000 and R4400). The clock rate associated with this single pipeline design is about 165 Mhz and the average performance is about 73 MIPS. For a superscalar design with two instructions issued per cycle ($P = 2$), optimum number of stages is still around 8. The Alpha architecture is similar to this design. Our results show that the optimal performance of $P = 2$ architecture is about 30% better than performance of $P = 1$. Issuing four

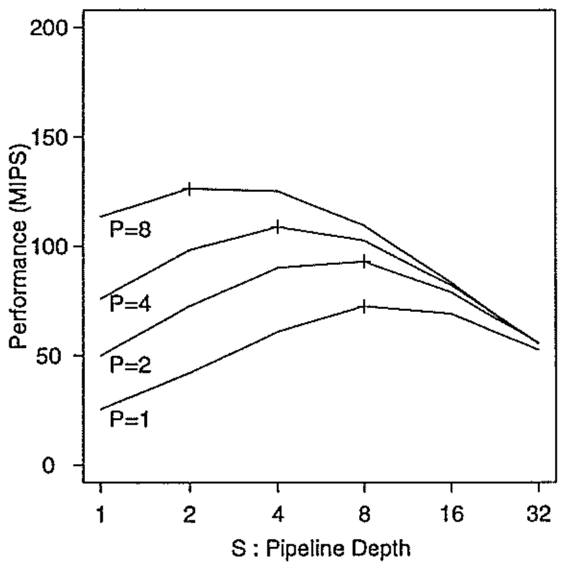

Figure 6: Clocked system performance

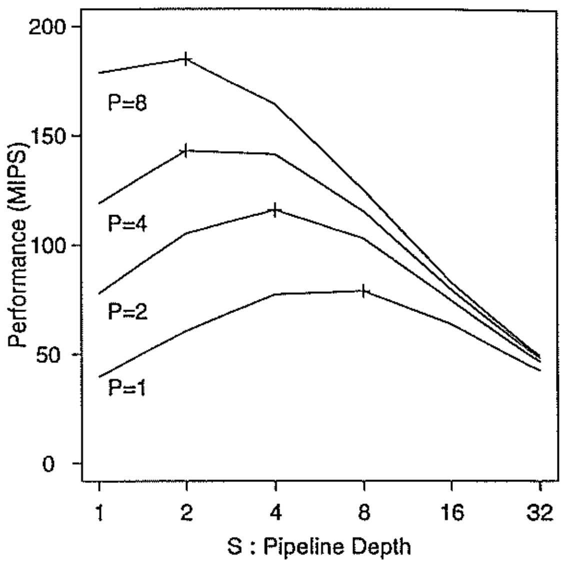

Figure 7: Optimistic asynchronous system performance

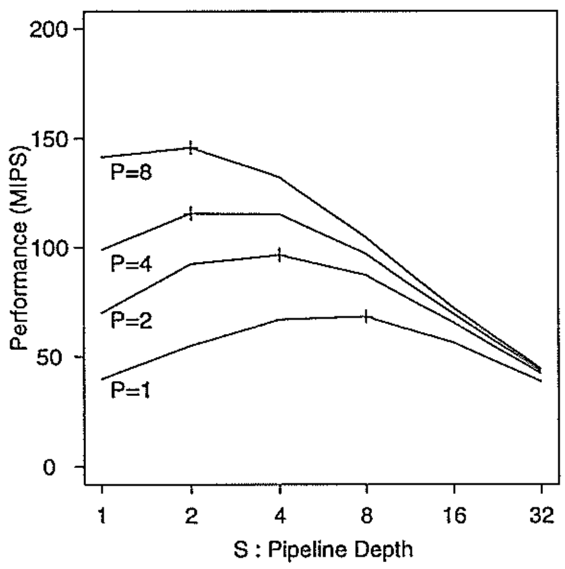

Figure 8: Pessimistic asynchronous system performance

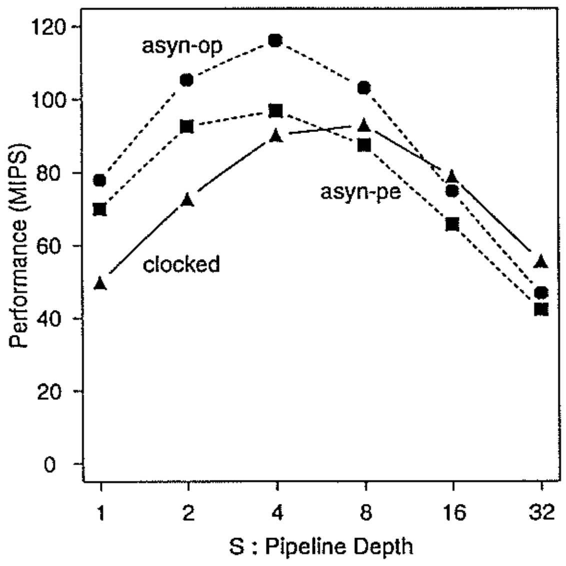

Figure 9: Performance comparison of design methodologies ($P = 2$)

or more instructions per cycle may be difficult and expensive. To do this, many modules have to be duplicated to reduce resource hazards. Our model shows that the optimal performance of $P = 4$ and $P = 8$ superscalar machines is only better than its $P = 2$ counterpart by 17% and 36%.

Figure 7 shows optimistic performance of asynchronous architectures. Note that from Equation 23 the cycle time in the optimistic case is independent of P. Therefore increasing P has no negative impact on cycle time and optimistic asynchronous superscalar designs have higher performance than clocked superscalar designs whose clock skew increases with P. In the case of $P = 1$, an aggressively designed asynchronous processor that can reach its optimistic performance bound runs only 9% better than its clocked counterpart. In the case of $P = 8$, however, the advantage of the asynchronous processor increases to 46%. Conservative asynchronous designs have somewhat higher performance then comparable clocked designs, however, this performance gain can be attributed to the environmental variation parameter in the asynchronous design.

In Figure 9, the clocked design and aggressive and conservative asynchronous designs are compared. Since $P = 2$ is widely employed in superscalar architectures, the three curves are plotted for this condition. This figure shows that optimal pipeline depth in the asynchronous case (4) is shorter than that in the clocked case (8). Since shorter pipelines are usually easier to design, this indicates another advantage of the asynchronous methodology. The difference between the optimistic and pessimistic bounds in the asynchronous design is about 20% which is attributed assumptions associated with the two models.

7 Conclusions

In this paper, a set of performance models for pipelined and superscalar systems are developed. These models provide a wide range of applications that cover both clocked and asynchronous methodologies. Typical values of the parameters in the models are selected, and performance curves are plotted and compared.

Our results show that the best combinations of (P, S) (for both clocked and asynchronous designs) usually have the characteristic of $P * S$ being equal to 8 or 16. This indicates a limit to instruction-level parallelism.

Clocked superscalar designs appear to gain little in performance for P values greater than 4. Asynchronous designs are more suitable to superscalar architectures, however, aggressive design techniques will be needed to push its performance towards the optimistic bounds presented.

A number of factors that are hard to model have been ignored in this analysis. One is the issue of workload imbalance. In this paper, it has been assumed that the overall function can be evenly partitioned into any number of pipeline stages. However, for larger S and P, it will be more difficult to have such a balanced partitioning. Another factor concerns chip size. It has been assumed that both asynchronous and clocked designs require about the same chip area. However, in asynchronous dual rail designs, for example, we can expect this design to require more area than the comparable clocked design. This has implications which have not been considered here.

Research is currently being pursued on the design of an aggressive asynchronous architecture which has features such as: large P, small S, a scoreboard that allows instructions to flow at high speed, and multiple buffers between stages to reduce blocking probability. The goal of this design is to push the asynchronous performance close to its optimistic bound. From our models it can be observed that if the optimistic performance of a $(P, S) = (8, 2)$ asynchronous superscalar machine (185 MIPS) can be achieved, this machine will run 2 times faster than a $(P, S) = (2, 8)$ clocked machine.

References

[1] Jean-Loup Baer. *Computer System Architecture.* Computer Science Press, Maryland, 1980.

[2] D. Bhandarkar and D.W. Clark. Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization. In *ASPLOS-IV Proceedings*, pages 310–319, Santa Clara, CA, April 1991.

[3] L.W. Cotten. Circuit Implementation of High-Speed Pipeline Systems. In *Proc. AFIPS - Fall Joint Computer Conference*, pages 489–504, 1965.

[4] P.K. Dubey and M.J. Flynn. Optimal Pipelining. *J. of Parallel and Distributed Computing*, pages 10–19, 1990.

[5] Fawcett. Maximal Clocking Rates for Pipelined Digital Systems. Master's thesis, EE, UI-UC, 1975.

[6] M.A. Franklin and T. Pan. Clocked and Asynchronous Instruction Pipelines. In *Proc. 26th ACM/IEEE Symp. on Microarchitecture*, pages 177–184, Austin, TX, December 1993.

[7] M.A. Franklin and T. Pan. Performance Comparison of Asynchronous Adders. In *Proc. Symp. on Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, Utah, November 1994.

[8] S.B. Furber, P. Day, J.D. Garside, N.C. Paver, and J.V. Woods. A Micropipelined ARM. In *Int'l Conf. on Very Large Scale Integration (VLSI'93)*, September 1993.

[9] J.D. Garside. A CMOS VLSI Implementation of an Asynchronous ALU. In *Proc. IFIP Conf. on Asynchronous Design Methodologies*, Manchester, England, March 1993.

[10] T.G. Hallin and M.J. Flynn. Pipelining of Arithmetic Functions. *IEEE Trans. Computers*, pages 880–886, August 1972.

[11] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Palo Alto, CA, 1990.

[12] N.P. Jouppi and D.W. Wall. Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines. In *ASPLOS-III Proceedings*, pages 272–282, April 1989.

[13] S.R. Kunkel and J.E. Smith. Optimal Pipelining in Supercomputers. In *13th Inter. Symp. Comput. Arch.*, pages 404–411, Tokyo, Japan, June 1986.

[14] S. Mirapuri, M. Woodacre, and N. Vasseghi. The Mips R4000 Processor. *IEEE Micro*, pages 10–22, April 1992.

[15] R.F. Sproull, I.E. Sutherland, and C.E. Molnar. Counterflow Pipeline Processor Architecture. Technical Report SMLI TR-94-25, SUN Microsystems Lab. Inc., Mountain View, CA, 1994.

[16] D.F. Wann and M.A. Franklin. Asynchronous and Clocked Control Structures for VLSI Based Interconnection Networks. *IEEE Trans. Computers*, pages 284–293, March 1983.