Widening Resources: A Cost-effective Technique for Aggressive ILP Architectures

David López, Josep Llosa, Mateo Valero and Eduard Ayguadé

Departament d'Arquitectura de Computadors. Universitat Politècnica de Catalunya.
Campus Nord Mòdul D6, Jordi Girona 1-3, 08034 Barcelona, SPAIN.
{ david | josepll | mateo | eduard }@ac.upc.es

Abstract

The inherent instruction-level parallelism (ILP) of current applications (specially those based on floating point computations) has driven hardware designers and compilers writers to investigate aggressive techniques for exploiting program parallelism at the lowest level. To execute more operations per cycle, many processors are designed with growing degrees of resource replication (buses and functional units). However, the high cost in terms of area and cycle time of this technique precludes the use of high degrees of replication. An alternative to resource replication is resource widening, that has also been used in some recent designs, in which the width of the resources is increased.

In this paper we evaluate a broad set of design alternatives that combine both replication and widening. For each alternative we perform an estimation of the ILP limits (including the impact of spill code for several register file configurations) and the cost in terms of area and access time of the register file. We also perform a technological projection for the next 10 years in order to foresee the possible implementable alternatives. From this study we conclude that if the cost is taken into account, the best performance is obtained when combining certain degrees of replication and widening in the hardware resources. The results have been obtained from a large number of inner loops from numerical programs scheduled for VLIW architectures.

1. Introduction

The architectural models implemented in current high-performance microprocessors are based upon hardware and software techniques to exploit the inherent instruction-level parallelism (ILP) of the applications. These models make use of deeper pipelines that reduce the cycle time and wider instruction issue units that allow the simultaneous execution of several instructions per cycle. As the number of transistors on a single chip continues to grow, more hardware can be accommodated on a chip. It is important to think of new processor organizations to take advantage of the additional transistors that will be available in the near future.

Very Long Instruction Word (VLIW) architectures are oriented to the exploitation of ILP. In a VLIW architecture, an instruction is composed of a number of operations that are issued simultaneously to the functional units (i.e. the scheduling is performed at compile time so the dispatch phase is very simple). Although there exists a few number of commercial VLIW machines, these architectures have been the subject of research in the last few years and will probably constitute the core of future designs [17].

The static nature of VLIW schedulings require good compilation techniques to effectively exploit the ILP available in real programs. Software pipelining [9] is a compilation technique that extracts ILP for the innermost loops by overlapping the execution of several consecutive iterations. In a software pipelined loop, the number of cycles between the initiation of successive iterations (termed *Initiation Interval*) is bounded either by the recurrences in the dependence graph or by the resource constrains of the target architecture [5, 20, 21].

The performance of loops bounded by resources can be improved by increasing the number of resources available in an architecture (*replication technique*). Using this technique we increase the number of operations that can be simultaneously executed over independent data. As an alternative to replication, the width of the resources can be increased (*widening technique* [11, 12]). Using this technique the same operation can be performed over multiple data. Both techniques can be combined in the same processor design.

The use of replication and widening allow us to have a scalable architecture in which we can add hardware to

0-8186-8609-X/98 $10.00 (c) 1998 IEEE

increase the number of operations performed per cycle. Although replication has been extensively used in the design of superscalar processors, only small degrees of widening have been applied to buses in some of them (IBM POWER2 [25]), and floating point units (FPU) in vector processors (NEC SX-3 [24]) and multimedia processors (AltiVec [18]).

This paper focuses on a cost-conscious evaluation of a broad range of VLIW processor designs in which several degrees of replication and widening are combined. For each configuration we evaluate the ILP achievable in the execution of a set of numerical programs. In order to perform a fair comparison of the different alternatives, it is mandatory to study the hardware costs for each configuration in terms of area and cycle time. These costs forces us to view the performance from a different perspective

The area cost defines those configurations that could be implemented in the next microprocessor generations, according to the predictions of the *Semiconductor Industry Association* [23]. For each generation we estimate the performance of a set of implementable configurations taking into account the number of cycles required to execute the programs and the cycle time. From this study we conclude that, for a given technology, the best performance is obtained when replication and widening are appropriately combined.

All the evaluations have been performed for VLIW architectures and numerical programs. Our workbench is composed of 1180 loops that account for 78% of the execution time of the Perfect Club [3]. The loops have been obtained using the experimental tool Ictíneo [2] and software pipelined using *Hypernode Reduction Modulo Scheduling* [15,16], a register pressure sensitive heuristic that achieves near optimal schedules. Register allocation has been performed using the wands-only strategy and the end-fit with adjacency ordering [22]. When a loop requires more than the available number of registers, spill code is added and the loop is rescheduled [14].

The organization of the paper is as follows: Section 2 describes the replication and widening techniques and outlines the advantages and drawbacks of both. Section 3 presents a study of the ILP achievable by both techniques: first we study the peak performance for a broad set of configurations (i.e. the performance assuming a perfect schedule and an infinite register file), and then the performance degradation due to the spill code added by the compiler when a limited register file is used. Section 4 describes the models used to estimate the area and cycle time cost of a configuration; these models are used in Section 5 to estimate the performance of the configurations under different technology limits. Finally, Section 6 summarizes the main conclusions of this work.

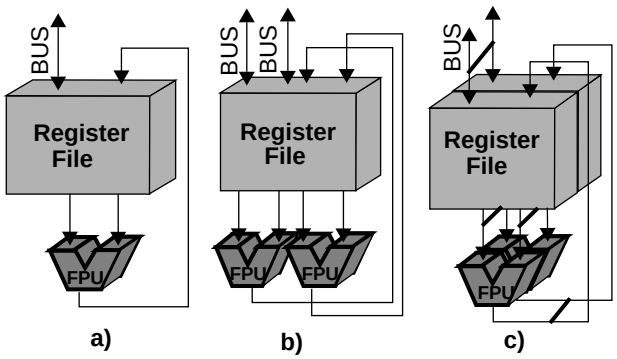

Figure 1: Several configurations: a) base configuration with 1 bus and 1 fpu. b) The same configuration after applying replication (i.e. 2 buses and 2 fpus) and c) the base configuration after applying widening (i.e. 1 bus and 1 fpu, both of width 2). Notice that the register file has been also widened.

2. Replication and widening: the techniques and their implications

In order to increase the number of operations performed per cycle, the resources of the processor must be increased. In this section we describe two possible techniques: replication and widening. Replication consists of increasing the number of resources by adding more independent functional units; widening consists of increasing the number of operations that a single functional unit can perform per cycle.

Figure 1 shows the use of replication and widening, both for buses and FPUs. Figure 1a shows a basic processor configuration with a single bidirectional bus and a single FPU. In this case we can perform one memory access and one operation per cycle. In order to issue two memory accesses and two operations per cycle, one could add another bus and another FPU (*replication*), as shown in Figure 1b. An alternative could be to duplicate the width of the bus, the register file and the FPU (*widening*) as shown in Figure 1c. In this case, two consecutive words in memory could be accessed and stored in a single register (of width 2), and one operation could be performed over registers of width 2. Replication is more versatile than widening: while replication can access to two independent words in memory or perform two independent operations per cycle, widening requires that the operations are *compactable* [12] in order to perform two operations in the same functional unit. For instance, two independent memory accesses with a stride different than one can be scheduled in the same cycle in a configuration with 2 buses; a configuration with a simple wide bus should schedule the two accesses in two different

cycles unless stride 1 appears. However replication has, in general, higher costs:

- Buses: widening affects the data bus, but not the control and address buses. On the other hand, replication increases all the buses between the register file and the first-level cache, so it requires more ports in the cache memory (multiported caches require more area and have more access time [8]). Also, widening requires the same number of address translations, while replication requires more addresses translated per cycle (this affects the number of ports of the TLB, causing an increase in cycle time and die area of the TLB [1])
- Register file (RF): in our proposal widening is applied to the buses, the FPUs and the register file. Every register in the RF increases its width in bits, but they have the same number of ports per bit. Applying replication increases the number of ports per bit. Both techniques increase the RF area and cycle time, but increasing the number of ports per bit has a higher cost than increasing the number of bits per register, as we show in Section 4.
- FPUs: both techniques require almost the same hardware at the FPU level, because we need to perform the same number of operations per cycle.

From the point of view of code generation, widening can reduce the total number of instructions (a single wide operation is the result of compacting multiple operations). This reduction of the code size can reduce the miss rate of the instruction cache and further improve performance.

3. Limits on ILP

In this section we study the maximum ILP achievable under optimal conditions and show the effect of the size of the register file. A baseline configuration, composed of 1 bus and 2 FPUs, all of width 1 (configuration named *1w1*) is considered. This configuration has the following characteristics: a store is served in 1 cycle; division and square root are not pipelined and require 19 and 27 cycles, respectively; the rest of the operations (load, add,...) are fully pipelined and require 4 cycles to be executed. In this study we increase the maximum number of operations performed per cycle by factors of 2, 4, 8, 16, 32, 64 and 128 using replication, widening or a combination of the two. The resulting configurations are labelled as *X*w*Y.* An XwY configuration has X buses and 2*X FPUs[1], all of them of width Y, and a register file in which every register has a width of Y words.

1.Preliminary studies show that a relation of 2 FPUs for each bus is the most balanced configuration. Also, we have based the cost calculations on the MIPS R10000, which can issue 2 floating point and 1 memory operation per cycle.

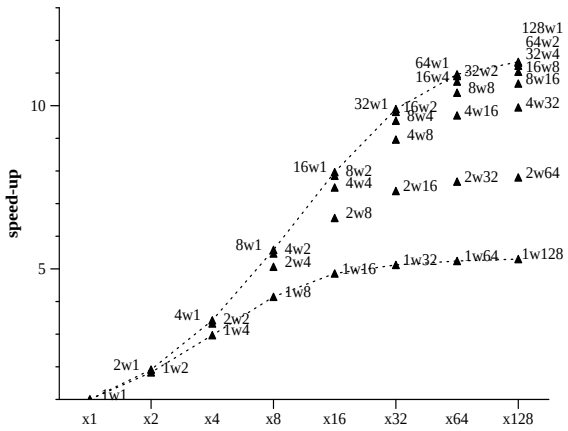

Figure 2: Speed-up for different configurations xwy.

3.1 Maximum ILP achievable

The parallelism exploitable from a loop is bounded both by the number of operations that have to be performed and by the recurrences of its dependence graph. For a processor configuration, loops can be classified into resource-bound loops or recurrence-bound loops. A resource-bound loop is a loop whose performance is limited by the resources available. A recurrence-bound loop is a loop whose performance is limited by the recurrences so. Therefore, this type of loop will have the same performance even with infinite resources.

Figure 2 shows the maximum performance achievable using the described techniques under optimal conditions: perfect scheduling, register file of infinite size and perfect memory. From these plots, several conclusions can be drawn:

- Configurations based on replication (i.e. configurations *Xw1,* upper plot in Figure 2) show a progressive performance degradation. This is because aggressive configurations can easily convert resource-bound loops into recurrence-bound loops, and these loops can not benefit by an increase in resources.
- Configurations based on widening (i.e. configurations *1wY,* lower plot in Figure 2) show even more performance degradation due to non-compactable operations. For instance, in a 1w8 configuration, either 8 compactable operations or 1 non-compactable operation can be issued per cycle; therefore, the presence of non-compactable operations introduces an enormous penalty on these configurations.
- Some of the intermediate configurations (i.e. configurations where replication and widening techniques are combined) also report good performance. For example, the behaviour of the 2wY

configurations saturates in the same way as the 1wY configurations, but the saturation point is close to a speed-up of 8 instead of 5. Also, the Xw2 configurations have performances very close to the Xw1 configurations.

3.2 The effects of spill code on performance

Increasing the number of operations that can be performed per cycle (by using either replication or widening) can reduce the Initiation Interval (II) of a loop. Regretfully, reducing the II can increase the register requirements [13]. If the registers required to schedule a loop on an architecture exceed the number of physical registers, spill code must be introduced in order to free some registers [14]. However, spill code increases the memory traffic and can result in an increase of the II, with the associated performance degradation.

When widening is applied, we have also a wide register file (i.e. in a XwY configuration, all registers are of Y words wide; in our case 64 bits times Y). For instance, a 32-RF 4w1 configuration can access to 32 registers of width 1 (i.e. 64 bits) while a 32-RF 4w2 can access to 32 registers of width 2 (i.e. 128 bits). Notice that if we schedule a loop with compactable operations in a 4w2 configuration, these operations produce 2 results that are stored in a single wide register, so we have an additional storage capacity; however, if the loop scheduled in the 4w2 configuration has no compactable operations, we do not benefit from this additional capacity.

Figure 3 shows the performance when spill code is taken into account. For each configuration, the loops have been scheduled assuming 32-, 64-, 128- and 256-RF and adding spill code when necessary[1]. Notice that the configuration 8w1 does not have the 32-RF bar. This is because a 8w1 configuration can produce 24 results per cycle (8 memory + 16 FPU), and we consider a 4-cycle latency configuration. In this case, the register pressure is so high, that the scheduler fails to produce a valid schedule with the available registers. This is the reason why we do not present, with this latency model, configurations with a factor higher than 8.

The results show that when the configurations become more aggressive, the need for spill code increases, reducing the performance. For example, configuration 4w2 has a performance of 2.25 (with 32-RF), 3.28 (64-RF), 4.39 (128-RF) and 4.76 (256-RF), while the 1w2 configuration achieves almost its maximum performance with a 64-RF.

1.The baseline is a configuration 1w1 with a RF of 256 register file because it does not require spill code, so it is similar to the baseline of Figure 2 (1w1 with an infinite RF).

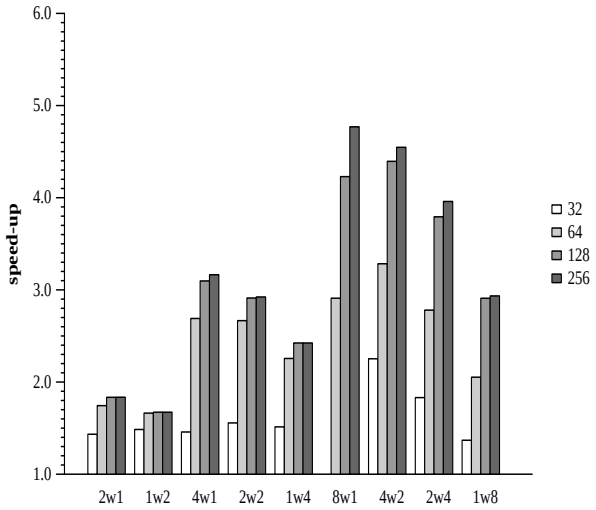

Figure 3: Performance/cost trade-off taking spill code into account for some configurations XwY with FPUs latency=4. Baseline: configuration 1w1 with a 256-RF.

The additional RF capacity of a configuration where widening has been applied reduces the need for spill code. For example the 8w1 configuration has a theoretical performance greater than the 4w2 configuration, but figure 3 shows that the configuration 4w2 with 64-RF has a performance greater than the 8w1 64-RF configuration, and the same happens with a 128-RF. Only with a 256-RF does the 8w1 have better performance than 4w2. Looking at the results, we can conclude that the additional capacity of the register file obtained when the widening technique is used, has an important impact on the final performance.

4. Design considerations

4.1 Area cost

To estimate the area cost, we take into account the SIA *Semiconductor Industry Association* predictions [23] for the technology size (λ) and the chip size in order to compute the number of λ^2 per chip for the next five generations (Table 1).

	1998	2001	2004	2007	2010
λ (μm)	0.25	0.18	0.13	0.10	0.07
Size (mm^2)	300	360	430	520	620
λ^2 per chip ($x10^6$)	4800	11111	25443	52000	126530
λ^2 / mm^2 ($x10^6$)	16	30.86	59.17	100	204.08

Table 1: Semiconductor Industry Association (SIA) predictions in 1994

We have estimated the area of a general purpose floating point unit (FPU) using the MIPS R10000 processor as a reference. The R10000 processor FPU includes a multiplier, an adder and a divider. We consider these components as the basic components of a general purpose FPU. With a 0.25 μm technology, the R10000 FPU requires 12 mm^2 of area [19]. So we assume the area of a FPU as 12 mm^2 x $16x10^6$ λ^2/mm^2 = $192x10^6$ λ^2.

The overall size of a register file is determined mainly by the size of a register cell. The other components that are needed to access the register file typically represent less than 5% of the area required by the register cells [10].

To access the register cell of a multiported RF, each port requires one transistor, a select line and a data line. In addition, a write port requires a second access transistor and a data line. The area of the register cell grows approximately as the square of the number of ports added because each port forces the cell to increase both the height and the width. Each additional port (read or write) adds one select line to the height of the cell. With respect to the width, each additional read port requires another data line and another access transistor, and each additional write port requires two data lines and two access transistors [7, 10, 11]. Table 2 shows the dimensions of several multiported register cells.

Ports	1R, 1W	2R, 1W	5R, 3W	10R, 6W	20R, 12W
W x H	50x41	64x41	162x81	316x145	568x257
Area (λ^2)	2050	2624	13122	45820	145976
Relative	1	1.28	6.4	22.35	71.21

Table 2: Dimensions of several multiported register cells.

To illustrate the cost difference between replication and widening, Table 3 shows the area cost of configurations 4w1, 2w2 and 1w4 for a 64-RF. Each configuration requires 2 read plus 1 write port per FPU and 1 read plus 1 write port per bus, so configuration 1w4 (2 FPUs and 1 bus, all of width 4) requires 5R+3W ports. Doubling the replication degree doubles the port requirements. Notice that these 3 configurations have the same area cost due to the FPUs (all of them require the same hardware to perform 4 floating point operations per cycle).

Configura-tion	Ports	Area of one memory cell (λ^2)	Bits per register	Total RF area (λ^2)
4w1	20R+12W	145976	64	$598x10^6$
2w2	10R+6W	45820	128	$375x10^6$
1w4	5R+3W	13122	256	$215x10^6$

Table 3: RF area cost for some configurations.

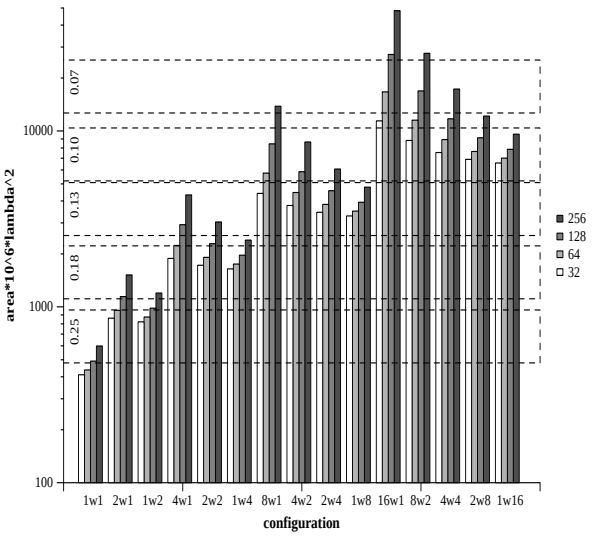

Figure 4: Area cost (register file plus FPUs)

Figure 4 summarizes the area cost of the different configurations tested. We assume that it is reasonable to use between 10% and 20% of the chip area for the functional units and the register file. These limits are shown by the five horizontal bands, one for each of the SIA predictions (notice that the vertical axis has logarithmic scale).

The area cost forces us to view the performance from a different perspective. For example, with a 0.10λ technology we can build a 8w1 configuration with 128-RF but not with 256-RF. The same technology allows us to build a 4w2 configuration with 256-RF, which has better performance than 8w1 128-RF (see Figure 3). The reader can find a more detailed study of the area requirements of these techniques in [11].

4.2 The register file access time

A multiported register file follows the scheme shown in figure 5. The access time model used in this paper is based on an adaptation proposed [6] for the register file of the CACTI memory model [26]. In the model, the access time of the register file is assumed to be governed by the read time, and can be written as the sum of the following terms:

- Decoder time: time to decode the register being accessed and select its wordline. This time depends mainly on the number of registers available.
- Wordline time: is the time required to drive the select line. It depends on the length of the line (that depends on the size in bits of every register and on the width of every register cell).
- Bitline time: is the time delay between the wordline going high and the sense amplifier being able to detect

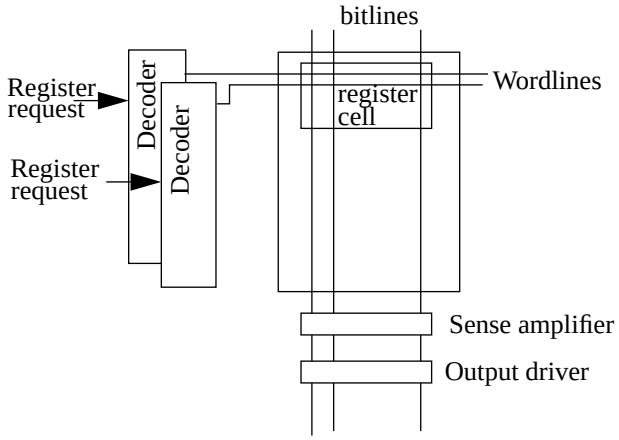

Figure 5: Multiported register file structure

the state of the cell. This delay depends mainly on the height of the cell and on the number of registers.

- Sense time: is the time delay through the sense amplifier.
- Outdrive time: is the time required to drive the read data onto the internal bus to the ALU.
- Precharge time: is the time to precharge the bitlines, comparators and internal decoder bus.

To summarize, the access time is governed by the number of registers, the width of every register and the size of every register cell (which depends on the number of read and write ports). Table 4 shows the relative access time for different configurations varying the number of registers (32, 64, 128 and 256), the width of every register (64 bits times the width degree) and the size of the register cell. In order to consider the access time independent of the technology used, all times have been normalized with respect to the time of the 1w1 32-RF configuration.

Configuration	Register file			
	32	64	128	256
1w1	1	1.05	1.18	1.34
2w1	1.49	1.54	1.70	1.87
1w2	1.10	1.15	1.29	1.45
4w1	2.44	2.51	2.69	2.90
2w2	1.65	1.72	1.87	2.06
1w4	1.22	1.27	1.43	1.60
8w1	4.32	4.41	4.61	4.87
4w2	2.75	2.82	3.00	3.23
2w4	1.85	1.92	2.09	2.29
1w8	1.39	1.45	1.62	1.80
16w1	8.04	8.15	8.39	8.72
8w2	4.89	4.99	5.20	5.48
4w4	3.10	3.18	3.38	3.61
2w8	2.12	2.20	2.38	2.60
1w16	1.68	1.75	1.93	2.14

Table 4: Relative register file access time (baseline: 1w1 32-RF)

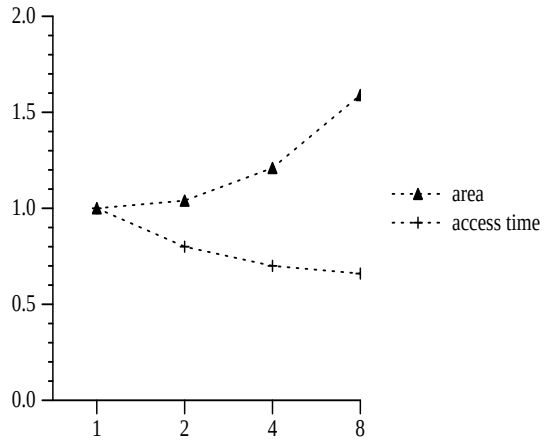

Figure 6: Behaviour of the RF area and cycle time of a 8w1 64-RF configuration with RF partitioning in 1, 2, 4 and 8 blocks.

To reduce the access time, a register file can be partitioned into several RF, maintaining copies of all the data [4]. For example, the RF of a configuration 8w1 can be implemented on a single RF where each cell requires 40 read plus 24 write ports (8R+8W for the 8 buses and 32R+16W for the 16 FPUs). This RF can be also implemented by two identical copies, where all functional units can write in both copies of the RF, but only 4 buses and 8 FPUs read each copy. In this case, 20R+24W ports are required for each copy. This means an increase of the RF die area, but the access time has been reduced. The configuration 8w1 can be partitioned in 1, 2, 4 or 8 blocks, having the relative area increase and cycle time decrease shown in Figure 6. Notice that the behaviour of area growth is exponential while the decreasing of the access time is logarithmic. A small partitioning, like a 2-partitioning has a slight increase in area and an important decrease in access time.

4.3 Code size

In a VLIW architecture, the instruction word has several atomic instructions (load, add,...). Using widening, one atomic instruction must specify several basic operations. For example, configuration 4w1 requires an instruction word long enough to fit 4 memory accesses and 8 floating point operations, while in configuration 2w2, the instruction word must fit up to 2 memory accesses and 4

Configu-	32 registers					64 registers					128 registers					256 registers				
ration	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
1w1	✓					✓					✓					✓				
2w1	✓	✓				✓	✓				☆	☆				☆	☆			
1w2	✓					✓					☆					☆				
4w1	☆	☆	☆			☆	❑	❑			❑	❑	❑			❑	❑	✻		
2w2	☆	☆				☆	☆				❑	❑				❑	❑			
1w4	☆					☆					☆					❑				
8w1	❑	❑	❑	✻		✻	✻	✻	✻		✻	✻	●	●		●	●	●	●	
4w2	❑	❑	❑			❑	❑	✻			✻	✻	✻			✻	✻	●		
2w4	❑	❑				❑	❑				❑	❑				✻	✻			
8w1	❑					❑					❑					❑				
16w1	●	●	●	●	×	●	●	●	×	×	×	×	×	×	×	×	×	×	×	×
8w2	✻	✻	✻	●		●	●	●	●		●	●	●	×		×	×	×	×	
4w4	✻	✻	✻			✻	✻	✻			●	●	●			●	●	●		
2w8	✻	✻				✻	✻				✻	✻				●	●			
1w16	✻					✻					✻					✻				

Table 5: Implementable configurations with a technology of 0.25 (✓), 0.18 (☆), 0.13 (❑), 0.10 (✻) and 0.07 (●). The white cell means that this RF partitioning is not applicable to this configuration. Symbol × means not implementable with any of the considered technologies.

floating point operations (but in the best case, both can perform the same number of basic operations). So the instruction length required by configuration 4w1 is 2 times the length required by configuration 2w2 and 4 times the length required by configuration 1w4. However, configuration 2w2 can require more instructions to perform a loop than configuration 4w1 because it is less versatile.

Figure 7 shows a comparison between the code size of different configurations that have the same peak performance. This is an extra advantage of the widening technique even though it does not affect our study because we consider a perfect memory.

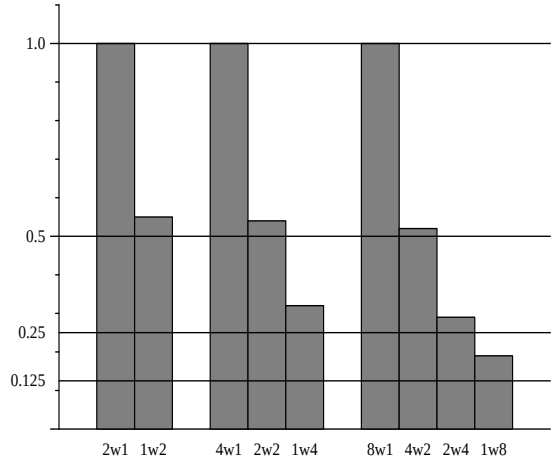

Figure 7: relative code size comparison

5. Performance/cost trade-offs under a technology limit

In this section we present a study of the best configurations for some technology generations considered. The methodology used is the following: we first select the configurations that can be implemented for each technology according to the SIA predictions and the area models in section 4. For each implementable configuration XwY(Z:n) (i.e. X buses and 2*X FPUs, all of width Y, with a RF of Z registers of width Y, partitioned in n-blocks) we calculate its cycle time, assuming that the cycle time is the RF cycle time. We adapt the latency of the FPUs to match this cycle time and we perform the scheduling to find the cycles required[1]. The cycles required to execute all the loops times the cycle time give us the final performance.

5.1 Implementable configurations for a given technology

We consider that a configuration is implementable, for a given technology, if the area cost of the FPUs and the register file is smaller than 20% of the total chip area available (see table 1). We have calculated the area for 32-, 64-, 128- and 256-RF, with all possible partitions. Table 5 shows the implementable configurations for each technology generation. A generation can implement all the configurations implementable by the previous generation plus the new ones listed in the table.

1.The cycles required are calculated as the cycles per iteration (Initiation Interval) times the number of iterations performed in the original loop execution.

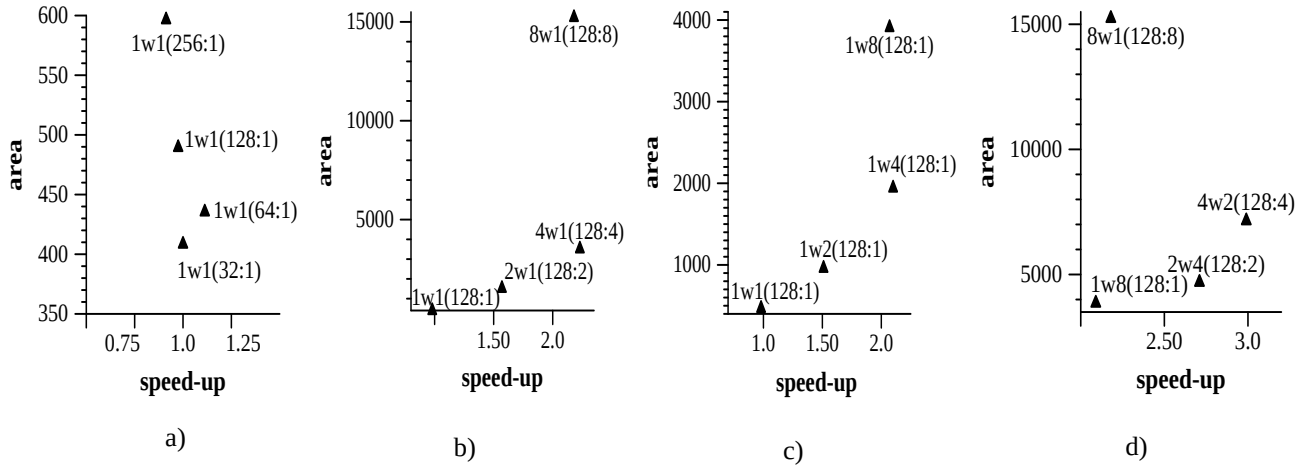

Figure 8: effects of a) increasing the number of registers in the RF b) replication, c) widening and d) different ways to implement a configuration with a peak performance of 8. All areas are in millions of λ^2.

5.2 The floating-point units latency

Each FPU requires an amount of time to perform one operation; its latency in cycles depends on the processor cycle time. This is important because it determines the scheduling of the loops. For this reason, we propose to compare configurations adapting the latency of the FPUs to the processor cycle time. The 4 cycle models we have tested are listed in table 6.

cycle model	cycles of operations			
	store	+,*, load	div	sqrt
4-cycles	1	4	19	27
3-cycles	1	3	15	21
2-cycles	1	2	10	14
1-cycle	1	1	5	7

Table 6: Cycles/operation for the cycle models tested. Operations div and sqrt are not pipelined, other operations are fully pipelined.

We assumed the 4-cycles model for configuration 1w1. Each configuration considered can be classified into a cycle model depending on its relative (from the 1w1 configuration) cycle time. A configuration with a relative cycle time Tc belongs to the z-cycles model, where $z=\lceil 4/Tc \rceil$. For example, the 2w4(32:1) configuration has a relative cycle time of 1.85 (i.e. 3-cycles model) while 2w4(128:1) configuration has a relative cycle time of 2.09 (i.e. 2-cycles model); and 2w4(128:2) configuration has a relative cycle time of 1.80 (i.e. 3-cycles model).

5.3 Performance evaluation

In this section we first discuss the individual effects of some parameters on the configurations evaluated. After this discussion, we show the best configurations for each technological limit. In all cases, we use a fixed timing model based on technology parameters for λ=0.25. We do not attempt to factor-in reductions in cycle time due to future technology generations.

Figure 8a shows the performance/cost ratio when we increase the number of registers available in the register file, for configuration 1w1. This configuration has negligible need for spill code when a 64 (or bigger) RF is available, so an increase of the RF does not affect the cycles required, but increases the RF cycle time. For this reason, the performance of RFs greater than 64 registers declines.

Figure 8b shows the performance/cost ratio when only replication is applied. Notice that a small degree of replication produces good performance (better than widening when we must schedule loops with non-compactable operations). However, configurations with a high degree of replication can become unimplementable (they occupy more than 20% of the total chip area) or suffer a decrease in performance because a small increase in IPC (instructions per cycle) is counteracted with a high increase of cycle time (like configuration 8w1 in Figure 8b).

On the other hand, Figure 8c shows the performance/cost ratio when only the widening technique is applied. A small degree of widening produces an increase in the area cost and cycle time that is smaller than what was noted with the replication technique. Also, the

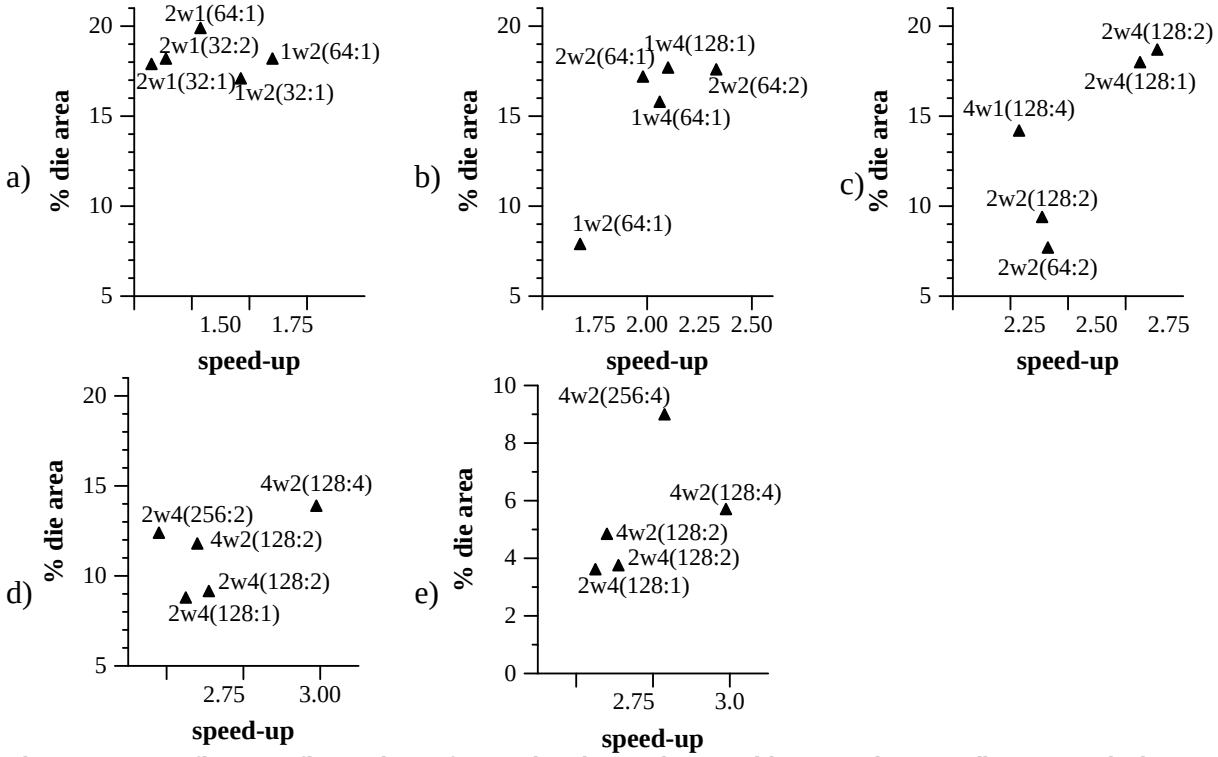

Figure 9: Top five configurations for technology a) 0.25, b) 0.18, c) 0.13, d) 0.10 and e) 0.07. In all cases, the possible increment of the clock speed has not been taken into account

increase of storage capacity due to applying the widening technique to the register file reduces the need for spill code, resulting in a good performance ratio. Nevertheless, there is a point at which the increment in IPC is so small that performance degrades due to increases in cycle time (like configuration 1w8 in figure 8c).

We conclude that combining a small degree of replication and widening can result in the best performance. This point can be observed in Figure 8d, which shows several configurations with the same peak performance.

Figure 9 shows, for each technology, the five configurations that achieve the best performance. In each plot of Figure 9, none of the most aggressive configurations are in the top-five configurations due to their high cost.

6. Conclusions

The inherent ILP of numerical applications requires an increase in the number of operations that can be performed per cycle. Two alternatives have been studied in this paper: replication and widening, for which we have done a performance/cost study. The results have been obtained using a large number of software pipelined loops from the Perfect Club benchmarks assuming a VLIW architecture.

We have proposed to combine replication and widening in the design of VLIW processors. We applied widening to the floating-point functional units, the register file and the buses between the register file and the first-level data cache. We have presented a study of the ILP limits of each configuration in optimal conditions, concluding that applying only the replication technique offers the best theoretical performance. However, when a limited register file is used the increase of storage capacity due to wider registers can reduce the need for spill code. The results show that this additional capacity has an important impact on the final performance (e.g. with a 128-RF, configuration 4w2 achieves better performance than configuration 8w1, whereas the latter has the best theoretical performance).

Taking into account that the replication technique is more expensive than the widening technique in terms of area cost and cycle time, we have estimated the cost of the configurations considered. We compare the performance of the configurations that can be built in the next processor generations (according to the SIA predictions). The performance has been calculated using the register file cycle time. To perform a realistic comparison, the RF cycle time has been reduced using the RF partitioning technique and the FPUs latency has been adapted to the cycle time.

From this study we conclude that, for a given technology, the best performance is obtained when combining a small degree of replication and widening in the hardware resources. For example, a 4w2 configuration with a 128-RF offers a speed-up of 1.66 with respect to a 8w1 configuration with a 128-RF, and occupies only 81% of the area.

Acknowledgements

The authors would like to thank the anonymous reviewers for their fruitful comments that undoubtedly have contributed to improve the quality of this paper. This work has been supported by the Ministry of Education of Spain under contracts TIC 429/95 and TIC 98-0511, by CEPBA (European Center for Parallelism of Barcelona), and by ESPRIT LTR Project No 24942 (MHAOTEU).

References

[1] T.M. Austin and G.S. Sohi. High-bandwidth address translation for multiple-issue processors. In *Proc. of the ISCA-23*, pp 158-167. May 1996.

[2] E. Ayguadé, C. Barrado, A. González, J. Labarta, J. Llosa, D. López, S. Moreno, D. Padua, F. Reig, Q. Riera and M. Valero. *Ictíneo: A tool for Instruction-Level Parallelism Research*. Res. Rep. UPC-DAC-1996-61. December 1996.

[3] M. Berry, D. Chen, P. Koss and D. Kuck. *The Perfect Club benchmarks: Effective performance evaluation of supercomputers*. Tech. Rep. 827, CSRD, U. of Illinois at Urbana-Champaign, November 1988.

[4] A. Capitanio, N. Dutt and A. Nicolau. Partitioned register files for VLIWs: A preliminary analysis of tradeoffs. In *Proc. of the MICRO-25*, pp 292-300, 1992.

[5] J.C. Dehnert and R.A. Towle. Compiling for Cydra 5. In *Journal of Supercomputing*, 7(1/2):181-227, 1993.

[6] Keith I. Farkas. *Memory-System Design Considerations for Dynamically-Scheduled Microprocessors*. PhD thesis, Dep. of Elec. and Comp. Eng., U. of Toronto, 1997.

[7] R. Jolly. A 9-ns 1.4 gigabyte 17-ported CMOS register file. *IEEE Journal of Solid-State Circuits*, V. 25(10):1407-1412, October 1991.

[8] T. Juan, J.J. Navarro and O. Temam. Data caches for superscalar processors. In *Proc. of the ICS-11*, pp 60-67. July 1997.

[9] M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proc. of the PLDI-88*, pp. 318-328, June 1988.

[10] Corinna G. Lee. *Code Optimizers and Register Organizations for Vector Architectures*. Ph. D. Thesis. U. of California at Berkeley. May, 1992.

[11] D. López, J. Llosa, M. Valero and E. Ayguadé. Resource widening vs. replication: Limits and performance-cost trade-off. In *Proc. of the ICS-12*, pp 441-448. July 1998.

[12] D. López, M. Valero, J. Llosa and E. Ayguadé. Increasing memory bandwidth with wide buses: Compiler, hardware and performance trade-off. In *Proc. of the 11th. Int. conf. on Supercomputing (ICS-11)*, pp 12-19. July 1997.

[13] J. Llosa, E. Ayguadé and M. Valero. Quantitative evaluation of register pressure on software pipelined loops. In *Int. Jour. of Parallel Programming*, vol. 26 n. 2 pp. 121-142. 1998

[14] J. Llosa, M. Valero and E. Ayguadé. Heuristics for Register-Constrained Software Pipelining. In *Proc. of the MICRO-29*, pp. 250-261, Dec 1996.

[15] J. Llosa, M. Valero, E. Ayguadé and A. González. Hypernode Reduction Modulo Scheduling. In *Proc. of the MICRO-28*, pp 350-360, 1995.

[16] J. Llosa, M. Valero, E. Ayguadé and A. González. Modulo Scheduling with reduced register pressure. In *IEEE Trans. on Computers*, vol. 47 n. 6 pp 625-638, June 1998.

[17] Microprocessor Report vol 11, no. 14. *Intel HP make EPIC disclosure*. October 1997.

[18] Microprocessor Report vol 12, no. 6. *AltiVec vectorizes PowerPC*. May 1998.

[19] K. Olukotun, B.A. Nayfeh, L. Hammond, K. Wilson and K. Chang. The Case for a Single-Chip Multiprocessor. In *Proc. of the ASPLOS-VII*, pp 2-11, October 1996.

[20] B.R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. MICRO-27*, pp. 63-74, November 1994.

[21] B.R. Rau and C.D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In *Proc. of the 14th Ann. Microprogramming Workshop*, pp. 183-197, October 1981.

[22] B.R. Rau, M. Lee, P. Tirumalai, and P Schlansker. Register allocation for software pipelined loops. In *Proc. of the PLDI-92*, pp. 283-299, June 1992.

[23] Semiconductor Industry Association. The National Technology Roadmap for Semiconductors. Semicond. Ind. Assoc. , San Jose, California 1994.

[24] T. Watanabe. The NEC SX-3 supercomputer system. In *CompCon91* pp. 303-308, 1991

[25] S.W.White and S. Dhawan. POWER2: Next Generation of the RISC System/6000 family. *IBM J. Res. Develop.* 38 (5), 493-502. September 1994.

[26] S.J.E. Wilton and N.P. Jouppi. CACTI: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, Vol. 31(5):677-688, May 1996.