Exploring Configurations of Functional Units in an Out-of-Order Superscalar Processor

Stéphan Jourdan, Pascal Sainrat, and Daniel Litaize

Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier, 118 route de Narbonne, 31062 Toulouse cedex, FRANCE

{jourdan,sainrat,litaize}@irit.fr
http://www.irit.fr/ACTIVITES/EQ_APARA/

Abstract

This study has been carried out in order to determine cost-effective configurations of functional units for multiple-issue out-of-order superscalar processors. The trace-driven simulations were performed on the six integer and the fourteen floating-point programs from the SPEC 92 suite. We first evaluate the number of instructions allowed to be concurrently processed by the execution stages of the pipeline. We then apply some restrictions on the execution issue of different instruction classes in order to define these configurations. We conclude that five to nine functional units are necessary to exploit Instruction-Level Parallelism. An important point is that several data cache ports are required in a processor of degree 4 or more. Finally, we report on complementary results on the utilization rate of the functional units.

Keywords: *Instruction-level parallelism, Superscalar microprocessor, Out-of-Order Execution, and Functional Units.*

Processor Date (ship)	PPC 604 94	MC 88110 92	Ultra Sparc 95	DEC 21164 95	MIPS 10000 95
Degree	4	2	4	4	4
Integer unit	2	2	2	2	2[3]
Shift unit		1			
Divide unit	1	1[1]			
Multiply unit		1[2]			
FP add unit	1	1	2	1	1
Convert unit					
FP divide unit		1[1]			1
FP multiply unit		1[2]		1	
Data cache port	1	1	1	2	1

Table 1 — *Configuration of Functional Units*

[1] Dividers are merged in the same functional unit
[2] Multipliers are merged in the same functional unit
[3] Both units handle integer instructions but only one processes shifts while the other processes divides and multiplies.

ISCA '95, Santa Margherita Ligure Italy

1. Introduction

Nowadays, the superscalar approach is unanimously recognized, but there are many trends and ways of implementing such architectures. For instance, a *branch-history table* and a *branch-target buffer* can be considered in order to improve the fetch effectiveness through branch prediction [YePa92], as well as reservation stations to implement out-of-order execution issue [Toma67].

Nevertheless, the performance implied by such features mainly relies on the computing capacities of the model of execution. Except for Mike Johnson's book [John 91], few studies have been carried out to determine the most cost-effective configurations of functional units, especially on out-of-order superscalar processors. Different choices have been made by manufacturers as shown in table 1 [IbMo94] [Moto91] [Sun95] [Dec95] [Mips94]. Note that among the five processors from table 1, the PowerPC 604 and the R10000 are out-of-order superscalar processors.

Throughout the paper, we study different configurations of functional units according to the degree of the processor. The *lookahead window* refers to all the instructions simultaneously held in the execution pipelines. Its size is defined in harmony with the degree.

This study was performed assuming an ideal instruction-fetch mechanism, no cache miss and a unified instruction-issue buffer. These assumptions were made in order to define the most cost-effective configurations of functional units while considering only data dependencies, and they are dealt with in section 2 as well as the model of execution and the simulation process. Results presented in this paper are for all the programs from the SPEC 92 suite. Section 3 describes these benchmarks. We then determine in section 4 the size of the lookahead window for a degree varying from 2 to 8. Section 5 reports on results of our simulations for a wide variety of integer and load/store unit configurations. Floating-point units are dealt with in section 6. Section 7 offers some complementary results on the utilization rate of the functional units. Conclusions are then summarized in section 8.

2. Model of Execution

The modeled architecture implements an out-of-order execution-issue policy and speculative execution in order to best exploit *instruction-level parallelism* (ILP).

In such an architecture, after their *fetching* and their *decoding*, instructions are *dispatched* from the *instruction-dispatch buffer* to the *instruction-issue buffer*. The upper bound of the number of instructions dispatched each cycle is called the degree of the proces-

sor. Instructions waiting for their operands do not stall the decoding: the missing operands will be forwarded to the instruction-issue buffer. The entries of the instruction-issue buffer are similar to the entries of the reservation stations of the IBM 360/91 [Toma67] except that they can be linked to more than one functional unit. An entry of the instruction-issue buffer holds the operation as well as the source operands when available, or tags to retrieve them otherwise. Entries holding all the values of their source operands, may be *issued* except when a conflict occurs: pipelined units begin the execution of only one instruction at each cycle. When an execution completes, the result is forwarded to the entries of the instruction-issue buffer which require it, if any.

Issuing techniques are the algorithms which arbitrate fireable entries of the instruction-issue buffer. This buffer can be unified (all entries are linked to all functional units), split (all entries are linked to only one functional unit), or mixed. A similar mechanism is called *the node table* by Butler and Patt in [BuPa92]. In their paper, they show that most issuing techniques give almost the same performance for a processor featuring a wide degree. We therefore decided to implement the most natural algorithm named *oldest first* where the entry which holds the oldest dispatched instruction has priority over the others.

To manage interrupts precisely, an entry associated to each dispatched instruction is enqueued in a *reorder buffer*. However, operand values or tags are always obtained during the decoding [SmPl85]. The reorder buffer maintains the initial program order, and its size defines the upper bound of the number of instructions which can be simultaneously processed after their dispatch. As mentioned below, this upper bound is the size of the lookahead window. When an instruction is executed, the result is also forwarded to the associated entry of the reorder buffer since it does not directly update the register file. The update will occur when no previously dispatched instruction can still generate interrupts. Instructions which may generate interrupts, are conditional branches, divides, and memory accesses. In the latter instruction class, subsequent instructions cannot update the file until the end of the address processing only. Once the register file is updated, the instruction is *retired* or *completed*: the entry is dequeued. Each cycle, multiple out-of-order retirements can be made.

To sum up, the different states followed by an instruction are *fetched, decoded, dispatched, issued, executed* and *retired*.

As we want to point out in this study the most cost-effective configurations of functional units to exploit instruction-level parallelism disregarding any other parameters which can lead to performance degradation, we assume:

- an ideal instruction-fetch mechanism : no instruction-cache miss, no branch misprediction and a dispatch only limited by the degree of the processor and the fullness of the reorder buffer,
- no data-cache miss,
- a unified instruction-issue buffer, in order to have an ideal issue scheme. Moreover the choice of another issuing technique (which gives roughly the same performance as *oldest first*) may not modify our result.

Table 1 shows that superscalar processors implement several functional units, each capable of servicing different instruction classes. We define these classes according to their functionality as follows:

- *integer* : arithmetic and logic operations,
- *shift* : shifts and bit-field manipulations,
- *integer multiply*,
- *integer divide*,
- *load/store*: memory loads and stores
- *floating-point arithmetic*,
- *floating-point convert*,
- *floating-point multiply*,
- *floating-point divide*.

We used the `pixie` profiler [Smit91] in order to produce instruction traces from a real processing of the SPEC benchmarks. From all the data reported by this software, we picked out only the opcode and the memory address (in the case of a memory access) for each instruction. These traces are read by our simulator which performs a *cycle-by-cycle* simulation and gathers the mean number of *instructions retired per cycle* (IPC). As previously outlined, the simulator models an ideal instruction-fetch mechanism. Thus, it only takes into account the degree of the processor, the size of the lookahead window, the memory and register data dependencies, and the configuration of functional units we want to evaluate.

Load/store instruction can bypass a store if, and only if, both addresses are known (the value to be stored is forwarded to the load when addresses match). Furthermore, the memory access of stores starts only when all previous instructions cannot produce interrupts anymore, in order to preserve memory coherency.

Latencies of the instruction classes which are listed in table 2, are those of the PowerPC 604. All functional units but divide units, are fully pipelined and mutually independent.

Finally, we wrote another simulator modeling an in-order issue scalar processor, that only keeps track of register and memory data dependencies. The basic assumption is that the current instruction can be executed whatever the processor state is, except of course when its operands are not yet processed. Performance results from this simulator are the best we can expect from an in-order issue RISC scalar processor without any branch-prediction scheme. In this paper, speedups are related to this scalar processor.

Instruction class	Latency	Instruction class	Latency
integer	1	*fp arithmetic*	3
shift	1	*fp convert*	1
integer multiply	3	*fp multiply*	3
integer divide	20	*fp divide*	18s / 31d
load/store	2 / 3	s stands for single precision and d for double precision	

Table 2 — *Latencies*

3. Benchmarks

3.1 Instruction Traces

The results presented in this paper are for programs from the SPEC92 suite [Spec92]. Two subsets are defined: CINT92 (*integer*) and CFP92 (*floating-point*). In order to make our evaluations, we use all programs from both sets (the 6 CINT92 and the 14 CFP92 programs). They were compiled on a R4600-based SGI workstation using the standard makefiles provided with the suite (with all optimizations turned on).As previously mentioned, `pixie` has been used to generate instruction traces (all *NOPs* being removed). Except for *backprop*, *dnasa7*, *wave5*, and *spice2g6*, all the programs have been run to completion: the smallest input files or slightly modified versions have been used. In all, about 600 million instructions have been captured. Because of initializations which do not concern floating-point data, we did not trace the first 200 million instructions of *spice2g6* but the next 50 millions. Table 3 gives the mean distributions.

	CINT 92	CFP 92
branch	*21.2 %*	*8.3 %*
jump	2.5 %	1 %
conditional branch	18.7 %	7.3 %
memory access	*37 %*	*41.7 %*
load	25 %	31.5 %
store	12 %	10.2 %
integer	*41.8 %*	*23.8 %*
arithmetic and logic	37 %	21.3 %
arithmetic	26.7 %	19.5 %
logic	10.3 %	1.8 %
shift	4.6 %	2.3 %
multiply	0.1 %	0.1 %
divide	0.1 %	0.03 %
floating-point	*0 %*	*26.2 %*
arithmetic	0 %	10.9 %
convert	0 %	5.4 %
multiply	0 %	9 %
divide	0 %	0.9 %

Table 3 — *Mean Distribution*

3.2 Arithmetic and Harmonic Mean

The arithmetic mean and the harmonic mean are related by this formula:

$$Harmonic\ \ Mean\ (unit) = \frac{1}{Arithmetic\ \ Mean\ (unit^{-1})}$$

Throughout the paper, our conclusions are based on mean results. We assume that a mean workload consists in the execution of the same number of instructions from each benchmark. For each of these benchmarks, simulation results give a mean temporal cost in *cycle per instruction* (CPI). The right way of processing the mean of IPC values is therefore the harmonic mean (the arithmetic mean for CPI values).

The speedup of a given configuration is the harmonic mean of IPC results from simulations on the given configuration, divided by the harmonic mean of IPC results from simulations on the scalar processor.

4. The Lookahead Window

4.1 Definition

The lookahead window is in fact an abstract representation of the instructions being processed. An instruction enters the window when it is dispatched. It exits the window when it is retired. One way to implement it is the coupling of a reorder buffer and reservation stations. In this case, the lookahead window is exactly the abstraction of the reorder buffer. The size of the window plays a leading role in processor performance. As a matter of fact, the lookahead window contains more and more instructions which can be processed in parallel, as its size increases. But each additional entry implies a hardware cost. We have therefore to minimize this size. This section deals with the study of the impact of the size of the lookahead window on the performance of an out-of-order issue superscalar processor. This study is made according to the assumptions outlined in section 2. Moreover, in order to avoid clouding results with an arbitrary configuration of functional units, we assume an infinite number of units. Finally, one can remember that the size of the lookahead window of the PowerPC 604 is 16 entries.

4.2 CINT 92

Figure 1-a plots the mean performance of the simulation results on integer programs, given in IPC, according to the size of the lookahead window. The speedups of superscalar configurations of degree 2, 4, 6, and 8, with regard to the scalar processor described in section 2 are given in figures 1-b through 1-e respectively. Indeed, it clearly appears in figure 1-a that a processor of degree 8 does not significantly underperform a processor of degree infinite (at least when the size of the lookahead window is within realistic values, i.e. lower than 100 entries). One can wonder about the feasibility of a processor featuring a higher degree. The lookahead window would have to be very large in order to give substantial performance improvement over a processor of degree 8. But this raises another problem which is the validity of the last instructions enqueued in the window: they would be dependent on many predicted branches.

Therefore, we limit our studies on out-of-order issue superscalar processors to degrees lesser or equal to 8.

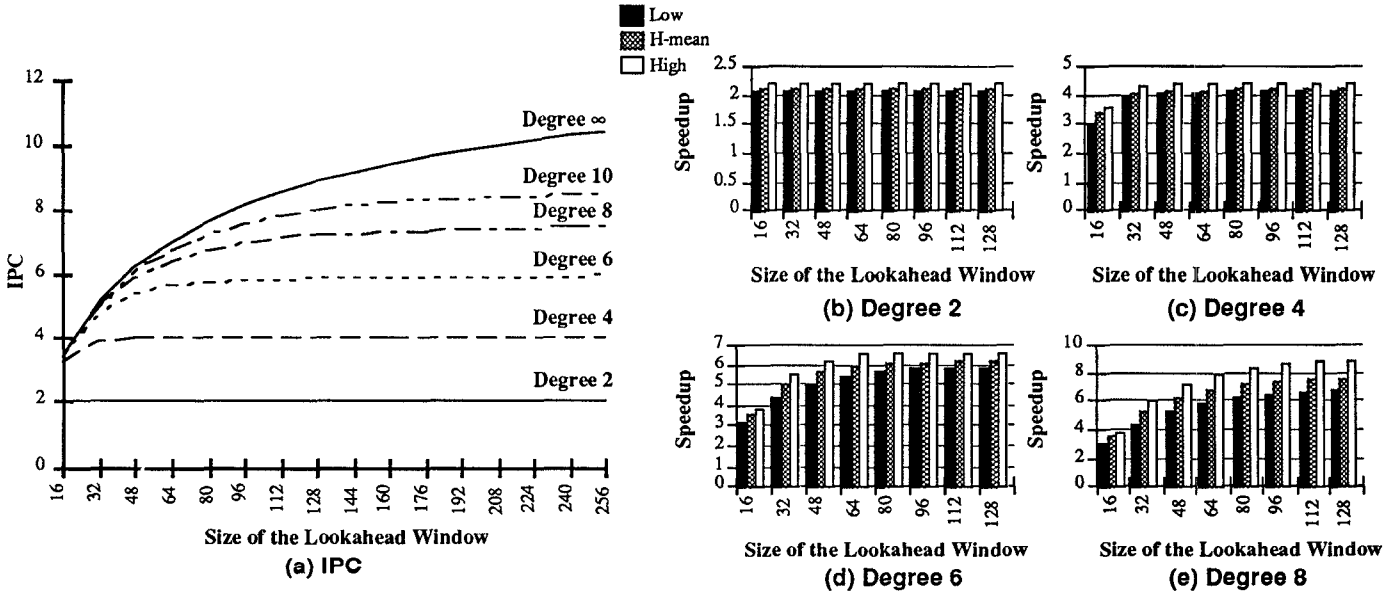

Figure 1 — *Impact of the Size of the Lookahead Window on Performance in Integer Programs*

In figures 1-b through 1-e, the speedups level off for and after a given size of the lookahead window. Moreover, as the degree increases, the upper bound of IPC values moves away from the degree itself. This degradation is due to data dependencies of which the impact is much more important when the degree is high (a processor of degree 2 leads roughly to a 2.0 IPC). Thus, enlarging the lookahead window would not drastically increase the performance because additional instructions would be, for the most part, dependent on at least one other instruction previously enqueued in the window.

However, the results are given for sizes of the lookahead window up to 256 entries. Nowadays, the PowerPC 604, for instance, implements a 16-entry window. Therefore, sizes higher than 100 seem unrealistic for the next few years, but their associated results are plotted for comparison. One can notice that, as previously outlined, such results do not show substantial wins.

Table 4 summarizes the best trade-offs concerning the size of the lookahead window and the implied performance according to the different out-of-order issue processors. The performance degradation with regard to a 256-entry window for each degree, is less than 6 %. Perhaps a better-suited compiler will give improved results for large window but certainly not enough to change our conclusions.

Note that speedups are related to our scalar processor which gives an IPC lower than one (0.945 IPC on CINT92 and 0.75 on CFP92). This explains why speedups of any out-of-order superscalar processor can be higher than the degree.

Degree of the processor	2	4	6	8
Size of the lookahead window	16	32	64	96
Speedup	2.11	4.09	5.91	7.36

Table 4 — *Size of Lookahead Windows (integer programs)*

4.3 CFP 92

Figure 2-a plots the mean performance of simulation results on the 14 floating-point programs, given in IPC, according to the size of the lookahead window. Figures 2-b through 2-e show the speedups with superscalar processors of degree 2, 4, 6, and 8 respectively.

We have not included in this paper the individual results for each CFP92 benchmark. One has to know however that their behavior, according to the size of the lookahead window, is highly dissimilar as shown with the low and high bars. In spite of these individual results, the curves of the harmonic means in figure 2-a increase steadily. Therefore it seems that averaging those 14 benchmarks eliminates any singular values.

Figures 2-b through 2-e clearly state that a large lookahead window is necessary for floating-point programs to best exploit ILP. This is due to the fact that the floating-point programs involve more dependencies than the integer programs and/or the latencies are longer (the mean IPC is lower). Floating-point programs require therefore a larger anticipation in order to overlap several parallelizable operations like independent and successive iterations of the same loop. As a result, the curves do not level off anymore for sizes of the window lesser than 256.

From those results, we cannot determine cost-effective sizes of the lookahead window: high IPCs mean unrealistic sizes of the window. Thus, we keep the sizes chosen in sub-section 4.2, knowing that the processor cannot exploit the whole ILP of the floating-point programs. Table 5 shows the best alternatives. It presents the degradation of performance with regard to the performance of a 256-entry window configuration.

Degree of the processor	2	4	6	8
Size of the lookahead window	16	32	64	96
Performance degradation (%)	16.22	23.25	19.21	15.95
Speedup	1.96	2.86	3.62	4.14

Table 5 — *Size of Lookahead Windows (floating-point programs)*

5. Configuration of Integer and Memory Units

5.1 Universal Integer Units

In usual programs, most instructions belong to the integer and load/store instruction classes (see table 3). In order to evaluate first how much and which integer and load/store units are needed, we do not limit the issuing of the other instruction classes. Thus, any fireable instruction, except for memory access and mono-cycle integer instructions, will be issued.

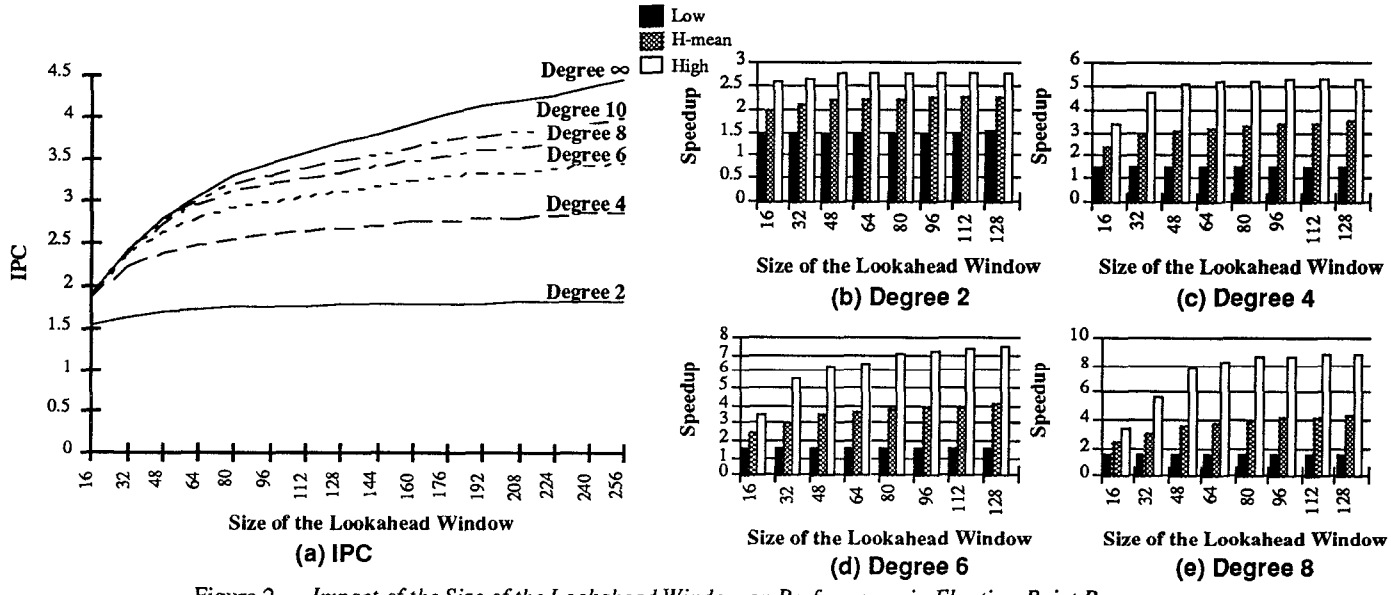

Figure 2 — *Impact of the Size of the Lookahead Window on Performance in Floating-Point Programs*

We first deal with two types of units :

- integer units which process only mono-cycle integer instruction,
- the single load/store unit which implements multiple accesses on the memory hierarchy.

Multiple cache ports can be implemented in various ways. One obvious way is the use of double-speed memories as implemented in the IBM Power 2 [SmWe94], and another is to use an interleaved structure in order to access multiple banks simultaneously provided there is no conflict. This latter solution is implemented in the Intel Pentium [Inte93] and the SGI TFP [Hsu94]. In the Dec Alpha 21164, 2 mirrored banks are implemented in order to feature 2 load ports [Dec 95]. Consequently, no access can be issued simultaneously with a store.

As caches are not modeled, we do not attend to a specific cache structure. Thus, the only issuing restriction on memory instructions is the number of ports available.

Figures 3-a through 3-d show the mean performance from simulations on the integer programs (given in IPC) according to the number of integer units and of memory ports.

Figure 3-a clearly states that, in a processor of degree 2, 2 integer units and 2 memory ports give 99% of the *ideal performance* (unlimited number of functional units). However, 2 integer units and 1 memory port is certainly the most cost-effective configuration, reaching 95% of the ideal performance. Moreover, adding integer units gives roughly no increase of performance.

Figure 3-b highlights the requirement on 2 data cache ports. A processor of degree 4 featuring three integer units and two data cache ports represents a substantial win (92.1 %).

The same arguments apply in a processor of degree 6 but for four integer units and three data cache ports as shown in figure 3-c, passing beyond 92 % of the ideal performance.

Finally, as shown in figure 3-d, a processor of degree 8 requires one additional integer unit and one additional memory port. Consequently, the performance degradation is lower than 9 %.

Another important point to emphasize would be the addition in each configuration of one more memory port. Indeed, this would

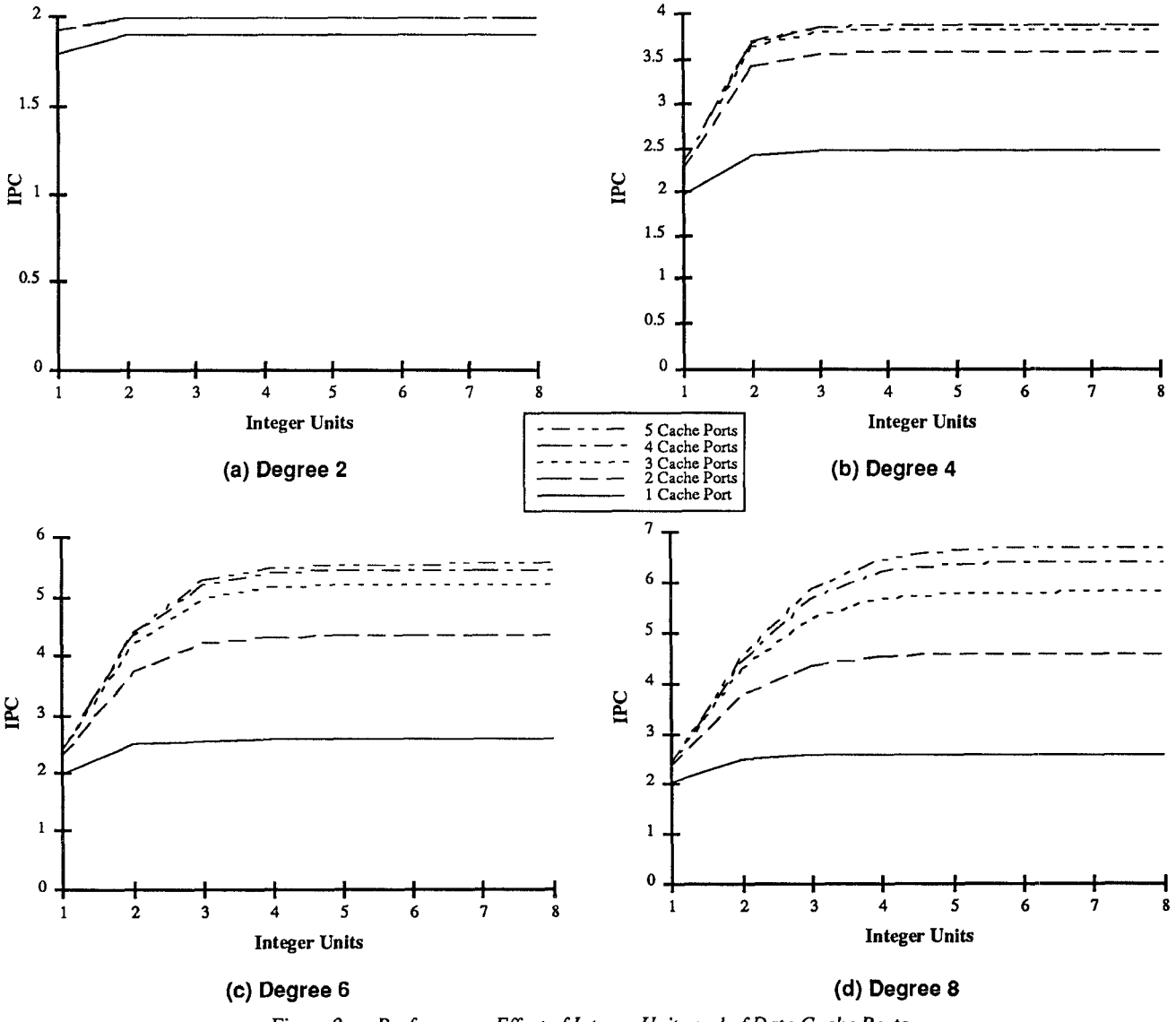

Figure 3 — *Performance Effect of Integer Units and of Data Cache Ports*

nearly lead to a 5 % improvement. Nevertheless, such configurations are not cost-effective because of the hardware cost of adding one cache port. Moreover, varying the number of address processings do not give better alternatives.

Results are not reported here for floating-point programs : the outlined configurations behave the same way concerning memory ports.

Table 6 summarizes these results.

Degree of the processor	2	4	6	8
Integer/Memory/Address	2/1/1	3/2/2	4/3/3	5/4/4
Speedup (CINT 92)	2.0	3.76	5.44	6.69
Performance degradation (%)	5	7.9	8	9
Speedup (CFP 92)	1.83	2.74	3.49	4.02
Performance degradation (%)	6.7	4	3.7	3.1

Table 6 — *Configurations of Functional Units (integer units and memory ports)*

5.2 Shift Units

Up to now, we consider that mono-cycle integer units were handling all classes of integer instructions as in the PowerPC 604, namely arithmetic, logic and shift instructions (integer and shift instruction classes). We must consider, as in the MC88110, the case where some units handle part of these classes. Mixed alternatives have also to be considered. For instance, a configuration can feature several units handling both instruction classes and others handling one of them. Note that such a solution does not increase the complexity of the control (i.e. the cycle time) since a priority list of units associated with each instruction class, can be set up in order to help the dispatch. Moreover, the priority has to favour the instruction dispatch in units handling only one instruction class.

In figures 4-a through 4-d, ISFU is a unit handling arithmetic, logic and shift instructions, IFU is a unit handling only arithmetic and logic instructions while SFU stands for a unit which handles only shift instructions. The listed configurations are ordered by increasing costs.

From all these results, it is clear that the extreme alternatives are not good solutions since they are either too costly or they give poor performance. In a processor of degree 2 or 4, a single ISFU has to be implemented while the others are IFUs. Such configurations give more than 99.8 % of the performance of configurations with only ISFUs. We thus have an economy on silicon area with a negligible performance loss. On the contrary, replacing one more IFU by one ISFU in processors of degree 6 and 8 are cost-effective, and leads to a large increase of performance of 1.1 % and 2.4 % respectively.

Furthermore, simulations which are not reported here, show that featuring one or several functional units handling multi-cycle instructions gives the same results.

Finally, table 7 below summarizes all these observations and gives the cost-effective configurations. As in previous tables, speedups are given related to our in-order scalar processor.

Degree of the processor	Window	Ports	IFU	IBSU	Multi-cycle	Speed up
2	16	1	1	1	1	2.0
4	32	2	2	1	1	3.75
6	64	3	2	2	1	5.44
8	96	4	3	2	1	6.69

Table 7 — *Configurations of Functional Units (CINT92)*

6. Floating-point Units

In most superscalar processors, all floating-point instructions are handled by specific floating-point units. This implies that the integer and floating-point data paths are decoupled as in the PowerPC 604. In order to exploit best ILP, it seems that several floating-point units have to be implemented. Table 1 shows various ways of implementing such units.

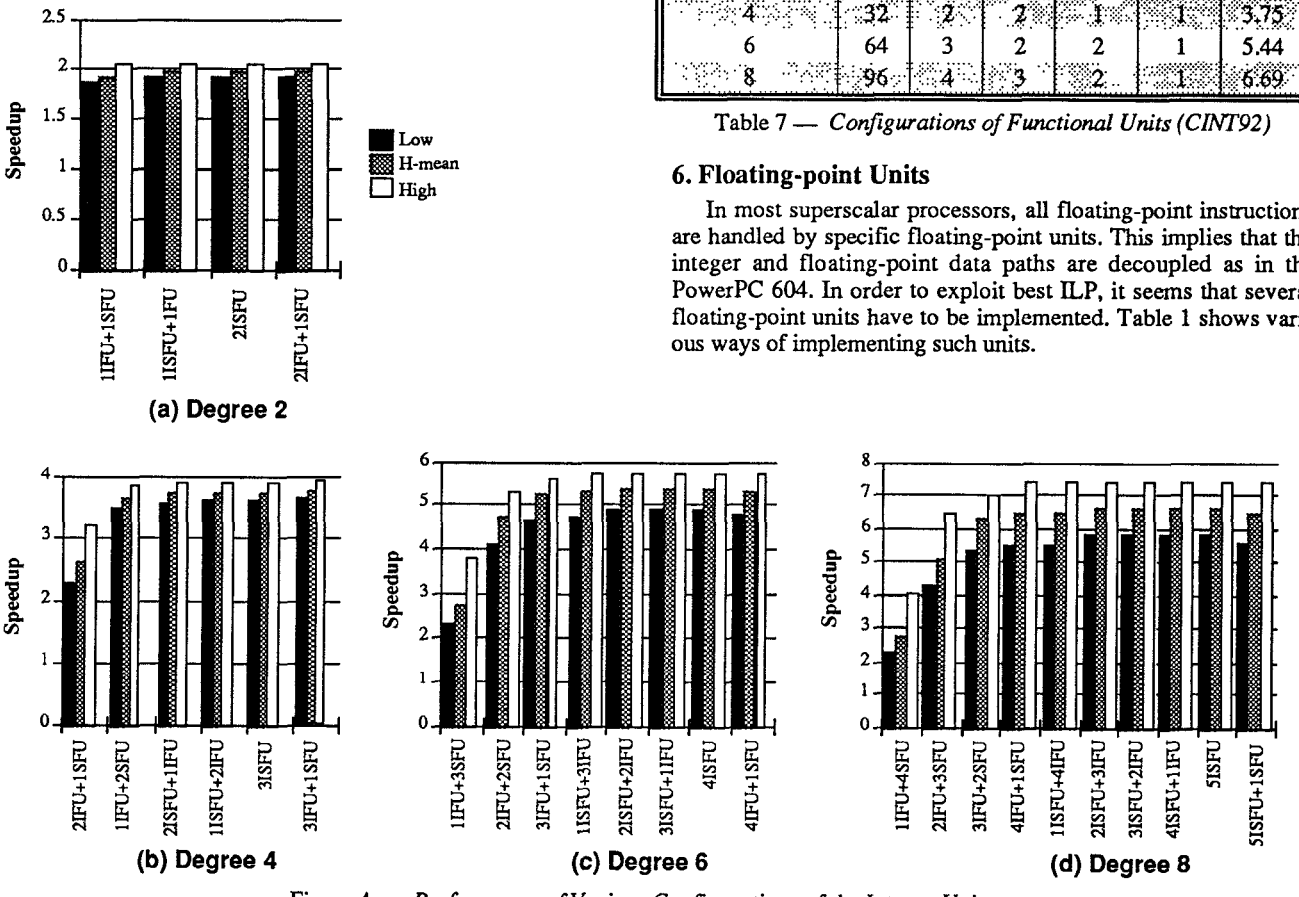

Figure 4 — *Performance of Various Configurations of the Integer Units*

Figure 5 reports on the impact of the number of issues of each floating-point instruction class on performance. The configurations listed in table 7 are used in this section. Figure 5 states that only one issue is required for all these classes except for divides which distinctively feature a non-pipelined execution with a long latency. Furthermore, the number of executed divides is low, and thus multiple issues are required primarily to allow simultaneous executions. A divide unit can therefore be merged with another unit without a significant loss of performance as long as sub-units are implemented.

Consequently, we define a *floating-point unit* (FPU) as a unit capable of servicing all classes. In such a unit, divides do not stall following issues since such instructions are executed in an independent sub-unit.

Figure 6 plots the speedup of configurations according to the number of floating-point units. For a processor of degree 2, a configuration with one unit provides 93 % of the performance of a configuration featuring an infinite number of FPUs. Processors of degree 4, 6, and 8 require 2 FPUs to achieve 96 %, 94 %, and 92 % respectively. Table 8 gives the associated speedups.

Degree of the processor	2	4	6	8
FPU	1	2	2	2
Speedup (CFP92)	1.70	2.63	3.29	3.68

Table 8 — *Configurations of Functional Units (CFP92)*

7. Occupation Rates of the Functional Units

Tables 9 and 10 report on the occupation rates of the previously defined configurations of functional units.

8. Concluding Remarks

In this paper, we have summarized the results of simulations intended to determine cost-effective configurations of functional units in order to exploit instruction-level parallelism in out-of-order superscalar processors. Throughout the paper, we have considered processors of degree 2, 4, and 6, as well as a more aggressive processor of degree 8 while taking into account only register and memory data dependencies. We have therefore ended up with configurations which are expected not to cloud the results of further studies. The defined configurations feature 5 units in a pro-

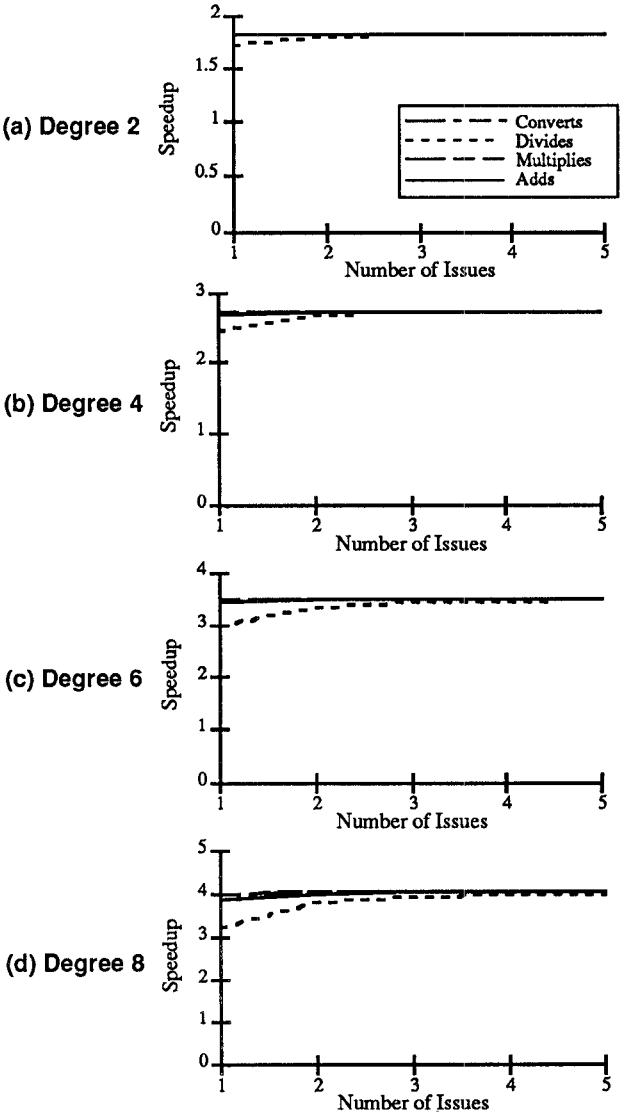

Figure 5 — *Impact of the Number of Floating-Point Units*

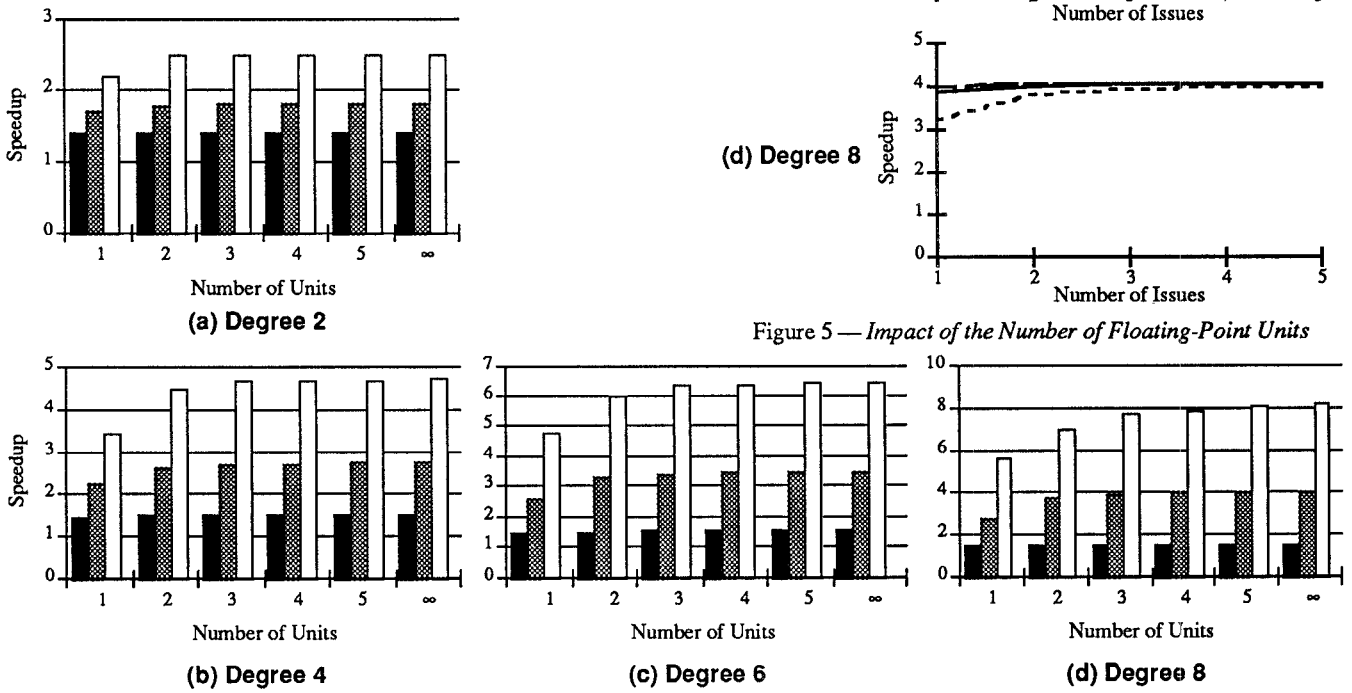

Figure 6 — *Performance of Various Configurations of the Floating-Point Units*

cessor of degree 2, and up to 9 units in a processor of degree 8. These configurations are listed in table 11[1].

Degree of the processor	2	4	6	8
Integer Unit	70	65.8	63.9	59.4
Memory Port	39.7	49.4	53.5	52.4
Multi-cycle Unit	0.8	1.5	2.2	2.8
FPU	0	0	0	0

Table 9 — *Occupation Rates (CINT92)*

Degree of the processor	2	4	6	8
Integer Unit	58.7	50.7	46	41.2
Memory Port	16.8	19.1	18.9	17.7
Multi-cycle Unit	1.2	2	2.5	2.9
FPU	34.9	29.5	39.7	46.8

Table 10 — *Occupation Rates (CFP92)*

Degree of the processor	2	4	6	8
ISFU	1	1	2	2
IFU	1	2	2	3
Multi-cycle Unit	1	1	1	1
FPU	1	2	2	2
Data Cache Port	1	2 (1)	3 (1)	4 (1)
Total Number of Unit	5	7	8	9
Speedup (CINT92)	2	3.75	5.44	6.69
Speedup (CFP92)	1.70	2.63	3.29	3.68

Table 11 — *Final Configurations of Functional Units*

The two three-dimensional graphs summarizes speedups with regard to the in-order scalar processor, with the same assumptions as previously. We consider the increase of the size of the *lookahead window* and of the *degree of the processor* (X axis), and the *addition of functional units* and of *data cache ports* (Y axis). We are mainly interested in comparison with the executive configuration of the PowerPC 604 (16-entry lookahead window, degree 4, base).

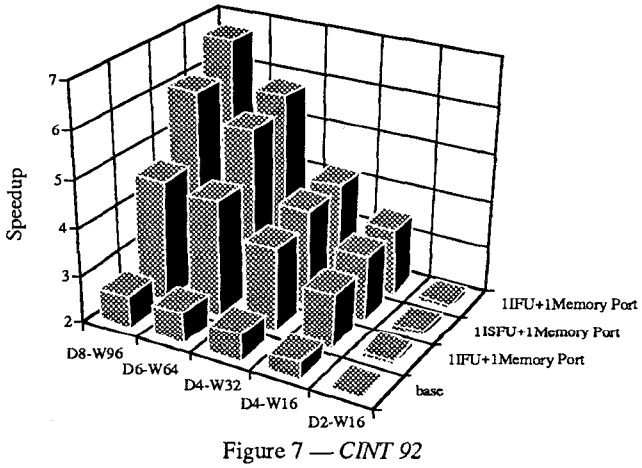

Figure 7 — *CINT 92*

[1] the load/store unit, whatever the number of memory ports may be, is described as a single unit.

The first chart, which concerns integer benchmarks, confirms our choices. For a given degree, bigger configurations present roughly no win. The chart also brings to light that adding one cache port and one IFU to a PowerPC 604 configuration improves the performance by more than 36 %. Our best configuration which is a processor of degree 8 with 4 cache ports and 3 additional integer units, gives a 2.86 speedup with regard to the executive configuration of the PowerPC 604. On the contrary, lowering the number of issues to two instructions par cycle leads to a 14 % performance degradation.

Throughout this study, we assume that these improvements can be made without degrading the cycle time.

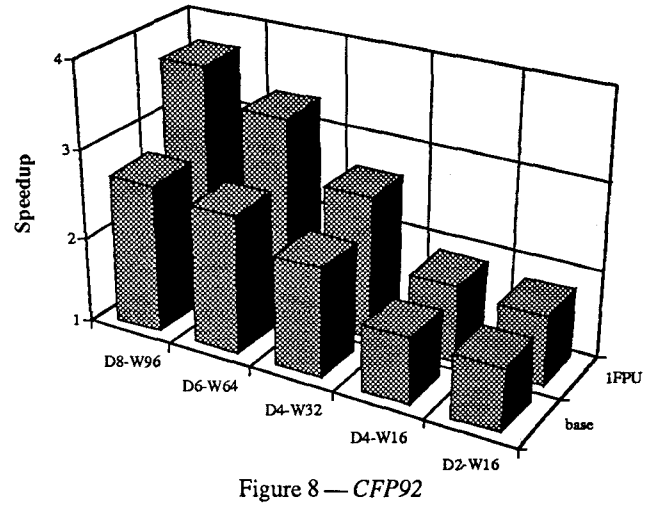

Figure 8 — *CFP92*

The second chart presents simulation results on floating-point programs. The same results can be highlighted. The speedup of the best configuration with regard to the executive configuration of the PowerPC 604 is 2.09. One can note that adding a floating-point unit to a PowerPC604 configuration gives only a 6.7 % improvement of performance.

9. Acknowledgements

We would like to express our thanks to Bhaskar Janakiraman, Joan Eslinger, Jack Carter, and Sassan Hazeghi (SGI) for their kindly help with Pixie.

10. References

[BuPa92] **M. Butler** and **Y. Patt**, "An Investigation of the Performance of Various Dynamic Scheduling techniques", *Proceedings of the 25th Annual International Symposium on Microarchitecture*, Decem-ber 1992

[Dec95] **DEC**, "Scheduling and Issuing Rules for the Alpha 21164", *Product Documentation*, November 1994

[Hsu94] **P. Yan-Tek Hsu**, "Designing the TFP Microprocessor", *IEEE Micro*, April 1994

[IbMo94] **IBM**, "PowerPC 604 RISC Microprocessor Technical Summary", *IBM Advance Information*, MPR604TSU-1, 1994

[Inte93] **Intel**, "Pentium Processor User's Manual", 1993

[John91] **M. Johnson**, "Superscalar Microprocessor Design, *Prentice-Hall*, 1991

[Mips94] **MIPS**, "R10000 Microprocessor Product Overview", October, 1994

[Moto91] **Motorola**, "MC88110: Second Generation RISC Microprocessor User's Manual", 1991.

[Smit91] **M.D. Smith**, "Tracing with Pixie", *Stanford University*, April 1991

[SmPl85] **J.E. Smith, A.R. Pleszkun**, "Implementation of Precise Interrupts in Pipelined Processors", *Proceedings of the 12th Annual International Symposium on Computer Architecture*, June 1985

[SmWe94] **J.E. Smith** and **S. Weiss**, "Power and PowerPC, Principles, Architecture, Implementation", *Morgan Kaufmann Publishers*, Inc., 1994

[SPEC92] **SPEC 92** — Technical Manual — Rev. 1.1, 1992

[Sun95] **SUN**, "UltraSparc: Next Generation Superscalar 64-Bit Sparc", *Compcon 95*, 1995

[Toma67] **R.M. Tomasulo**, "An efficient Algorithm for Exploiting Multiple Arithmetic Units", *IBM Journal, vol. 11*, January 1967.

[YePa92] **T.Y. Yeh** and **Y.N. Patt**, "A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution", *Proceedings of the 25th Annual Symposium on Microarchitecture*, Portland, Oregon, December 1992.