A Path to Complexity-Effective Wide-Issue Superscalar Processors

André Seznec

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Path to Complexity-Effective Wide-Issue Superscalar Processors

André Seznec

N°4242

Septembre 2001

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4242--FR+ENG

INRIA
RENNES

A Path to Complexity-Effective Wide-Issue Superscalar Processors

André Seznec

Thème 1 — Réseaux et systèmes
Projet CAPS

Rapport de recherche n° 4242 — Septembre 2001 — 23 pages

Abstract: The advance of integration allows implementation of very wide issue superscalar processors on a single chip. Aggressive speculative execution as well as simultaneous multithreading can exploit such wide issue superscalar processors. Unfortunately, with the increase of issue width, processor designers are facing new difficulties to enable high clock frequency and to master silicon area and power consumption.

Due to performance issues, when doubling the issue width from 4 to 8 instructions per cycle on a superscalar processor, one has also to double the number of physical registers. Combined with the doubling of the number of register ports, this leads to an eight fold increase of the silicon area devoted to the register file on conventional monolithic register file architecture while the silicon area devoted to functional units only doubles. At the same time, the peak power consumption of the register file also raises quasi-quadratically with the issue width. Moreover, read operations on the register file have to be deeply pipelined. Wake-up logic as well the bypass network in the processor are also becoming limiting factors when the issue width increases.

In this paper, we present three mechanisms to reduce the number of read and write ports *on every individual physical register* in a wide-issue clustered superscalar processor, respectively limited read port arbitration, register write specialization and register read specialization. Then we show that combining register write specialization and register read specialization, one can build a 8-way 4-cluster superscalar processor where each individual physical register is implemented as four identical (2-read, 2-write) registers instead of a single copy (16-read, 8-write) register in conventional designs. This dramatically reduces the silicon area, the peak power consumption and the access time of the register file. As a side effect, the complexities of the bypass network and of the wake-up logic are also significantly reduced. In particular, fast-forwarding is simplified on a 8-way 4-cluster processor. Limited read port arbitration can be used to further reduce the complexity of the register file.

Such a complexity reduction can not come for free, but only costs some degrees of freedom on the policy for allocating instructions to clusters and some extra complexity in the register renaming process.

Key-words: Superscalar processors, register file, wake-up logic, bypass network, silicon area, power consumption

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Vers l'implémentation de processeurs superscalaires très larges

Résumé : Dans ce rapport, nous proposons de nouvelles directions pour simplifier la réalisation de processeurs supercalaires très larges (c-à-d séquençant 8 instructions ou plus par cycle.

Mots-clé : Processeur superscalaires, fichier de registres, surface de silicium

1 Introduction

The path for performance on general-purpose superscalar processors combines instruction level parallelism, very aggressive speculative execution and likely simultaneous multithreading (SMT) [12] while pushing clock frequency, but also mastering power consumption and silicon area. The physical register file, the bypass network and the selection and wake-up logic [9] are major obstacles for high clock frequency. Moreover their relative contributions to the silicon area and the power consumption increases dramatically with the issue width.

In this paper, we show that clustered superscalar architectures offer several opportunities to efficiently address these issues through a reduction of the number of read and write ports on *each individual register*.

We point out that since most data are provided by the data bypass, one can limit the number of read ports on the register file provided that some arbitration is performed for the read ports of the register file. While global read port arbitration cannot be realistically implemented on a wide-issue superscalar processor, **limited read port arbitration** can be implemented on a smaller cluster (e.g., 2-issue cluster).

Unlike some VLIW ISAs (e.g., Multiflow [2]), the ISAs used on the overwhelming majority of general-purpose computers feature a single logical general-purpose register file, and generally a second register file for floating-point registers. This explains that up to now all superscalar processor designs have applied the following unwritten rule :

> *every general-purpose physical register can be the source or the result of any instruction executed on any integer functional unit.*

Transgressing this rule allows us to limit the register file complexity. We point out that, distinct clusters can be forced to write in distinct subsets of the register file, thus allowing to reduce the number of write paths on each individual register. We refer to this principle as **register write specialization**. In particular, register write specialization makes the use of distributed (multiple copies) register files a complexity-effective alternative to the use of a centralized monolithic register file. One can also constrain the clusters to read their operands in only a subset of the register file, provided that every instruction remains executable by, at least, one of the clusters. We refer to this method as **register read specialization**.

The combination of register write specialization and register read specialization is very attractive. We refer to clustered architectures implementing this combination as WSRS (for register Write Specialization, register Read Specialization) architectures. It is possible to build the register file of a k-cluster ($k \leq 7$) WSRS architecture at a surprisingly low complexity: assuming 2-way clusters, each physical register can be implemented using 4 copies of a (2-read,2-write) register instead of a single copy ($4k$-read, $2k$-write) register or k copies of a (4-read, $2k$-write) register for conventional processors[1]. Moreover, the complexity

[1] As an assumption in the remainder of the paper, we will assume that any instruction or micro-operation presented to the execution core has at most two register operands and produces at most one register result. This assumption will be justified in Section 2

of the bypass point (respectively wake-up logic entry) is equivalent to the complexity of the bypass point(respectively wake-up logic entry) of current 4-way issue superscalar processors.

Such a complexity reduction does not come completely for free, but at a surprisingly limited cost. Some degrees of freedom on the policy for allocating instructions to clusters disappear and some extra hardware complexity is added for register renaming.

Paper organization

The remainder of the paper is organized as follows. In Section 2, we present and justify a few architectural assumptions we made for this paper. Section 3 summarizes previous related works on optimizing physical register file organizations and clustered architectures for out-of-order execution processors. In Section 4, we study the complexity of the register file in superscalar processors. Limited read port arbitration, register write specialization and register read specialization are presented in Section 5. In Section 6, we present and analyze a 4-cluster WSRS architecture. For this architecture, we first analyze the extra complexity in the register renaming process and the extra constraints on the policy for allocating instructions to clusters. Then we detail the benefits of combining register write specialization and register read specialization on the complexity of register file and on the complexities of the wake-up logic and of the bypass network, particularly the fast-forwarding capability.

Section 7 shows that the WSRS architecture scales up to 7 2-way issue clusters using only 4 identical (2-read,2-write) registers to implement a physical register. We further show that it is possible to design even wider issue processors while still mastering register file, bypass network and wake-up logic complexities. Finally, Section 8 summarizes this study.

2 Some architectural assumptions

Two operands, one result operations Throughout this paper we will assume that any instruction or micro-operation executed by the execution core reads at most two general purpose or floating-point register operands and writes at most one general purpose or floating point register operand result. More exactly, we will assume that all the dynamic pipeline stages ranging from renaming logic, dispatch, execution and write back treat only this kind of instructions or micro-operations.

Alpha and MIPS ISAs feature this property. Through decoding and splitting instructions in micro-operations, recent Intel IA32 implementations also respect this constraint. Translating instructions in two operands, one result micro-operations could also be applied to other ISAs (even RISC ISAs).

Fixed register read cycle and fixed write back cycle We assume that all kind of instructions read their operands on the same cycle in the pipeline. Furthermore, we also assume that the Write Back is performed on a fixed cycle for all instructions: for one-cycle latency operations, the Write Back is delayed to match the longer latency of other instructions (e.g., 2 or 3 cycles for loads)[2] Therefore, if a maximum of N instructions can be issued per cycle then N write ports can support these writes without any arbitration.

3 Related work

Previous research work on improving access time, silicon area and power consumption of the register file includes virtual-physical register file (limits the number of physical registers) , register caching (caching the critical registers) and use of clustered architectures. Using

[2]The alternative, i.e. direct early write back in the register file leads either to an increase of the number of register write ports or to arbitration for register write ports.

several clusters of functional units also addresses the complexity bypass network as well as the complexity of the wake-up logic.

Virtual-Physical registers Monreal et al. [8] proposed to delay the allocation of the physical register until instruction execution or even result write back. The renaming of registers is replaced by the allocation of a virtual stamp which does not connect directly with any physical location. A physical register is associated with the virtual stamp at instruction execution (or result write back). This solution allows to reduce the number of required physical registers, therefore to reduce the silicon area of the register file and its power consumption. This approach attacks the number of physical registers necessary in a superscalar processor. It is orthogonal to our approach which reduces the cost of each individual physical register.

Register caching Cruz et al. [3] remarked that not all physical registers have to be accessible on the very next cycle. Many physical registers are not even ever read since they are used only once and they are captured through the bypass network. They proposed to use a register file cache. Only registers likely to be useful in the very next cycles are written in the register file cache. A complete register file copy is maintained, but it can feature a longer access time as well as fewer read ports. This organization allows a low latency register access while supporting a large number of physical registers. Register caching might be used to reduce register access time in combination with our proposals.

Clustered Microarchitectures Palacharla et al [9] studied critical paths in wide-issue superscalar microprocessors. The register file access time, but also selection and wake-up logic and the data bypass logic were identified as critical paths.

In order to reduce these critical paths, they proposed to use a 2-cluster architecture (Figure 1). 2N functional units are grouped in two identical symmetric clusters of N units. Two local copies of the register file are maintained in order to enable short access time. The results from instruction executed on Cluster C0 are not forwarded on the next cycle to functional units in Cluster C1, since the transit delay would exceed one cycle. The instructions are dispatched either to Cluster C0 or to Cluster C1 early in the pipeline, therefore two half-size instruction windows are maintained. Selection logic is also replicated and its complexity is largely inferior to a centralized instruction window: one has to chose N instructions out of W/2 instructions instead of 2N out of W.

The Compaq Alpha 21264 processor [7] implements this technic.

Several other authors have studied clustered architectures. Baniasadi and Moshovos [1] studied policies for dynamically distributing instructions on a quad-cluster architecture.

Farkas et al [5] proposed to use two distinct physical register files each of them associated with a subset of the ISA logical registers. Each physical register file is associated with a cluster of functional units. The main difficulty with this approach is that whenever an instruction uses two logical operands mapped to the two distinct subsets of the register file, moves have to be generated by the hardware between the two physical register files. The load balancing of the clusters is also very sensitive to code generation. In some sense this work is very close to our proposals, since the unwritten rule we cited in the introduction is also broken.

VLIW ISAs such as Multiflow [2] or more recently LX [4] implement distinct logical register files that are accessed by different clusters of functional units. Whenever the different operands for an instruction lie in different register files, the compiler is responsible to generate moves between the register file to allow the execution of the operation. This allows to implement wide-issue processors while using register files with a limited number of read and write ports. Our proposal also enables the implementation of wide-issue processors the use of physical register files with a small number of read and write ports for ISAs featuring a *single* logical register file.

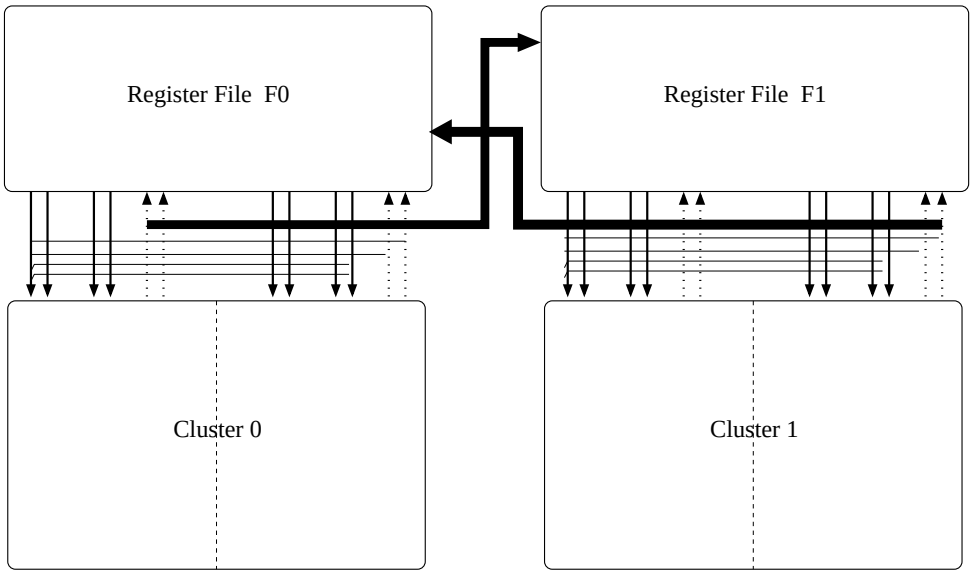

Figure 1: Palacharla et al.[9] 2-cluster architecture

4 Complexity of a register file in a wide-issue superscalar processor

In this section we analyze the factors of complexity in the design of multiported register files in terms of silicon area, access time and peak power consumption.

4.1 Influence of the issue width

Large number of ports The general-purpose register file is central in a superscalar processor, since all the functional units read their operands and write their results onto it. Assuming that all operands can be read onto it and that all results must be written onto it, the number of access ports increases linearly with the number of instructions that may be issued on every cycle.

Large number of registers Many factors convey to increase the numbers of physical registers that are needed on every new generation of superscalar processors. First, wider issue leads to the need for wider instruction windows for parallelism extraction. Second, since instructions are validated in-order, many physical registers have to remain alive while their last read has already been issued, but has not been validated. Simulations results illustrating [6] (assuming a short six-cycle pipeline) indicated that, assuming precise exception, a 100-entry general-purpose integer register file would be sufficient for a 4-way issue processor, but that more than 150 registers would be needed for a 8-way issue processor. Third, in order to reach very high clock frequency, one has also to deepen the pipeline. While the depth of the overall pipeline is increasing, the life of the physical registers is also lengthening: x extra cycles on the pipeline length after register renaming may lead to maintain $x * N$ extra physical registers alive on a N-way superscalar processor.

4.2 General complexity of a multiported register file

4.2.1 Single centralized register file or distributed register file

Using two or more identical copies of the register file allows to use register files with a lower number of read ports. In a clustered architecture, a local copy of the register file may be

associated with each cluster, therefore the registers are close to the functional units. The register read access time is shorter than on a monolithic centralized register file since 1) each copy of the register file is smaller and 2) the distance (and therefore the transit delay) from the register file to the functional units is shorter.

4.2.2 Silicon area

The silicon footprint of a multiported register file is dominated by the area devoted to memory cells [14]. When the number of ports is high, the size of a register cell is approximately a quadratic function of the number of ports on the register file [11]. In a conventional multiported memory cell featuring N_{read} ports and N_{write} ports, N_{read} bitlines, N_{read} wordlines wires and $2N_{write}$ bitlines and N_{write} wordlines wires must cross the cell [14]. w being the width of each wire (i.e. the width of the wire itself plus the distance with the neighbor wire), the area devoted to the cell is given by:

$$w^2 * (N_{read} + N_{write}) * (N_{read} + 2 * N_{write}) \quad (1)$$

At equal numbers of registers, a (16-read, 8-write) register file will roughly occupy 4 times the area of a (8-read, 4-write) register file and 16 times the area of a (4-read, 2-write) register file !

4.2.3 Power consumption and access time

In order to evaluate the peak power consumption and the access time for multiported register files[3]. , we use the CACTI2.0 package [13]. Since CACTI2.0 is devoted to evaluate power consumption and access time on caches, we discarded the tag path in the measures presented here.

In Table 1, we report results for configurations approximately representative of the needs in number of physical registers for respectively 12-way, 8-way and 4-way superscalar processors. Both centralized and distributed register files are considered: C represents the number of distinct physical register files, S being the total number of distinct registers, R the number of read ports and W the number of write ports. Due to the 4-6 year microprocessor design cycle, current research propositions cannot appear in products before 2006-2008. Therefore we present this evaluation for a two generation ahead technology CMOS $0.10\mu m$[4] and a 5 Ghz clock. If the current trend in the increasing of clock frequency continues then one can reasonably expect to achieve frequencies in the 5 Ghz range using CMOS $0.10\mu m$.

For computing this pipeline depth, we assume a single extra pipeline cycle for driving the data from the exit of the register file to the entry of the functional units when the register file is distributed and two extra pipeline cycles when the register file is centralized.

Table 1 also reports the silicon area devoted to represent a single bit of a physical register and the relative silicon footprint of the register file compared with a 128-entry (8-read, 4-write) register file.

Analysis Table 1 clearly indicates that a monolithic centralized register file for a 12-way or 8-way superscalar processor will not represent a valid approach in terms of peak power consumption (around 100 W for a 12-way processor) as well as in terms of pipelining. Using a distributed (i.e multiple copies) register file allows to slightly reduce both the peak power consumption and access time, but using more than two copies of the register file results in an increase of the silicon area.

[3]The reported power consumptions are peak power consumptions. In a processor, many data are captured on the bypass data path and effective register reads are not useful. The information on the presence/absence and the producer of a register operand on the bypass network has to be maintained in the wake-up logic. An immediate power consumption optimization consists in using this information to initiate only the reads on the register file when they are useful.

[4]"The technology scaling in CACTI 2.0 should work well down to below $0.1\mu m$." Norm Jouppi, March 2001, private communication

C*(S,R,W)	1*(384, 24, 12)	3*(384,8,12)	2*(256, 16, 8)	2*(256,8,8)	4* (256, 4 , 8)	1*(128,8,4)
nJ per cycle	21.07	15.37	8.01	6.72	6.37	2.19
Power at 5 Ghz (W)	105.35	76.95	40.05	33.60	31.85	10.95
Access time (ns)	1.51	1.02	0.95	0.77	0.65	0.53
Pipeline cycles	10	7	7	5	5	4
Reg. bit area (x w^2)	1728	1920	768	768	960	192
Relative size	27	30	8	8	10	1

Table 1: Estimates for different register file configurations

This clearly points the challenges that architects and designers are facing with register files in wide-issue superscalar processors. When doubling the issue width from 4 to 8, the total silicon area and the total power consumed by the functional units double while the latency of instructions remains constant. For the register file, the silicon footprint is multiplied by eigth (may be 10), the power consumption is at least tripled and extra pipeline stages are encountered on the register file access.

5 Three directions for reducing the number of ports on each individual register

In this section, we present three orthogonal methods to reduce the number of ports on each individual register on a clustered superscalar processor. **Limited read arbitration** and **register read specialization** aim at reducing the number of read ports on the registers. **Register write specialization** reduces the number of write ports on each individual register.

5.1 Limited read port arbitration

In a wide-issue superscalar processor, many of the operands are not furnished by the register file, but are directly captured on the bypass network. Therefore, there is an opportunity to limit the number of read ports on the register file if only operands really read from the register file busy a read port.

However, this induces dynamic arbitration on the register file read ports. This arbitration must take place on the same cycle as the wake-up and selection. To the best of our knowledge, no processor design implementing such an arbitration on the register read ports has been disclosed.

Nevertheless, for a 2-way issue cluster, a limited read port arbitration would be possible to implement as follows: 2 independent arbitrations are performed respectively for the first operand and the second operand. The first instruction has priority and therefore is always executed. That is the second instruction is executed only when it is able to get its operands, that is, for each of the inputs of the instruction, the operand is either obtained directly from the bypass network or, the first instruction did not use the register port and the second instruction can use it.

Even such a simple arbitration scheme will slightly lengthen the selection logic. However for the WSRS architecture we describe in Section 6, the wake-up logic is simpler than on conventional architecture (see 6.5.4): one might then invest the saved delay in limited read port arbitration.

Applying this limited read port arbitration on a 8-way 4-cluster processor allows to use a (8-read, 8-write) register file instead of a (16-read, 8-write) register file. Column 2*(256,8,8) in Table 1 provides estimates for 2 copies of such a register file: assuming a centralized register file, limited read arbitration would allow to more than halve the peak power consumption and also to halve the silicon area.

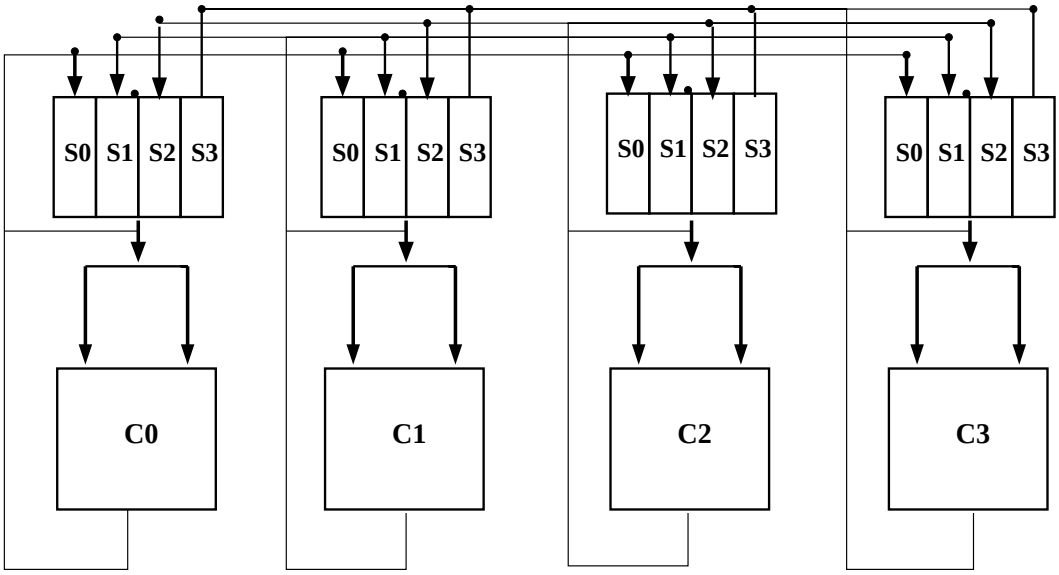

Figure 2: Register write specialization on a 4-cluster architecture

5.2 Register write specialization

From Formula 1, the number of write ports on a register has a dramatic impact on its silicon area.

On conventional architecture, each individual register in the physical register file can be written as the result of any functional unit. Register write specialization consists in limiting the number of possible writers to each individual register. On a clustered architecture, this can be implemented as follows:

- The physical register file is partitioned in several distinct subsets of registers.
- Each of the clusters writes only in one of the subsets.

A 4-cluster architecture featuring a distributed register file and register write specialization is illustrated on Figure 2. Assuming 2-way clusters, each physical register will be built using four (4-read, 2-write) identical registers instead of four (4-read, 8-write) identical copies when register write specialization is not implemented.

5.2.1 Distributed register file versus centralized register file

Using a distributed register file with a conventional approach increases the total silicon area devoted to the register file (see columns 1*(256,16,8) and 4*(256,4,8) in Table 1). Using register write specialization reverses this phenomenom as illustrated by the following example.

From Formula 1, we can derive that on a 8-way 4-cluster processor using a distributed register file, the four (4-read, 2-write) register cells needed to represent a physical register bit will occupy a total silicon area of $192 * w^2$ instead of $360 * w^2$ for the single (16-read , 2-write) register cell of a centralized register file (and $768 * w^2$ for a (16-read, 8-write) register cell or $960 * w^2$ for four (4-read, 8-write) register cells when register write specialization is not used !).

5.2.2 A deadlock issue and its turn-arounds

There is a possible deadlock issue when using register write specialization in the particular case where the number of physical registers in each of the register subset is smaller than the

number of logical registers in the ISA. For instance, one could construct a situation where, on a given point in the program all the logical registers are mapped onto a single subset.

This deadlock will not occur if the register subsets feature at least the same number of registers as the number of logical registers in the ISA. However, for SMTs or for ISAs featuring very large numbers of registers (e.g., IA64) or when three or more register subsets are considered, this might not be a realistic solution. Two possible turnarounds can be considered:

- The allocation of instructions to clusters may be in charge of avoiding this deadlock.
- An exception is raised whenever the deadlock is detected [5]. Moves that will mapped some of the logical registers on the other register subsets are then issued.

5.2.3 Register write specialization and register renaming

On conventional designs, register renaming and allocation of instructions to clusters are independent and can be performed either in parallel or sequentially. When register write specialization is used, the cluster that executes an instruction determines the register subset where the instruction result will be written.

In this paper we will assume that instructions are first allocated to clusters then renamed[6]. That is once the instruction has been allocated to a cluster, register renaming **has** to take this constraint in account. Note that since allocation of instructions to the clusters precedesthe final pipeline stage in the register renaming process, this is likely to generate an extra pipeline stage.

The register renaming process we present in this section is based on the use of free lists of registers. We make no hypothesis on the instruction allocation policy to clusters.

This process is illustrated in Figure 3 for a group of four instructions for a conventional monolithic register file superscalar architecture as well as for a superscalar processor for which write specialization is assumed with two register subsets, respectively the odd registers and the even registers.

For both designs the principles are the same. At a first step, dependencies are propagated within the group of N instructions that are renamed on a single cycle. Independently N new free registers are selected. At the second step, registers are renamed and a new map table is written.

For the monolithic register file, a single free list of registers is maintained, N new free registers are picked from this list in order to rename the results of a group of N instructions.

When register write specialization is used, the selection of free registers includes an extra step. A free list per register subset is maintained. N free registers are picked from each free list. Allocation of instructions to clusters has been done before register renaming, and this information is used through the form of a subset target vector.

Recycling registers This renaming mechanism consumes a lot of free registers. This waste of free registers can be limited if static (or balanced within the group of instructions renamed on a single cycle) cluster allocation policy is used. For example, round robin allocation of instructions to clusters was shown to be a quite acceptable policy for a 4-cluster architecture [1].

However, if no assumption is made on the policy for allocating instructions to clusters then on every cycle, N registers must be picked on each free list for renaming any group of N instructions.

The recycling of free registers for both designs can be handled in pipelined mode as follows for each of the free lists:

[5] This can be implemented through maintaining a count of logical registers mapped onto the register subfile

[6] The alternative solution (register renaming first, instruction allocation to clusters at second) may lead to very unbalanced workloads on clusters.

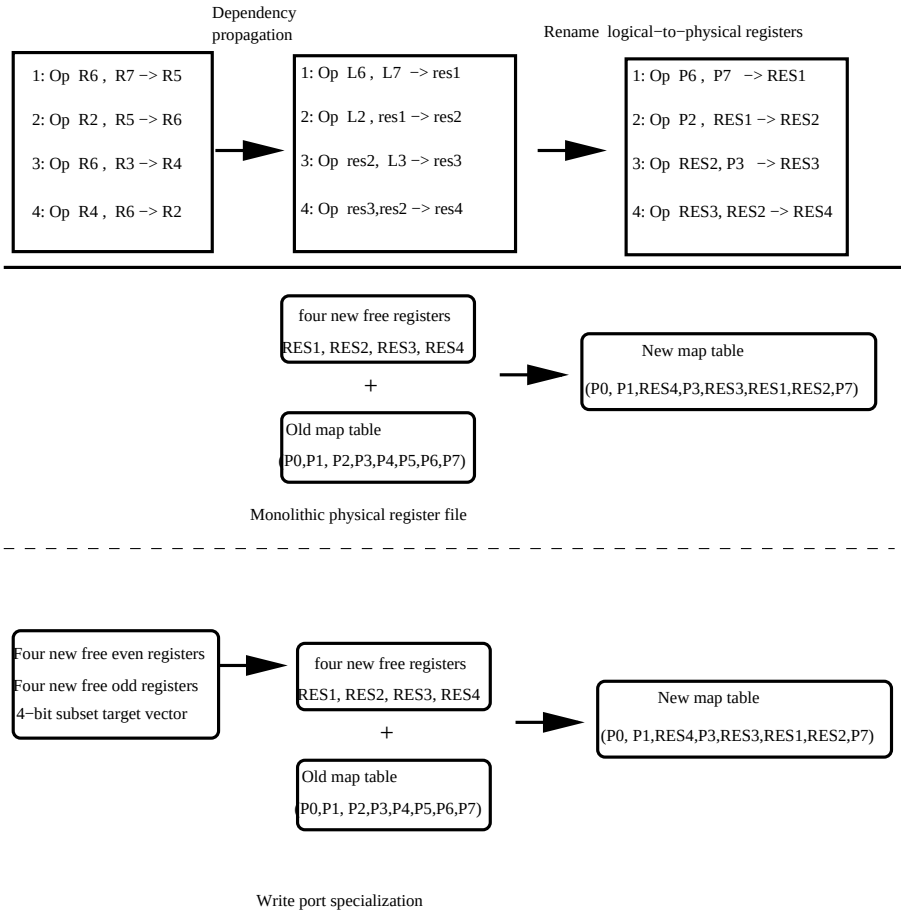

Figure 3: Register renaming

1. build the two lists of registers to be recycled (list of registers freed by committed instructions and list of registers that were not attributed to the group of instructions renamed on the previous cycle).
2. independently pack both lists
3. make a single list
4. append this list to the free list.

Note that a large number of free registers are not accessible since they are flowing through the recycling pipeline. On a 8-way issue 4-cluster superscalar processor, assuming 4 register subsets and that in average 6 instructions per 8-instruction block produce a register result, an average of 26 instructions has be recycled on each cycle. A four-cycle register recycling pipeline will lead to an average of 104 free registers in progress in the recycling pipeline (and therefore unaccessible till they flow out from this pipeline).

An alternative register renaming design An alternative design eliminates the difficulty with unaccessible free registers, but at the cost of lengthening the register renaming pipeline.

For a group of N instructions to be renamed on a single cycle, the exact numbers N_i of registers required from each register subset S_i is first computed from the subset target vector. Then the exact numbers of required free registers are picked from each free list.

These groups of registers are then expanded and merged using the subset target vector. Careful design should limit the extra pipeline length incurred by such a design to 2 or 3 cycles.

5.2.4 Pools of functional units and register write specialization

This paper essentially focuses on clustered architectures assuming identical clusters of functional units. However register write specialization can also be applied in the context of pools of identical functional units.

For instance, if the processor features a pool of load/store units, a pool of simple integer units and a pool of long latency units, then the three pools can write onto three different subsets of the register files. Note that these register subsets may have different sizes.

5.3 Register read specialization

On conventional architecture, each individual register in the physical register file can be read as an operand by any functional unit. Register read specialization consists in limiting the number of possible readers to each individual register. On a clustered architecture, this can be implemented as follows: for each of the clusters, every first (resp. second) operand of an instruction will come only from a subset of the registers.

For instance, on a 2-cluster architecture, one can force all instructions executed by cluster C_0 (resp. C_1) to read their second operand (if any) on the even (resp. the odd) registers. This would reduce the number of read ports on each individual register.

Register read specialization puts a strong constraint on the allocation of instructions to clusters: the instruction must be allocated to a cluster that is able to read its operands. Note also that, for any possible location of the operands of an instruction the register file, there **must** exist at least one cluster able to execute it.

6 4-cluster WSRS architecture

We define a WSRS (for register Write Specialization, register Read Specialization) architecture as an architecture then that implements both register write specialization and register read specialization. In this section, we present a 4-cluster WSRS architecture.

Through this example, we show that combining register write specialization with register read specialization puts some constraints on the allocation of instructions to clusters, but allows significant savings on the complexity of the register file as well as on the complexity of the bypass pathes and the wake-up logic.

6.1 A 4-cluster WSRS Architecture

A 4-cluster WSRS architecture is illustrated in Figure 4:

- Functional units are grouped into four identical clusters, C_0, C_1, C_2 and C_3.
- The set of registers is split into four distinct register subsets S_0, S_1, S_2 and S_3.
- **Register write specialization**: The results produced on cluster Ci is written on the register subset Si.
- **Register read specialization**: for each instruction executed on a given cluster, the first (resp. the second) operand is read on a fixed pair of register subsets.

For a given (dyadic) instruction, its execution cluster as well as its register subset target are determined by the the register subsets where its operands are located. For instance, an instruction which first operand lies in register subset S2 and which second operand lies in register subset S1 will be executed by cluster C3, its results will be written on register

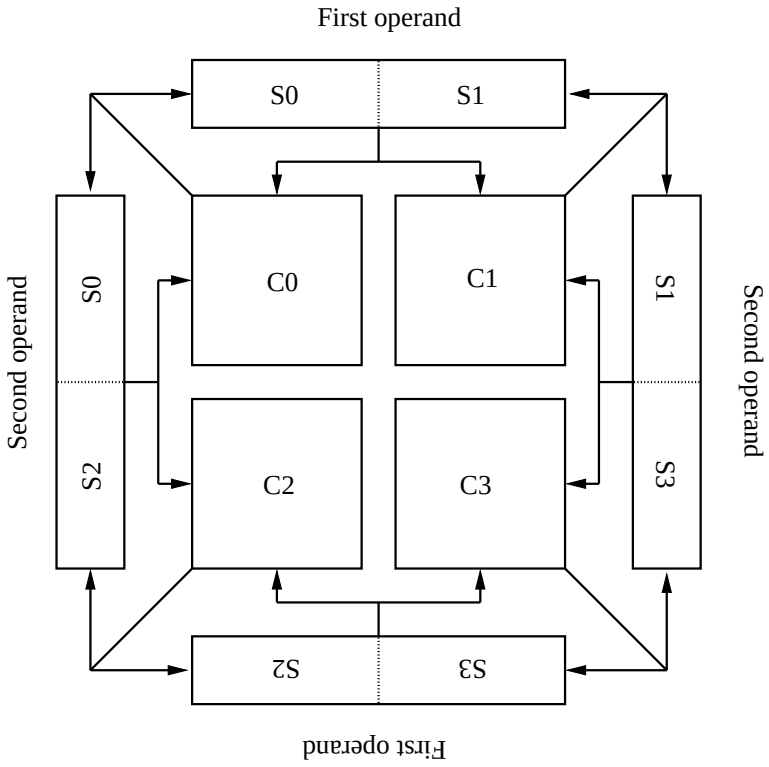

Figure 4: A 4-cluster WSRS architecture

subset S3. We will see that this allocation of instructions to clusters can be handled just before register renaming in the pipeline.

Compared with a conventional superscalar architecture, the 4-cluster WSRS architecture presents a major difference: any given physical register is connected with half of the functional unit entries and can be written by only one fourth of the functional units.

Figure 4 also illustrates a quite natural way to distribute the register file on the 4-cluster WSRS architecture. Each of the register subfiles only implements two of the four register subsets.

We will show that, on a 8-way 4-cluster WSRS architecture using a distributed register file, the silicon area devoted to the physical register file is five to ten times smaller than on a conventional 8-way superscalar processor. At the same time, the register file access time and its peak power consumption are also significantly reduced. Peak power consumption and silicon area can even be further halved provided that limited read port arbitration on register file read ports are implemented.

A second benefit of the 4-cluster WSRS architecture is to reduce the complexity of the bypass path in the processor. For a given functional unit entry in a given cluster, only half of the functional units could have produced the operand, then this entry has only to be connected to the output ports of these functional units.

A third benefit of the 4-cluster WSRS architecture is to reduce the complexity of wake-up logic, since the wake-up logic in a cluster has only to monitor half of the result buses in the processor.

6.2 Cluster Allocation and Register renaming

The allocation of instructions to clusters and register renaming are strongly linked in the 4-cluster WSRS architecture:

- The register subfile sources of the operands determine the cluster and target register subset for the instruction result.

Once the register subfile target has been determined, register renaming can be handled as described in 5.2.3.

We show here that the computation of the register target subfile can be handled before register renaming without creating any new critical electrical path.

Determining the register subfile target For the sake of simplicity, we assume here that the instructions are all dyadic.

A simple rule determines the cluster that will execute an instruction I:

> The first operand position determines on whether the instruction will be executed on the top or down 2-cluster, and the second operand determines whether the instruction will be executed on the left or right 2-cluster.

Remark that the computation of the two bits in the number of the target cluster for an instruction are independent.

6.3 Fast-forwarding

Forwarding an operation result for using it as an operand on the very next cycle is a very challenging task in a wide-issue superscalar processor, since the distance between the functional units might be long. A systematic one (or more) cycle delay for forwarding results might seriously impair performance.

For a two-cluster solution, Palacharla et al. [9] proposed to enable this fast-forwarding only inside a single cluster. The result from cluster C0 is available as an operand for cluster C1 with a one cycle delay. One can then try to optimize the distribution of instructions among the two clusters to take part of this restricted fast forwarding.

Figure 4 illustrates a possible layout of the 4-cluster WSRS architecture where the consumer cluster is always close to (i.e. touches) the producer cluster. Such a layout may favor a simple implementation of fast-forwarding capabality.

The choice of implementing complete or partial fast-forwarding capability is out of the scope of this paper. However, three possibilities of increasing complexity are natural:

- Fast-forwarding inside a single cluster.
- Fast-forwarding inside pairs of adjacent clusters ((C_0, C_1) and (C_2,C_3) for instance).
- Complete fast-forwarding.

6.4 Policies for allocating instructions to clusters

We list here the degree of freedoms that can be exploited for the allocating the instructions to the clusters on the WSRS architecture.

6.4.1 Monadic instructions

Up to now, we have focussed our attention on dyadic instructions. However, a large fraction of the instructions are either monadic either noadic[7].

Monadic instructions offer a degree of freedom for the distribution of instructions among clusters since they can be executed by two clusters. However this may lead to a slight unbalancing in the workload: chains of dependent monadic instructions will be executed on a single cluster pair (either (C_0, C_1) or (C_2, C_3)).

[7]Simple preliminary tests showed that 70-95% on the SPEC2000!

6.4.2 Dyadic instructions

On highly optimized codes, the compiler tends to maintain invariant operands in the registers in order to avoid repetitive loads of the same data.

On the WSRS architecture, this may lead to unbalance the workload among the clusters. Let us consider a loop body example illustrated in Figure 5, we suppose that R1 is an invariant register initially stored in S_0 and that R4 is only modified by instruction (1). All iterations of instruction (1), except the first, will be executed by the same cluster.

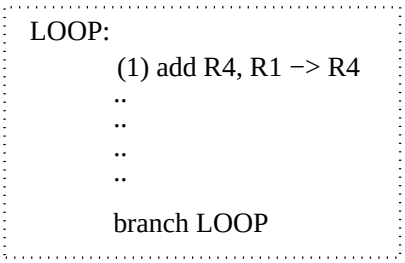

Figure 5: A loop example

Exploiting commutative dyadic instructions This phenomenom can be limited through exploiting the commutativity of many dyadic instructions (add, or, exclusive-or, ..).

Commutative dyadic operations can be executed on two clusters provided that the two operands do not lie in the same register subset.

This second degree of freedom can be exploited through inverting the two operands *before cluster allocation* in the pipeline.

"Commutative" clusters In order to further improve the degree of freedom provided by the commutative dyadic instructions, functional units can be implemented in order to be able to execute instructions in two forms inverting the operands order (i.e for instance computing A-B and -A+B).

This can be implemented through either the use of a switch on the entries of the functional units or through implementing both functions in the execution units. Any dyadic instruction with register operands in different register subsets can be executed on two clusters.

Alternative cluster-register interconnection The interconnection of registers with registers illustrated in Figure 4 favors locality for fast-forwarding instruction results. On the other hand, it may lead to some unbalancing in the workload among the clusters (for chains of monadic instructions for instance).

Alternative cluster-(register subsets) interconnection can be considered. For instance, cluster C_i writing register subset $S_{(i+2)\ mod 4}$ is a possibility. This topology would allow to distribute chains of dependent monadic instructions on the four clusters. On the other hand, implementation of a complete fast-forwarding bypath would be much more difficult using this topology rather than the original one in Figure 4.

6.5 Complexity of the 4-cluster WSRS architecture

In this section, we compare the 4-cluster WSRS architecture with a conventional 8-way superscalar processor in terms of complexity of implementation. By complexity, we mean many parameters including cycle time, power consumption, silicon area, addition of extra components and logic, ..

We first point out the restrictions, the extra pipeline step and extra hardware logic associated with the 4-cluster WSRS architecture. Then we analyze the complexity advantages of 4-cluster WSRS architectures on the register file, on the wake-up logic and on the bypass matrix.

6.5.1 Constraints and extra hardware

Four identical clusters The 4-cluster WSRS architecture involves four identical functional units clusters. Each cluster must be able to execute every kind of instruction. This is not an issue for standard ALU operations, but might be considered as an issue for less frequent instructions such as integer division or multiplication. Replicating dividers and multipliers on every cluster might be quite expensive. As an alternative to complete replication, a divider (resp. multiplier) can be shared among two adjacent clusters. Static arbitration among the two clusters will allow a quite smooth sharing of the clusters.

Distribution of instructions among clusters The distribution of instructions among clusters is determined by the position of their operands in the register subfiles. We have pointed out degrees of freedom associated with monadic instructions (and to some limits with dyadic instructions). However this constrained distribution may slightly impair performance compared with other clustered architectures where the distribution can be optimized dynamically at run-time [9, 1].

Need for more physical registers On the 4-cluster WSRS architecture, a lot of registers are unaccessible during the register recycling pipeline (see 5.2.3).

Longer renaming pipeline and extra free lists The renaming pipeline is longer on 4-cluster WSRS architecture since it involves an extra step: the computation of the register subset target vector. It also uses three extra free lists compared with conventional renaming (see 5.2.3).

6.5.2 Complexity of the register file

CACTI2.0 simulations and Formula 1 are used to evaluate the complexity of the register file[8].

Table 2 reports data on the silicon area, access time and peak power consumption of the register file for 8-way issue superscalar architecture for four architectures, a conventional monolithic register file architecture(conv-C), a 4-cluster 8-way architecture using a distributed register file (conv-D), a 4-cluster WSRS architecture without assuming limited read arbitration (4-WSRS) and a 4-cluster WSRS architecture assuming limited read arbitration (4-WSRS-LRA), and a conventional 4-way issue superscalar architecture as a reference (4-way).

For 8-way conventional architecture, a total of 256 registers are assumed and 128 registers for the 4-way issue architecture. In order to to compensate the large number of free registers that are unaccessible because of the register recycling pipeline on the 4-cluster WSRS architectures, 384 registers are assumed for these architectures.

As in Section 4.2.3, 0.10 μm CMOS technology and a 5 Ghz clock are assumed. We assume also a single extra pipeline cycle for driving the data from the exit of the register file to the entry of the functional units when the register file is distributed and two extra pipeline cycles when the register file is centralized.

For 4-WSRS, CACTI simulations indicate that using two (2-read, 4-write) copies of each of the 4 register subset pairs is a (slightly) better solution than using a (4-read, 4-write) copy

[8]CACTI2.0 software was modified to model also register files for which register write specialization is implemented

in terms of access time as well as peak power consumption[9]. Therefore, we report CACTI results for this configuration. Using limited read arbitration allows to use a single (2-read, 4-write) copy for each register subset pair.

Finally, for each of the architectures, we report the relative silicon area of the register file compared with 4-WSRS-LRA.

Analysis By reducing the number of ports on each individual register, the 4-cluster WSRS architecture enables a dramatic complexity reduction. The silicon area devoted to a single register bit file compared with conventional designs: compared with a distributed register file for a conventional architecture, the total silicon area is reduced by a factor $\frac{20}{3}$.

Peak power consumption is also reduced by more than one third for a distributed register file. A shorter register pipeline access is also enabled: the extra pipeline stage(s) lost on register renaming is likely to be compensated by this benefit.

When applied, limited read arbitration further halves the silicon area as well as the peak power consumption since a single copy register subset pair is used.

Finally note that, when limited read arbitration is used on a 8-way 4-cluster WSRS architecture, a 384-entry register file features smaller silicon footprint, lower power consumption and a shorter read access time than a 128-entry register file om a conventional 4-way superscalar processor.

	conv-C	conv-D	4-WSRS	4-WSRS-LRA	4-way
F*(S,R,W)	(256, 16, 8)	4 *(256,4,8)	8*(192,2,4)	4*(192,2,4)	(128,8,4)
nJ/cycle	8,01	6.37	3.82	1.91	2.19
Peak power (W)	40.05	31.85	19.10	9.55	10.95
Access time (ns)	0.95	0.65	0.42	0.42	0.53
Pipeline cycles	7	5	4	4	4
register bit area (x w^2)	768	960	96	48	192
relative area	$\frac{32}{3}$	$\frac{40}{3}$	2	1	$\frac{4}{3}$

Table 2: Register file complexity

6.5.3 Bypass path complexity

As the access to the register file will be pipelined in future generations of processors, two distinct issues must be distinguished: first the complexity of the bypass network needed to forward the data to the functional units without waiting for the cycle when the data can be directly read from the register file, second the fast-forwarding capability (i.e. the ability to use an instruction result as an operand for a dependent instruction on the very next cycle).

Bypass point complexity

At execution, operands of an instruction are selected among all the possible sources: the register file and results that have flown from their execution units but are still not available from the register file.

On wide-issue superscalar processors, the cost of a complete bypass network is huge: if the read-write pipeline on the register file is X-cycle long and if the register can be produced by N possible units then, for each functional unit input, up to $X*N$ results are still unaccessible from the register file. That is bypass logic must choose among $X*N+1$ possible sources for each operand.

Table 3 illustrates the numbers of possible sources for an instruction operand assuming the read pipeline lengths in Table 2 for a 8-way issue superscalar processor. The 4-cluster

[9]Each individual register features only 2 write ports.

WSRS architecture benefits from two factors: only 4 possible sources for the operands and a shorter register read pipeline.

Architecture	conv-C	conv-D	4-WSRS
sources	57	41	17

Table 3: Number of possible source origins for an instruction operand

The effective design of a bypass point is out of the scope of this paper. However, even with "only" 17 possible operand sources, this will be a challenging issue.

Fast forwarding

As already pointed out in Section 6.3, implementing complete fast-forwarding between all functional units in the processor is likely to impair processor cycle.

In a first approximation, the fast-forwarding delay increases with the distance between the functional unit producing of a register and the functional unit consuming it. A second order factor is the number of entries that have to be fed within the the next cycle. Therefore, critical pathes for fast-forwarding on a 4-cluster WSRS architecture are shorter than on a conventional superscalar processor, for complete fast-forwarding as well as for restricted fast-forwarding (fast-forwarding inside a pair of clusters).

Note that in this latter case, in the absence of any optimization in the allocation of instructions to clusters, three out of four of the consumer instructions will be correctly located to capture the data on the bypass network on the production cycle for the 4-WSRS architecture, but against only one out of two for a conventional 2-cluster (or 4-cluster) architecture.

6.5.4 Wake-up logic complexity

In an out-of-order execution processor, an instruction can not be issued until its operands have become valid. The wake-up logic is responsible for monitoring the source dependencies for instructions in the issue window. On each cycle, the wake-up logic entry associated to an instruction must monitor every possible source for any of its operands and checked it against its register operand numbers.

For a given instruction executed on a given cluster on a 4-cluster WSRS architecture, a given operand can only be produced by two of the four clusters in the processor. Therefore the complexity of an entry in the wake-up logic in a 8-way 4-cluster WSRS architecture is the same as the complexity of an entry in the wake-up logic in a 4-way issue conventional processor in terms of silicon area, power consumption and access time.

On the other hand, on a 4-cluster WSRS architecture, each of the clusters must be able to accept all the instructions renamed on a single cycle. This leads to a very wide write port on the wake-up logic.

7 Towards very wide issue superscalar processors

4-cluster WSRS architectures represent a very attractive solution to implement 8-way superscalar processors using 2-way clusters. However, SMT architectures [12] make a clear case for designing even wider issue processors if one is able to reach high clock frequency and to master power consumption and silicon area. 12-issue (resp. 16-issue) 4-cluster WSRS architectures involving 3-issue (resp. 4-issue) clusters represent possible design points for such architectures.

In this section, we present other possible (relatively) low complexity design points for WSRS architectures First, we present a solution featuring 7 2-issue clusters where only four

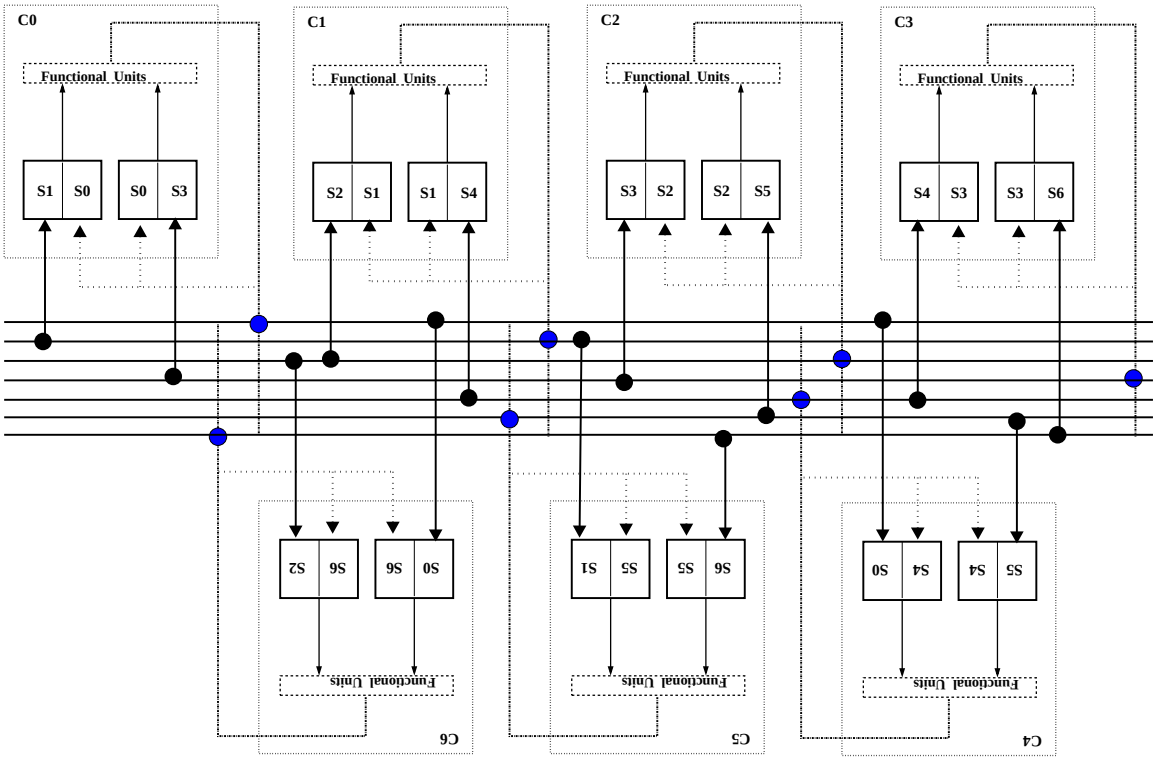

Figure 6: A 7-cluster WSRS architecture using distributed register file

(2-read, 2-write) copies of each individual register are used, but for which instruction allocation to clusters is strongly constrained. Second, we present a 9-cluster WSRS architecture, which naturally scales from the 4-cluster WSRS architecture presented in the previous section. Compared with the 4-cluster WSRS architecture, it suffers an overcost of 2 extra (2-read,2-write) copies for implementing each physical register, but benefits from the same degrees of freedom for instruction allocation to clusters.

These two design points illustrate the spectrum of trade-offs that combining register write specialization and register read specialization offers for designing wide-issue superscalar processors.

7.1 A strongly constrained 7-cluster WSRS architecture

If as in 6.4.2, the functional units of the clusters are assumed to be able to execute all dyadic instructions in two forms through inverting the operand order then it is possible to build a K-cluster 2K-way WSRS architecture using registers that are read only from 8 different functional unit entries for any number K smaller or equal to 7.

The register file is divided in K distinct subsets S_i. Each cluster writes its results on a distinct register subset (register write specialization). The first operand (resp. second operand) of an instruction executed in a cluster C_j belongs to a pair of register subsets, i.e it can only have been produced by two clusters (register read specialization).

The 7-cluster WSRS architecture is perfectly symmetric. One can either implement it with a centralized register file or with a local register files associated with each cluster. Note that in this latter case, the different local register subfiles are all different. This distributed register file architecture is represented in Figure 6.

Three possibilities exist for the choice of distributing the register subset targets for clusters[10]:

1. Cluster C_i writes its result in register subset S_i. As for dyadic instructions, both operands can be read from S_i, this solution favors fast-forwarding inside a cluster.
2. Cluster C_i writes its result in register subset S_j, $j \neq i$, S_j being a possible source for one operand for cluster C_i. This solution allows some fast-forwarding inside a cluster.
3. Cluster C_i writes its result in a register subset S_j, that is not read connected with it. This solution guarantees that chains of dependent instructions will exercise different clusters.

Trade-off between complexity and instruction allocation policy

Assuming 2-way clusters, the silicon area devoted to each register, the complexity of each individual bypass point, and the complexity of an entry in the wake-up logic in each cluster are the same for the illustrated 7-cluster WSRS architecture as for the 4-cluster WSRS architecture presented in the previous section:

- Each of the two register subfiles in a 2-way cluster features only 2 read ports and 4 write ports, each write port being connected with only half of the registers in the subfile.
- Each of the bypass points has to capture results coming from only two cluster sources.
- Each entry in the wake-up logic has also to monitor the results coming from only two clusters.

But on the other hand, the policy for allocating instructions on clusters is more constrained on a 7-cluster WSRS architecture than on a 4-cluster WSRS architecture: clusters have to be able to execute non-commutative instructions in two forms and even in this condition and the allocation of monadic instruction to clusters is the only degree of freedom for the policy for allocating instructions to clusters.

7.2 A 9-cluster WSRS architecture

A 9-cluster WSRS architecture is illustrated on Figure 7. As in Figure 4, cluster C_i writes registe subset S_i.

Compared with the previous 4-cluster and 7-cluster WSRS architecture, the complexities of the register file, of the wake-up logic and of the bypass network are higher, but they remain in a reasonable range (when compared with conventional superscalar architectures). For sake of simplicity, we assume here 2-way clusters:

- The register file can be built using only (2-read, 2-write) registers, each physical register being replicated 6 times, i.e. 2 extra copies when compared with the 4-cluster WSRS architecture.
- An operand can only be produced by 3 clusters, i.e. there are only 6 possible sources (plus the register file) for the first operand. That is the complexity of the bypass point (for the first operand) and the complexity of a wake-up logic entry are in the same range as the ones for a conventional 6-way superscalar processor.

On the other hand, policies for allocating instructions on clusters may leverage the same degrees of freedom as for the 4-cluster WSRS architecture (monadic instructions, commutative dyadic instructions, ..).

[10]The first one is illustrated on Figure 6

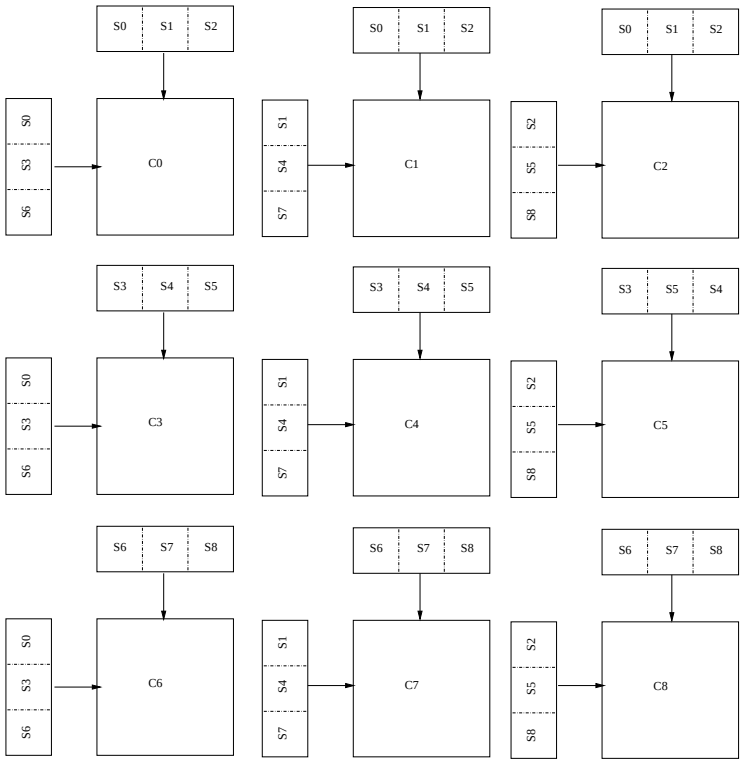

Figure 7: A 9-cluster WSRS architecture (*Write buses are omitted*)

Remark Assuming that each cluster is able to execute operations in the two forms inverting the operands, for any K smaller than 17, it is possible to design a K-cluster 2K-way WSRS architecture using only 6-copy (2-read, 2-write) registers and for which any operand has only three possible producers.

8 Conclusion

The designers of wide issue out-of-order execution superscalar processors are facing major obstacles. When the issue width doubles, the number of physical registers required for performance also doubles [6] as well as number of ports on the register file. As the silicon area of a multiported memory increases quadratically with the number of access ports, this leads to an eight-fold increase of the silicon area occupied by the register file, while the silicon area occupied by the functional units only doubles. At the same time, power consumption and acces times also raises. The complexity of the wake-up logic and the bypass network also increases with the number of possible sources for the operands. As a consequence, the relative part of the silicon area and power consumption spent in functional units decreases dramatically when the issue width increases while register access, but also wake-up and selection logic [10] requires deeper and deeper pipeline.

The mechanisms proposed in this paper for clustered architectures may allow to break this infernal trade by reducing the number of functional units entries and exits connected with each individual register.

Register write specialization allows to reduce the number of write paths to each individual register. It makes the use of (multiple copies) distributed register files a complexity-effective solution for silicon area, peak power consumption and access time. Register write specialization imposes a link associating register renaming with allocation of instructions to clusters.

Register read specialization allows to reduce the number of read paths to each individual register.

Combining register write specialization and register read specialization in WSRS architectures enables complexity effective register files for wide issue clustered superscalar processors. It also dramatically reduces the complexity of the bypass network and of the wake-up logic. For up to 7 2-way issue clusters, we have shown that one can design WSRS architectures with the following properties:

- The silicon area devoted to each individual register is smaller than in current generation 4-way superscalar processors, since the number of write ports on each individual register is decreased to two while the number of read ports remains eight.
- The peak power consumption of the register file is mastered and increases linearly with the number of clusters.
- The access time on the register file is determined by the size of the register subfiles and does not depend on the number of clusters.
- The complexity of each individual bypass point does not depend on the number of clusters and is equivallent to the complexity of the bypass point on a conventional 4-way superscalar processor.
- The complexity of a wake-up logic entry in each individual cluster is equivallent to the complexity of a wake-up logic entry on a conventional 4-way superscalar processor.

Limited read port arbitration can also be considered to further reduce the silicon area and the peak power consumption of the register file.

The 4-cluster WSRS architecture can be implemented as cluster-centric (as shown in Figure 4). This naturally simplifies the implementation of fast-forwarding.

We have further shown that, it is possible to design even wider issue WSRS architectures while still mastering the complexity of the register file.

For clustered superscalar architectures, our propositions trade the complexity of the register file, of the bypass network and of the wake-up logic against degrees of freedom for allocating of instructions to clusters and a more complex register renaming. The location of the physical register operands restricts the set of clusters that can execute an instruction. However monadic instructions can be executed on several clusters. One can also execute commutative dyadic operations on several clusters, .. Policies for instruction allocation to clusters can exploit these degrees of freedom to balance the workload among the clusters. Further studies are needed to explore these policies, to quantify the performance of WSRS architectures and to look for the best trade-offs for cluster-register interconnections.

Finally, while our study has mainly targetted dynamically scheduled architectures, we would like to point out that the structure we have proposed for the register file and the clusters could also be used to define scalable statically scheduled multi-clustered VLIW architectures (a la LX[4]).

References

[1] A. Baniasadi and A. Moshovos. Instruction distribution heuristics for quad-clustered, dynamically-scheduled, superscalar processors. In *Proceedings of the International Symposium on Microarchitecture, (Micro 28)*, dec 2000.

[2] Robert P. Colwell, Robert P. Nix, John J. O'Donnell, David B. Papworth, and Paul K. Rodman. A VLIW architecture for a trace scheduling compiler. In *Proceedings Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II) Computer Architecture News Operating Systems Review SIGPLAN Notices*, pages 180–192, Palo Alto, CA, October 1987. ACM.

[3] José-Lorenzo Cruz, Antonio Gonzalez, Mateo Valero, and Nigel Topham. Multiple-banked register file architectures. In *Proceedings of the 27th International Symposium on Computer Architecture*, june 2000.

[4] Paolo Faraboschi, Geoffrey Brown, Joseph A. Fisher, Giuseppe Desoli, and Fred (Mark Owen) Homewood. Lx: A technology platform for customizable VLIW embedded processing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 203–213, Vancouver, British Columbia, June 12–14, 2000. IEEE Computer Society and ACM SIGARCH.

[5] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. The multicluster architecture: Reducing cycle time through partitioning. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-97)*, pages 149–159, Los Alamitos, December 1–3 1997. IEEE Computer Society.

[6] Keith I. Farkas, Norman P. Jouppi, and Paul Chow. Register file design considerations in dynamically scheduled processors. In *Proceedings of the Second International Symposium on High Performance Computer Architecture*. IEEE, January 1996.

[7] Richard E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[8] Teresa Monreal, Antonio Gonzalez, Mateo Valero, José Gonzalez, and Víctor Vinals. Dynamic register renaming through virtual-physical registers. *Journal of Instruction-Level Parallelism*, May 2000.

[9] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In 24^{th} *Annual International Symposium on Computer Architecture*, pages 206–218, 1997.

[10] J. Stark, M. D. Brown, and Y. N. Patt. On pipelining dynamic instruction scheduling logic. In *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, December 2000.

[11] Marc Tremblay, Bill Joy, and Ken Shin. A three dimensional register file for superscalar processors. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, Jan 1995.

[12] Dean M. Tullsen, Susan Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22th Annual International Symposium on Computer Architecture*, June 1995.

[13] Steven J. E. Wilton and Norman P. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, May 1996.

[14] Victor Zyuban and Peter Kogge. The energy complexity of register files. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED-98)*, August 10–12 1998.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399