A Case for a Complexity-Effective, Width-partitioned Microarchitecture

Olivier Rochecouste, Gilles Pokam, André Seznec

To cite this version:

Olivier Rochecouste, Gilles Pokam, André Seznec. A Case for a Complexity-Effective, Width-partitioned Microarchitecture. [Research Report] PI 1742, 2005, pp.27. inria-00000211

HAL Id: inria-00000211

https://hal.inria.fr/inria-00000211

Submitted on 13 Sep 2005

PUBLICATION INTERNE
N^o 1742

A CASE FOR A COMPLEXITY-EFFECTIVE, WIDTH-PARTITIONED MICROARCHITECTURE

OLIVIER ROCHECOUSTE , GILLES POKAM , ANDRÉ SEZNEC

ISSN 1166-8687

A Case for a Complexity-Effective, Width-partitioned Microarchitecture

Olivier Rochecouste[*] , Gilles Pokam [**] , André Seznec [***]

Systèmes communicants
Projet CAPS

Publication interne n°1742 — Août 2005 — 27 pages

Abstract: Current superscalar processors feature 64-bit datapaths to execute the program instructions, regardless of their operands size. Our analysis indicates, however, that most executions comprise a large amount (40%) of *narrow-width operations*; i.e. instructions which exclusively process narrow-width operands and results. We further noticed that these operations are well distributed across a program run. In this paper, we exploit these properties to master the hardware complexity of superscalar processors. We propose a width-partitioned microarchitecture (WPM) to decouple the treatment of narrow-width operations from that of the other program instructions. We split a 4-way issue processor into two clusters: one executing 64-bit operations, load/store and complex operations and the other treating the 16-bit operations. We show that revealing the narrow-width operations to the hardware is sufficient to keep the workload balanced and the communications minimized between clusters. Using a WPM reduces the complexity of several critical processor components : register file and bypass network. A WPM also lowers the complexity of the interconnection fabric since the 16-bit cluster is only able to propagate narrow-width data. We examine simple configurations of WPM while discussing their tradeoffs. We evaluate a speculative heuristic to steer the narrow-width operations towards clusters. A detailed complexity analysis shows using a WPM model saves power and area with a minimal impact on performance.

Key-words: Hardware complexity, power consumption, superscalar processor, width-partitioned microarchitecture, register file, narrow-width operations, data-width predictor

(Résumé : tsvp)

[*] orocheco@irisa.fr
[**] gpokam@cs.ucsd.edu
[***] seznec@irisa.fr

Résumé : Les processeurs superscalaires actuels implémentent des chemins de données 64-bit pour exécuter les instructions d'un programme, indépendamment de la taille des opérandes. Notre analyse indique toutefois que la plupart des exécutions comportent une fraction considérable (40%) en *opérations tronquées* ; c-à-d. les instructions manipulant exclusivement des opérandes et des résultats de petite dimension. Nous avons aussi remarqué que les opérations tronquées sont bien distribuées au cours d'une exécution. Cette étude exploite ces propriétés pour maîtriser la complexité des processeurs superscalaires. Pour ce faire, nous proposons une microarchitecture clusterisée (WPM) pour découpler le traitement des opérations tronquées de celui des autres instructions du programme. Nous partitionnons ainsi le processeur entre deux clusters : un cluster 64-bit exécutant les opérations 64-bit, *load/store* et complexes, et un cluster 16-bit traitant les opérations 16-bit. Considérant les propriétés relatives aux opérations tronquées, nous montrons que révéler ces dernières au matériel est suffisant pour maintenir l'équilibrage des charges et minimiser les communications entre les clusters. Le modèle WPM réduit efficacement la complexité de plusieurs composants critiques du processeur : fichier de registres, réseau de *bypass*. Ce modèle réduit aussi la complexité du réseau d'interconnexion car le cluster 16-bit peut uniquement propager des données 16-bit. Nous examinons différentes configurations du WPM en discutant de leurs compromis. Nous évaluons une heuristique spéculative pour distribuer les instructions vers les clusters. Une analyse détaillée de la complexité indique que le modèle WPM réduit la consommation et la surface de silicium avec un impact minimal sur les performances.

Mots clés : Complexité matérielle, consommation électrique, processeur superscalaire, microarchitecture clusterisée, fichier de registres, opérations tronquées, prédicteur de largeur

1 Introduction

Increase in processor performance is strongly correlated to exploiting large ILP and sustaining fast clock rates. Although processor designers have been able to keep up with this performance growth during the past few decades, it now becomes increasingly challenging to do so without overcoming several major obstacles. Among these, the impact of wire delays [12] is expected to prevail as device features become smaller. Other major factors that preclude yielding higher clock frequencies include the physical register file, the bypass network, the wakeup and the selection logic [21]. In addition, as the complexity of these structures increases dramatically with larger issue widths, the impact on power consumption and area also appears to be a serious matter.

Researchers have proposed a large number of solutions to overcome the aforementioned issues. Some studies have considered partitioning the hardware resources into clusters of computational units to reduce the overall complexity [21, 9, 1]. In these studies, the partitioning is dictated by the need to break the complexity growth factor of the critical components by reducing their sizes. Hence, the resulting clusters have simpler structures, thereby enabling fast clock rates. However, a major bottleneck with this approach is the interconnect fabric used to communicate data between clusters. This interconnect fabric is relatively slow and dissipates a large amount of power [19]. It is therefore desirable to minimize the number of inter-cluster communications while also keeping the workload among clusters balanced.

Other studies have considered a careful design of the critical processor components to reduce this complexity. These studies are mainly directed by empirical analysis made on runtime data, such as the seminal observation made by Brooks et al. [4] that most applications only need part of the full datapath-width to execute. Several optimizations have been proposed which exploit this narrow-width operand property of programs to reduce power consumption [4, 24, 5, 14] or to improve performance [8, 17, 25, 18, 20]. While they do actually help reducing the complexity of certain critical processor components (e.g. the register file), quantifying their impact on the overall microarchitecture is more difficult. This is because many of these proposals feature complex implementations, sometimes requiring major changes to the hardware.

This paper proposes to make efficient use of the available silicon by exploring new possibilities of partitioning a microarchitecture based on narrow-width data. Central to our approach is the observation that the occurrence of *narrow-width operations*, i.e. instructions exclusively comprising narrow-width operands, and the other program instructions is relatively balanced and highly interleaved across a complete program run. We observed this program property on the MediaBench and SPEC2000 benchmarks. Because of the relative prevalence of these narrow-width operations in programs, about 40% of the instructions exhibit this property for the considered benchmarks, we suggest to use a width-partitioned microarchitecture (WPM) to master the hardware complexity of superscalar processors. In a WPM, we resort to partitioning to decouple the treatment of the narrow-width operations from that of the other program instructions. This provides the benefit of greatly simplifying the design of the critical processor components in each cluster (e.g. the register file) as no additional hardware is required for managing each type of instruction; yet, the interleaving of the two instruction types balances the workload among the clusters. We also show that WPM reduces the complexity of the interconnect fabric. In fact, since clusters with narrow-width datapath can only communicate narrow-width data, the datapath-width of the interconnect fabric is significantly reduced, yielding corresponding saving of the interconnect power and area. We present an efficient design of WPM, discussing various implementation choices including steering heuristics to distribute instructions among the clusters and a detailed analysis of the complexity factors affecting the performance, power and area. Our complexity analysis shows that using a WPM architecture instead of a classical 64-bit 2-cluster microarchitecture can indeed save power and silicon area with only a minimal impact on the overall performance.

The remainder of this paper is organized as follows. Section 2 elaborates on the motivations of this work, providing some intuitive observations about the rationale of our approach. WPMs are described in detailed in Section 3, while their complexity analysis is discussed in Section 4. The instructions steering mechanism is presented in Section 5.2. Results are presented in Section 6, while Section 7 discusses the related work. We conclude in Section 8.

2 Motivations

In recent works, several authors [4, 18, 24, 8] have pointed out the large availability of narrow-width data within compute-intensive integer and multimedia programs. To exploit this program property, various definitions of the operations executing with narrow-width operands have been assumed; depending on their application to the architecture. Brooks et al. [4] have qualified a *narrow-width operation* as an instance where both source operands can be represented with fewer than 16 bits, whereas Pokam et al. [24] considered the basic-block granularity to define narrow-width regions in a program. We formulate a different assumption that considers a narrow-width operation to be an operation where *no operand exceeds 16 bits*, including the destination operand.

Characterizing narrow-width operations We have quantified the number of occurrences of these narrow-width operations across the Mediabench and the SPEC2000 benchmarks. Our bitwidth analysis is exclusively devoted to operations processed through the integer functional unit, including the address calculation. Operations that execute with narrow-width operands in the two's complement form are also considered. Figure 1 reports the classification and the distribution of the integer operations using narrow-width operands. As a convention, we note N for a narrow-width operand and F for a full-width operand. We use a 3-letter notation for categorizing an operation: the two leading letters represent the width type of the source operands and the last letter is the width type of the result. For instance, NFN stands for an operation which processes a narrow-width an a full-width source operands and produces a narrow-width result. For the monadic operations, we consider that all of the source operands feature the same width.

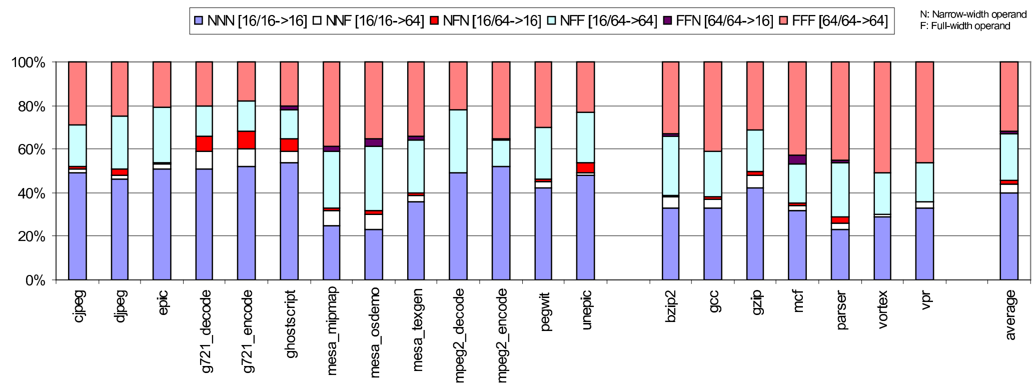

Figure 1: Classification and distribution of integer operations using narrow-width data.

We observe from Figure 1 that a significant part of the integer execution is devoted to the narrow-width operations (NNN), about 40%. These results corroborate the prior observations made by Brooks regarding the prevalence of narrow-width data for the integer operations. This suggests a scheme that decouples the processing of narrow-width operations onto dedicated narrow operators. This would reduce significantly the complexity of certain processor components lying on the critical path. In this study, we advocate the use of decoupling the processing of narrow-width operations onto dedicated narrow-width clusters, where one or more clusters feature a narrow datapath-width. We refer such a partitioned model as a *width-partitioned microarchitecture* (WPM). As for a conventional partitioned architecture, a WPM calls for a proper steering mechanism to distribute the narrow-width operations. It is crucial for both performance and power that the steering heuristics balance the workload among clusters while minimizing inter-cluster communications.

Inter-cluster communications Figure 1 also provides an estimate of the average number of communications that take place within a WPM. An inter-cluster communication in WPM can be triggered if an operation consumes a narrow-width value produced in a remote cluster (e.g. **NNF, NFN, NFF**) or if it produces a narrow-width value that must be propagated to a remote cluster (e.g. **NFN, FFN**). As shown in Figure 1, this concerns roughly 20% of the integer operations. For a WPM, this might translate into the worst-case scenario where one operation out of five triggers an inter-cluster communication. However, this is a maximal bound since this is strongly correlated with the narrow-width operations distribution and the data dependency among operations, i.e. not all the narrow-width operations have a data dependency with the other larger width program instructions. Our result section indeed shows that the number of inter-cluster communications is far below this bound.

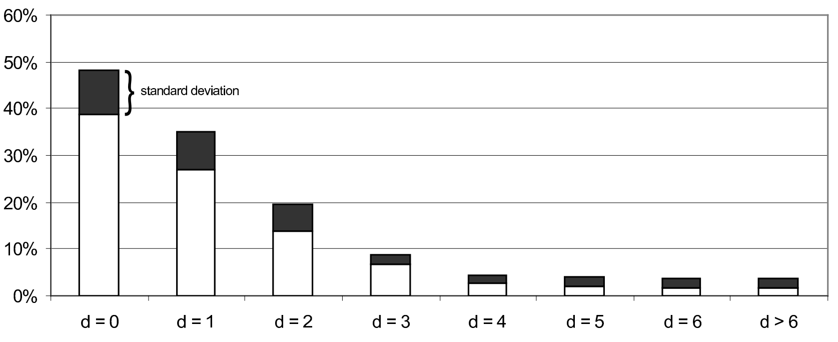

Figure 2: Distance between narrow-width operations and the other program instructions at runtime.

Workload balance Another relevant task for the instructions steering mechanism is to guarantee a good workload balance among clusters. We have approximated the workload balance that a WPM might be subject to as follows. For each operation, we have collected the distance separating a narrow-width operation from the next operation that executes with larger data. Figure 2 displays the mean of the most frequent distances observed over all benchmark applications at runtime. The standard deviation across applications is also reported and reveals the strong correlation of narrow-width distribution between applications. Another phenomenon illustrated in Figure 2 is the dominance of short distances at runtime. This may be due to the fact that we also included address calculations which frequently solicit the full datapath-width. This might therefore mean that occurrences of narrow-width operations are highly interleaved with the other operations in program execution. From a WPM viewpoint, this means that a simple steering heuristic may be able to achieve a balanced workload.

3 Width-partitioned Microarchitecture

Most integer and multimedia applications exhibit a large fraction of narrow-width operations that are also well distributed across the execution. To take advantage of this program property, we examine a novel partitioned architecture that can efficiently operate on narrow-width operations as well as on the other program instructions, with reduced complexity. We refer to this novel partitioned organization as *width-partitioned microarchitecture* (WPM). This section describes the implementation of such a 4-way WPM design. One can easily consider scaling up this design to larger issue-width. To do so will require some modifications to the inter-cluster communication model. This is however beyond the scope of this paper.

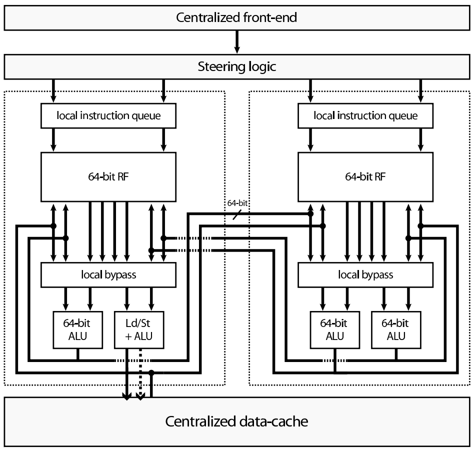

Figure 3: Baseline organization

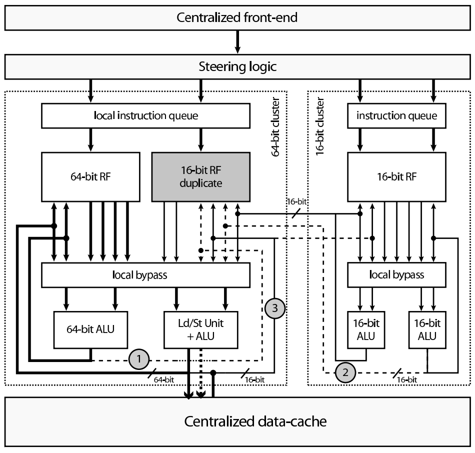

Figure 4: WPM organization

3.1 Baseline model

Our baseline model is derived from the Alpha 21264 [13]. It is a 64-bit, out-of-order, dual-cluster machine. We assume that the floating-point operations are processed in a dedicated cluster not described in this paper. Figure 3 shows the block diagram of this baseline organization. As depicted in the figure, the processor front-end (fetch, decode and rename) and the data cache are shared by all clusters. Similarly to the Alpha 21264 [13], we assume that the issue queues are decoupled from the reorder buffer and partitioned among clusters. The other components comprise the functional units and the register file which is duplicated onto each cluster. Both clusters are capable of issuing up to two instructions per cycle. Every 64-bit ALU can treat complex instructions such as multiplication or shift operations. We assume that the scheduling of memory operations is restricted to a single cluster. In addition, we consider that the load/store unit is capable of processing integer and logic operations. Since we examine a dual-cluster implementation, a fully-connected topology is advocated to circumvent potential resources contention and maximize performance. For supporting this topology, each register file (RF) copy must feature a number of write ports equal to the total number of ALUs, i.e. 4-write ports per cluster.

Once fetched and decoded, instructions are proceeded by the renaming stage. At this step, the steering logic is responsible for dispatching the instructions to the proper cluster. We rely upon an instruction steering heuristic similar to [7] which steers instructions to the cluster that produces most of its operands if this cluster comprises the proper functional unit. An instruction can only access its source operands from the local RF. We assume that inter-cluster communications are implicitly done by propagating every results to the local and the remote RF. For the producing cluster, data are bypassed in the same cycle to allow

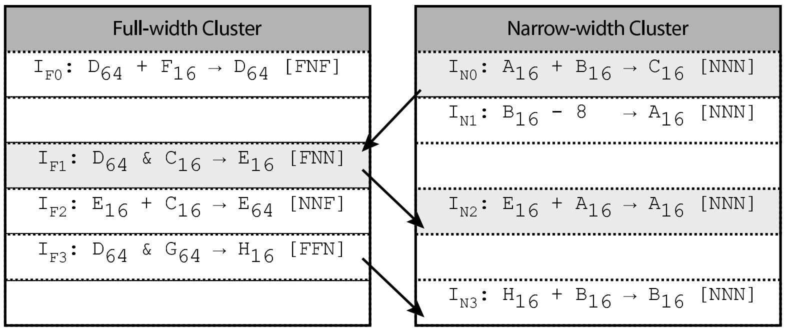
Full-width Cluster
I_{F0}: D_{64} + F_{16} → D_{64} [FNF]
I_{F1}: D_{64} & C_{16} → E_{16} [FNN]
I_{F2}: E_{16} + C_{16} → E_{64} [NNF]
I_{F3}: D_{64} & G_{64} → H_{16} [FFN]
Narrow-width Cluster
I_{N0}: A_{16} + B_{16} → C_{16} [NNN]
I_{N1}: B_{16} - 8 → A_{16} [NNN]
I_{N2}: E_{16} + A_{16} → A_{16} [NNN]
I_{N3}: H_{16} + B_{16} → B_{16} [NNN]

Figure 5: Inter-cluster communication scenario. The suffix indicates the width of the operand.

back-to-back executions, whereas broadcasting data to the other cluster takes additional cycles.

3.2 WPM design

The *basic WPM* design considered in this study splits the integer core into two distinct clusters: (1) a main *full-width cluster* featuring a 64-bit datapath and (2) a *narrow-width cluster* featuring a 16-bit datapath. As in the baseline model, each cluster is composed of a set of functional units (FUs) and a local RF. The narrow-width cluster features two 16-bit ALUs and a local 16-bit RF (called narrow-width RF). As shown in Section 4, this organization dramatically reduces the overall processor complexity as there is no need for additional hardware to keep track of the different datapath-width execution modes. The narrow-width RF has four read ports and two write ports to provide support for the execution of two operations per cycle. The full-width cluster, on the other hand, comprises a 64-bit ALU and one load/store unit capable of executing simple arithmetic and logic operations. A 64-bit local RF (called full-width RF) is provided with four read ports and two write ports to support the execution of two 64-bit operations per cycle. Restricting the load/store unit to the full-width cluster is coherent with our approach since address calculations generally operate on the full datapath-width. Figure 4 illustrates this basic WPM organization. Similar to the baseline model, we do not consider partitioning the processor front-end and the data cache. We do however need to address with care the communication between the narrow-width cluster and the full-width cluster.

3.2.1 Inter-cluster communications

The need to communicate data between the narrow-width cluster and the full-width cluster is dictated by the propagation of data dependency among the narrow-width operations and the other program instructions. Consider for instance the execution scenario depicted in Figure 5. Operation I_{N0} on the narrow-width cluster produces a value that is later needed by operation I_{F1} executing on the full-width cluster. The result of this operation is then consumed on the narrow-width cluster by operation I_{N2}. These edges actually label the data dependencies between the operations. The number of such edge is the cut-size between the set of narrow-width operations and the other program instructions and is actually a maximum bound on the total amount of inter-cluster communications, e.g. three communications in the given example.

A first naive implementation is to make each FU on each cluster be write-connected to the RF of the remote cluster. This will add four additional write ports on each RF: two for the 64-bit ALU and the load/store unit, and two others for the two 16-bit ALUs. Obviously, this is detrimental for the performance and the power consumption considering the fact that potentially 20% of the operations in the full-width cluster will be contributing to the inter-cluster communications (see Figure 1). The 16-bit duplicate RF shown in Figure 4 has been specifically thought to break down this complexity. This RF provides a copy of the narrow-width RF and is kept synchronized with it by the functional units in both clusters.

Narrow-width cluster implementation details Regarding the narrow-width cluster, two write ports are provided by the 16-bit duplicate RF to allow the 16-bit ALUs to keep each writeback register up to date with their copy in the remote cluster. There are two reasons to maintain the ALUs in the narrow-width cluster fully-connected with the remote RF copy. First, as shown in Figure 1, the availability of narrow-width operations in programs is large enough to justify the need of more communication bandwidth between the narrow-width cluster and the full-width cluster. Second, Figure 1 evidences the fact that among the operations that may potentially involve a remote communication with the narrow-width cluster, the **NFF** operations are by far the largest. A **NFF** operation may consume its value from the 16-bit duplicate RF, meaning that the copy must have been kept up to date by the narrow-width cluster.

Full-width cluster implementation details In the full-width cluster, two read ports and two write ports are provided by the 16-bit duplicate RF to allow the 64-bit ALU and the load/store unit to read and to write back their result. However, only one write port is actually connected to the remote RF copy. This latter is motivated by the observation that only a small fraction of the operations executing on the full-width cluster needs to be synchronized with their remote copy. In fact, these operations are restricted to the subset of instructions that produce a 16-bit result, e.g. I_{F3} and I_{F1} in Figure 5. As illustrated in Figure 1 with **FFN** and **NFN**, their representativeness in programs is negligible; there is therefore no need to provide the full write bandwidth to keep both copies synchronized. Moreover, our analysis showed that among those instructions that may involve a remote communication with the narrow-width cluster, a large percentage of these are actually narrow-width loads. This explains the additional port on the narrow-width RF which is write-connected with the load/store unit in the full-width cluster. The broadcast to the remote RF copy is done each time the load/store unit writes to the 16-bit duplicate RF.

Since only the load/store unit maintains both RF copies synchronized with each other, it is possible that a value being written back by the 64-bit ALU in the 16-bit duplicate RF is not available in the narrow-width RF when a dependent narrow-width operation is ready to issue. In that case, we assume the hardware automatically inserts a copy instruction to forward that value to the RF copy [22]. Note however that this case is rare since the only operations that may potentially communicate their result to the remote narrow-width cluster are **FFN** and **NFN**. These operations contribute for less than 3% on our benchmarks (see Figure 1). It is also important to note that only narrow-width operations can be executed on the narrow-width cluster. The other types of operations containing a narrow-width data, i.e. **FFN, NNF, NFN** and **NFF**, execute on the full-width cluster and read/write their narrow-width data from/to the 16-bit duplicate RF. The 16-bit duplicate RF therefore serves both as a copy of the narrow-width RF and also a local 16-bit RF since many values may be read and written into it without actually modifying their copy. We show indeed in Section 4 that this actually significantly reduces the complexity.

3.2.2 Limited inter-cluster connectivity

We also explored a scheme with limited inter-cluster connectivity to further mitigate overall complexity. In this new organization, we reduce the number of write ports on the 16-bit duplicate RF from 4 to 2. In the narrow-width cluster, we remove the path labelled 2 in Figure 4, meaning that only one 16-bit ALU is now able to propagate its result to the remote RF copy. In the full-width cluster, we note that there is no need to provide 2 write ports on the 16-bit duplicate RF as maintaining it synchronized with the copy is done by the load/store unit. Moreover, our analysis shows that there are only a few operations (**NFN** and **FFN**) executing on the main cluster that produce narrow-width results. Hence, it makes sense to remove the path labelled 1 in Figure 4, but operations producing a narrow-width data will now have to be steered toward the load/store unit. Note however that the 64-bit ALU can still execute operations with narrow-width data. If the operation produces a narrow-width result, this result will have to be written back to the 64-bit RF. Albeit a more efficient use of computing resources can be realized by doing so, it should be noticed that we may however miss some optimization opportunities.

We also propose to optimize the number of inter-cluster communications as a small fraction of the integer operations executing on the full-width cluster use and produce narrow-width data. For this purpose, we advocate using a copy instruction scheme [22] to update the content of a 16-bit register only when necessary. This approach can lead to significant power savings in the interconnect fabric and register files. Nevertheless, using this approach may also have a negative impact on the overall performance as the consuming operations will be delayed until the copy instructions write their results back. To mitigate the performance degradation, we propose to broadcast the value of load operations as done in the basic WPM. It makes sense to do so as we observed that load operations which produce narrow-width data are relatively frequent at runtime. However, a more efficient optimization would be related to the use of a *narrow-width usage predictor* to predict on which cluster a value will likely be consumed. This scheme could be very effective in both reducing the number of communications and improving performance while also eluding the needs of a copy operation scheme. Examining this approach is however left for future research.

4 Complexity Analysis

In this section, we aim to compare the implementation complexity of the baseline partitioned processor presented in Section 3 with the WPM. We consider two implementations of WPM for the comparison. The first one is called *WPM basic*. It corresponds to the basic WPM configuration described in Section 3.2.1. The second implementation is called *WPM limited*. It reduces the number of write ports on the 16-bit duplicate RF of *WPM basic* to 2 (see Section 3.2.2). The comparison will mainly emphasize the complexity-effectiveness of the following main processor structures: the register file, the bypass network, the wakeup and select logic, and the interconnect.

4.1 Register file

The complexity of the register file is mainly characterized by three factors: the area, the access time and the power consumption. For the last two points, we based our estimations on CACTI [28], which we modified appropriately to model a register file[1]. For all the results presented in this section, we assume a 0.13μm CMOS process technology for the register cell implementation.

Silicon area For a conventional multi-ported register file featuring N_{read} ports and N_{write} ports, a total of N_{read} bitlines, N_{read} wordline, $2 \times N_{write}$ bitlines along with N_{write} wordline wires must cross each memory cell. Equation (1) depicts the silicon area that is typically devoted to a physical register featuring N_{regs} registers comprised of R_{width} bits each. In the given equation, ω denotes the width of a wire [29].

$$N_{regs} \times R_{width} \times \underbrace{\omega^2 \times (N_{read} + N_{write}) \times (N_{read} + 2 \times N_{write})}_{cell\ size} \qquad (1)$$

Equation (1) shows that the area devoted to the register file is the product of the number of registers, N_{regs}, the number of bit per register, R_{width}, and the size of a memory cell. The area thus increases linearly with the number of bits and size of the register file, whereas it grows more than quadratically with increasing number of read/write ports. In WPM, the treatment of narrow-width operations is decoupled from that of the other program instructions, yielding a dramatic reduction of R_{width}. This yields a significant area reduction (about 81%) as shown in Table 1, i.e cluster 1 in *WPM basic* and *WPM limited*. In the full-width cluster, the number of write ports on the 64-bit register file is halved. The total area reduction of the RFs in the main cluster (cluster 0 in Table 1) is about 34% for the basic WPM and 43% for the limited-connectivity WPM.

[1]The tag path has been omitted

	Conventional		**WPM basic**		**WPM limited**	
cluster	0	1	0	1	0	1
RF	1	1	2	1	2	1
reg. width (in bits)	64	64	64/16	16	64/16	16
nb of registers	80	80	80/80	80	80/80	80
(R,W) ports	(4,4)	(4,4)	(4,2)/(2,4)	(4,3)	(4,2)/(2,2)	(4,3)
RF area (x ω^2)	491520	491520	245760 + 76800	89600	245760 + 30720	89600
Area reduction	-	-	34%	81%	43%	81%
RF access time	0.6326	0.6326	0.6000/0.5278	0.5342	0.6000/0.4916	0.5342
Access time reduction	-	-	min(6%,16%)	15%	min(6%,22%)	15%
RF nJ/access	0.5431	0.5431	0.4267/0.1977	0.3500	0.4267/0.1571	0.3500
Energy reduction	-	-	21%/63%	35%	21%/71%	35%

Table 1: Estimate of RF complexity. Energy consumption is average energy per read/write access.

Access time The access time of a register file is mainly dominated by the wire propagation delay. As the size and the area of the register file increase, signals propagate along long wires, resulting into larger propagation delays. Since WPM reduces the width of the narrow-width register file by almost a factor of four, shorter word-lines are required to propagate signals. In the narrow-width cluster, this translates into significant access time reduction compared with the conventional processor, about 15% as evidenced in Table 1. In the full-width cluster, the register file access time is dominated by the access time of the 64-bit register file. Since the number of write ports on that register file has been halved, wires length is reduced, resulting into smaller access time (6% less than the baseline model). Therefore, WPM still proves to be more complexity-effective as compared to the conventional processor.

Power consumption The register file layout as well as the number of ports attached to it have a dramatic impact on the power consumption. On each register file access, one or more word-lines go high, while all bit-lines are precharged and sensed in order to determine the state of the attached register cells. In WPM, with the bitwidth size reduction, only a small fraction of these bit-lines are driven. This yields a significant energy reduction of the narrow-width RFs, about 35% as shown in Table 1. The wires length increases with the number of ports, raising the wire capacitance significantly. With WPM, the number of write ports on the 64-bit RF is halved. Hence, the wire capacitance is reduced, explaining the energy savings of the full-width RF (21% to 71%) as shown in Table 1. As a result, WPM consumes less energy on a register file access as compared with the conventional model since the energy per access is lower in all cases.

4.2 Bypass network

Bypassing allows the result of an operation to be consumed by another dependent operation before it gets written to the register file. The complexity of the bypass network is dominated by the number of bypass paths and the time required for a value to be propagated along each of these paths. In our basic WPM design, any operation on the full-width cluster can have its input operand coming from one of six sources: the 64-bit ALU, the load/store unit, the two 16-bit ALUs of the narrow-width clusters, the duplicate 16-bit register file and the 64-bit register file. As a consequence, operand muxes with a fan-in of 6 are required to gate an operand source to its FU. In the conventional processor model, a fan-in of 5 is required instead. Hence, this design slightly increases the complexity of the bypass network at that point. However, we believe the substantial complexity reduction obtained elsewhere (e.g. register file, interconnect) will likely make up for the slight increase in area and access time due to these muxes. Note however that, with the limited-connectivity WPM design, the bypass complexity of both approaches become equivalent. In addition, the bypass complexity on the narrow-width cluster is always reduced since the corresponding fan-in is at most 4.

4.3 Wake-up and select logic

On a modern superscalar machine, an operation waits in the issue window for its source operands to become available before being steered to a particular FU. Assuming a dyadic instruction with up to n distinct producers that may produce a value for each one of its source operands, a total of $2 * n$ comparators must be implemented in the wakeup logic to track all these possible wakeup points. With our baseline WPM design, 4 possible wakeup sources must be monitored on the full-width cluster, compared to 3 on the narrow-width cluster. With the conventional processor, the number of distinct wakeup points is 4. These designs are therefore equivalent with a slight advantage to our approach regarding narrow-width operations. Note however that with the limited-connectivity WPM design, the number of wakeup sources on the full-width cluster drops from 4 to 3.

4.4 Interconnect fabric

The impact of the interconnect on the area, power and delay is expected to grow as the device features become smaller. This trend intensifies on partitioned microarchitectures as long interconnect wires are required to connect distant clusters. Recent research in this line reveal that up to 50% of the total dynamic power consumption is due to the interconnects [19], while a significant performance degradation can be attributed to interconnection delays as device features get smaller [27, 12]. In this section, we show how WPMs can help to tackle these issues.

Interconnect area Figure 6 illustrates the physical layout of a wire. The physical design of wires imposes a minimum spacing between them to mitigate the performance degradations due to sidewall capacitance between parallel adjacent wires. The area occupied by wires is proportional to the number of wires, the width of a wire and its length [27]. In the conventional model, the data transfer between clusters involves sending 64-bit of data. Hence, each inter-cluster connection requires 64 wires. Using WPM, the number of interconnect wires is reduced by four.

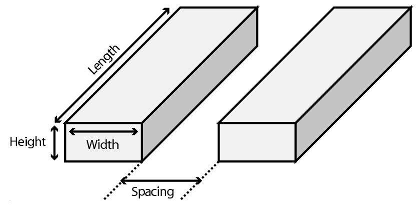

Figure 6: Wire layout

Interconnect power The power dissipated by wires [12] is generally expressed as $P = a * f * N_{wire} * C * V^2$. In the given equation, a represents the activity factor on the wire, f is the wire switching frequency, N_{wire} models the number of wires, C is the wire capacitance, while V is the voltage swing. Since WPM reduces the value of N_{wire} by four, the activity factor a and the switching frequency f get also reduced as fewer wires implies a high likelihood of having smaller values of the effective switching frequency $a * f$. As a consequence, compared to a conventional clustered architecture, WPM provides the potential of significantly reducing the interconnect power consumption.

Opportunity for reduced delays So far, we have assumed working with homogeneous interconnect wires, i.e. the physical characteristics of the wire shown in Figure 6 were kept the same throughout this study. However, it is possible to vary these physical characteristics to reduce the interconnect delay or power consumption. The main idea is to take advantage of the area reduction obtained with WPMs (see paragraph above) to design wires with appropriate characteristics that may accelerate the data communication time between narrow-width and full-width clusters.

$$R_{wire} = \frac{\rho}{(Height - barrier)(width - 2 * barrier)} \tag{2}$$

$$\begin{aligned} C_{wire} &= \epsilon_0 (2K\epsilon_{horiz}\frac{Height}{Spacing} + 2\epsilon_{vert}\frac{Width}{lspacing}) \\ &+ fringe(\epsilon_{horiz}, \epsilon_{vert}) \end{aligned} \tag{3}$$

To see how this may be achieved, consider a wire with resistance R_{wire} and capacitance C_{wire}, shown in Equation (2) and Equation (3), respectively [12]. In the above equations, ρ is the material resistivity, H and W represent the height and the width of the wire, b models the thin barrier that prevents copper from diffusing into surrounding oxide, the various ϵ represent the different dielectric constants for vertical and horizontal capacitors, K accounts for the Miller-effect, while *lspacing* is the spacing between adjacent metal layers. The delay, D, at which data are transfered along wires is proportional to $R_{wire} \times C_{wire}$.

To improve the delay, D, it is sufficient to increase the wire width and spacing in R_{wire} and C_{wire}, respectively. This results into a significant reduction of the delay, but at the cost of increase area overhead. Our design suits well such a purpose since, usually, interconnect wires of less than 20 bits are considered for this type of implementation [2]. In such cases, the resulting area occupancy is somewhat equivalent to that of the conventional architecture with 64-bit wires optimized for bandwidth, i.e. wires with small width and spacing. In addition, since the spacing is increased in C_{wire}, the capacitance itself gets reduced, resulting in a significant reduction of the wire power consumption. It has recently been shown that this technique can be incorporated into modern processors with only marginal increase in complexity [2]. The authors reported a 70% reduction of the delay and 16% reduction of the dynamic power consumption when compared with a wire that is optimized for bandwidth as in the conventional processor case. Considering the potential reductions of the delay and power consumption we just elaborated, we believe that WPMs provide a strong motivation for the deployment of such heterogeneous interconnects.

5 Instruction Steering Mechanism

Various steering schemes [3, 7, 9, 21, 1] have been considered in the literature for allocating instructions to clusters. Most of these schemes relied upon heuristics that strive to minimize communications and workload imbalance. In our study, we showed that both the amount of communication and the load balancing among clusters are very tight to the availability and the distribution of narrow-width operations in programs (see Section 2). Hence, the main challenge with WPM is to reveal all the narrow-width instructions. Several studies [18, 24] have pointed out the strong predictability of data width. These studies show that simple schemes are capable of achieving high data-width coverage, about 95%. This section considers a simple data-width predictor scheme to uncover narrow-width operations. We show how this predictor can be integrated into the steering mechanism to speculatively steer instructions to the proper cluster. Finally, since a wrong data-width prediction lead to an erroneous execution, we show how a replay mechanism corrects this at runtime.

5.1 Data-width predictor

The data-width predictor is used to identify the program instructions that produce a narrow-width result. The bitwidth of memory operations is also predicted. To keep track of previous data-width predictions, we maintain an array of 3-bit saturating counters indexed by the instruction address. Since we use the instruction address to index the array, the table lookup can be performed as soon as the instruction address is known and will therefore not lie on the critical path. An operation is predicted to be narrow-width whenever the counter is saturated. Otherwise, it is considered to be a full-width. In our study, we discriminate between two types of data-width mispredictions. A *conservative misprediction* takes place whenever a data-width larger than the effective data-width is predicted. A conservative does not affect the execution and reflects the number of optimization opportunities that

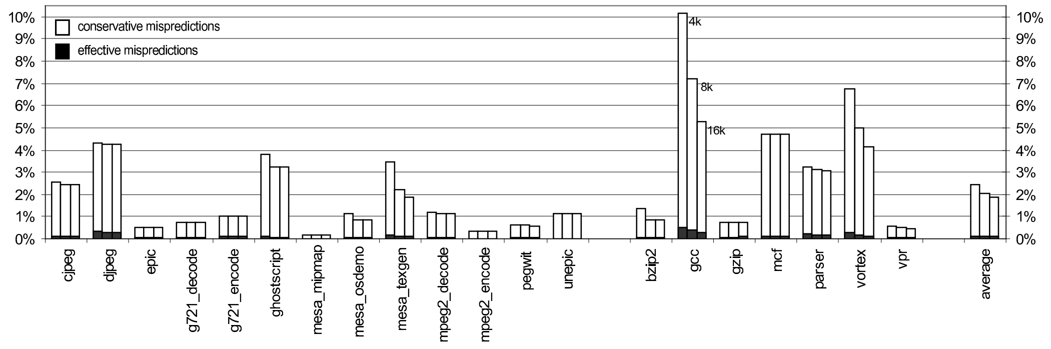

Figure 7: 3-bit bimodal data-width misprediction rates. Each bar corresponds to $4K$, $8K$, and $16K$ entries.

we might miss. An *effective misprediction* occurs whenever a data-width is predicted with a narrower size than the effective data-width. In this latter case, it is necessary to resort to the recovery mechanism described in section 5.3. The saturating counter is updated as follows: it is incremented upon encountering a narrow-width operation and reset to zero upon encountering a full-width operation. The rationale behind doing so is to increase the accuracy of narrow-width operations predictions at the cost of missing some optimization opportunities. Note that adding more hysteresis bits can further reduce the number of effective mispredictions.

Figure 7 illustrates the misprediction rates of this data-width predictor. Table sizes ranging from $4K$ to $16K$ have been considered. The results discriminate between conservative mispredictions and effective mispredictions. In average, around 2.5% of conservative mispredictions and around 0.1% of effective mispredictions are encountered for a predictor table featuring only 4K entries. Note that a few benchmarks (gcc, vortex) encounter a significant conservative misprediction rate, but still exhibit a low effective misprediction rate ($< 0.5\%$). We observed that these benchmarks might benefit from increasing the predictor table size.

5.2 Speculative instruction steering

Our steering mechanism assigns clusters to instructions being renamed according to the location of the source operands along with the decision of the data-width predictor. This simple heuristic considerably simplifies the steering logic. It is possible to consider more complicated heuristics based upon dependency chain information among operations so that performance is maximized. Using such approach, however, would likely place the steering logic in the critical path as more logic will be needed. Note that as there exists only one load/store unit, memory operations are scheduled on the main cluster. We assume that the renaming process is aware of the register file affiliation to clusters, this can be done through an explicit numbering of the physical register addresses, e.g. odd/even numbering.

The steering of instructions to clusters proceeds as follows. Let us consider I, the instruction to be steered. Depending upon the physical location of I's source operands, two cases may occur:

- *All the source operands of I reside in the narrow-width RF (16-bit).* In this case, if the data-width predictor outcome indicates a 16-bit data-width, I has to be dispatched to the narrow-width cluster. Otherwise, I will be assigned to the full-width cluster.
- *Any source operands of I reside in the full-width RF (64-bit).* In this case, we make the conservative decision to dispatch I to the main (full-width) cluster. If the data-width predictor outcome indicates a 16-bit data-width, I will produce its operand on the 16-bit duplicate RF. Otherwise, the result of I will have to be written back onto the 64-bit RF.

5.3 Recovery mechanism

In our study, we assume that a misprediction is detected at the execution stage by a zero detection logic which is available in many current implementations [4]. Whenever an effective misprediction occurs, a replay trap must be triggered by the hardware. In this case, the pipeline in each cluster must be flushed to prevent from any potential resource deadlocks. Prediction tables are updated and instructions are then reassigned accordingly. In our context, using the recovery mechanism is similar in cost to a branch misprediction recovery. Furthermore, both structures involve the same operations. Hence, the logic required for the recovery mechanism can be shared with the data-width predictor so that the hardware complexity is made negligible.

6 WPM Evaluation

In the previous sections, we argued that WPM reduces the complexity of a conventional clustered processor. In this section, we present an evaluation of WPM, showing how it compares with the baseline model.

6.1 Simulation methodology

For our experiments, we used a modified version of the MASE microarchitectural simulator which is based on SimpleScalar [16]. In particular, MASE was modified to model the clustering of integer FUs and the duplication of integer RFs. The modifications take into account the contentions on the cluster interconnect, the issue queues, the physical register files and the register renaming. We also model the bimodal data-width predictor along with the data-width recovery mechanism. Our baseline microarchitecture is derived after the

Parameter	**Configuration**
Fetch queue	16
Branch predictor	bimodal
Data-width predictor	4k 3-bit bimodal
Fetch width	4
Issue width	2 per cluster
Decode width	4
Retire width	4
Issue queue	16 per cluster
FUs (64-bit)	ALU + LD/ST
FUs (16-bit)	2 ALUs
Register file	80
ROB	64
LSQ	16
homog. interconnect delay	2 cycles
heter. interconnect delay	1 cycle
64-bit interconnect power	1.0
16-bit interconnect power	0.84

Table 2: Simulated machine parameters.

MediaBench	
Benchmark	Input
epic	test_image.pgm [.pgm.E]
g721	clinton.g721 [.pcm]
ghostscript	test.ppm
jpeg	testout.ppm [.jpeg]
mesa	-
mpeg2	mei16v2.m2v out.m2v
pegwit	pegwit.enc [.dec]
SPECInt2000	
Benchmark	Input
bzip2	input.graphic
gcc	scilab.i
gzip	input.random
mcf	inp.in
parser	ref.in
vortex	lendian1.raw
vpr	place.in

Table 3: Benchmark applications.

Alpha 21264 [13]. Table 2 summarizes the main machine parameters assumed for the rest of this study. The relative delay estimates as well as the relative power consumption values for the homogeneous and the heterogeneous interconnects are directly derived from [2]. Note that the processing of floating-point operations is done in a separate cluster as in the Alpha 21264. Two WPM configurations are considered for comparison with the baseline processor introduced in Section 3.1. These are the basic WPM described in Section 3.2.1 and the limited-connectivity WPM discussed in Section 3.2.2.

We conducted our evaluation with several benchmarks collected from MediaBench and SPEC2000. All applications were compiled with the PISA gcc compiler using -02 and -funroll-loops optimization flags. Table 3 presents the benchmarks along with the input data sets used for collecting the performance numbers. The applications which execute with fewer than 300 millions instructions were run until completion. For the others, we fast-forwarded past 100 million instructions and simulated over 200 millions instructions.

6.2 Workload balance

Workload balance is a critical factor for performance in a clustered microarchitecture. If the charge on a cluster is unbalanced with respect to another, performance may be significantly impaired since one cluster might be overloaded while the other might be idle for most of the execution. We estimated the workload balance in WPM as the difference in the number of ready instructions in each cluster at each cycle [7], i.e. a zero difference identifies a perfect balance scenario, a difference of one means one cluster has one more instruction than the other, etc. The results presented in Figure 8 consider the workload balance distribution of our basic WPM implementation featuring a 4k bimodal data-width predictor. It is shown

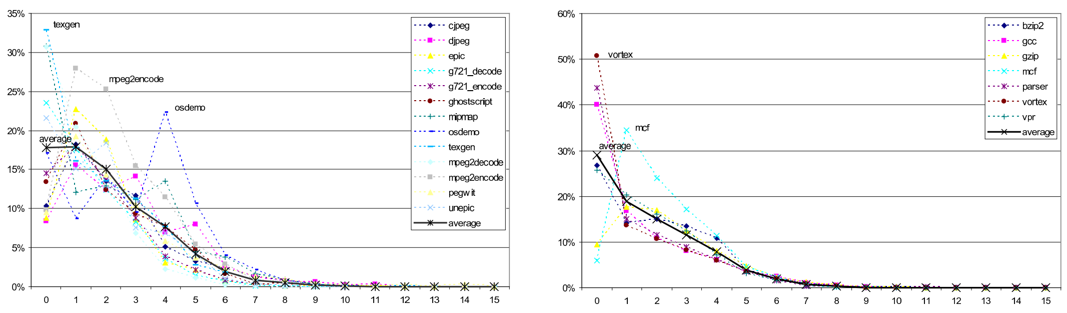

Figure 8: Workload balance distribution for MediaBench (left) and SPEC2k (right) benchmarks.

that, on average, the clusters workload is very well-balanced for about 50% of the program execution ([0-2] differences) whereas it is reasonably balanced for 80% of the execution ([0-5] differences). Considering the asymmetric nature of our WPM implementation, these results appear to be yet very promising. In addition, we only consider a simplified steering heuristic which does not rely upon any workload information. It may therefore be feasible to improve the WPM steering mechanism but at the cost of missing some narrow-width optimizations.

6.3 Performance impact

We considered the balanced RBMS steering heuristic [7] to assign clusters to instructions in the baseline microarchitecture. The balanced RBMS heuristic tries to minimize the number of communications by steering dependent operations to the same cluster while also taking into consideration the charge of the clusters. For a conventional clustered microarchitecture, the performance degradation - measured in terms of overall IPC - primarily depends upon the workload distribution and the number of inter-cluster communications. For a WPM, data-width mispredictions may further affect performance as additional cycles are needed for recovering to a correct state. To exhibit the effectiveness of our proposal without accounting for the impact of mispredictions, we implemented a basic WPM featuring an oracle data-width predictor. Figure 9 shows the performance degradation with the baseline model for different WPM configurations.

As shown in Figure 9, the average performance of the basic WPM with an oracle bitwidth predictor is very close to that of the conventional clustered model that uses a complex steering mechanism. We can, however, notice that some applications (epic, mpeg2_decode, unepic) perform better on the basic WPM featuring an oracle width-predictor than on the baseline model. We observed that this is due to the fact that the workload is very unbalanced on these applications when considering the baseline steering heuristic. Considering a

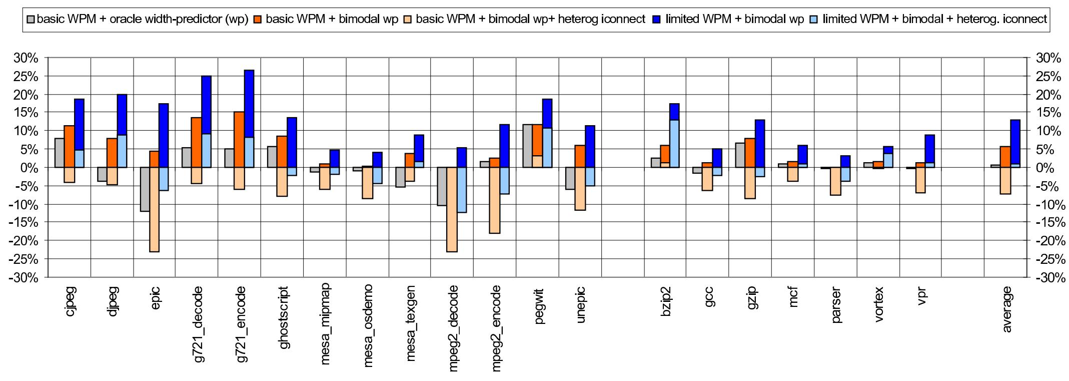

Figure 9: IPC variation.

realistic data-width predictor adds of course some overhead. Figure 9 shows indeed that the performance is degraded by about 6% on average for the basic WPM with a bimodal data-width predictor. First, this degradation accounts for the data-width mispredictions and the cost of the replay. On the other hand, this degradation also depends upon the distribution of the narrow-width operations which is determined by the data-width predictor. By referring to Figure 7, we can observe that some benchmarks (gcc, mcf, vortex) exhibit a high number of conservative mispredictions. However, this doesn't affect the performance since only the effective mispredictions are driving the replay, which really is the performance bottleneck. In addition, the relative good workload balance of these applications also contributes to keep this overhead low. The additional performance degradation observed with the limited WPM scheme is principally due to the copy instructions that need to be scheduled each time an operation must consume an operand value that is only available remotely. These copy operations are meant to synchronize a local RF with its remote copy. The latency of the copy operation is therefore equal the delay of the interconnect. As a result, the dependent operation must stall that long until the operand value is available locally. Figure 9 shows that this may have a detrimental impact on performance, with an average slowdown of almost 13% observed on the benchmarks. However, as noted earlier (see Section 3.2.2), by considering a ***data-width usage predictor***, copy instructions may be issued speculatively before use, just after an operation is issued that may produce a narrow-width value consumed remotely. This approach would be very effective to mitigate the performance degradation observed with the limited WPM scheme. Finally, we also considered an implementation with an heterogeneous interconnect which is able to propagate the data twice as faster than in the baseline clustered model. With this scheme, Figure 9 shows that an average performance improvement of 6% is observed for the basic WPM and a performance degradation of 1% is obtained for the limited WPM as compared to the baseline clustered model. Note that these results consider a very conservative heterogeneous interconnect delay. For instance, as

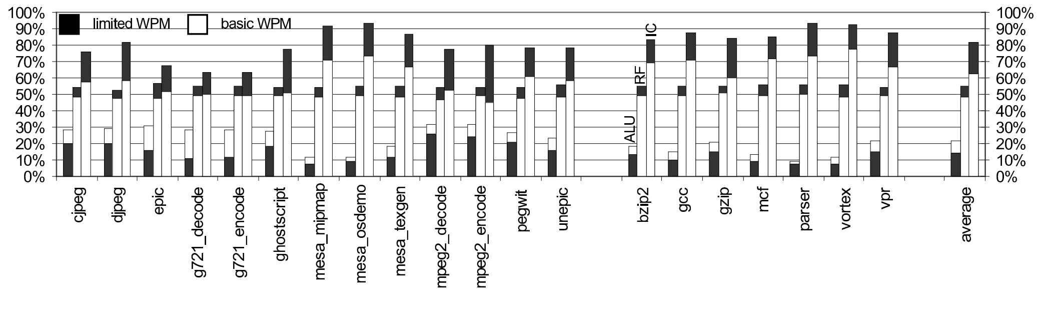

Figure 10: Power savings breakdown (ALU, RF, interconnect (IC)).

for comparison, [2] considers that the wire delay of the heterogeneous interconnect can be reduced by a factor of 3.

6.4 Power reduction

With respect to a conventional superscalar processor, a clustered architecture can achieve significant energy savings due to the decrease in complexity of various critical processor components. Our WPM model benefits from these energy savings while further reducing power dissipation in most of the datapath components. In the first place, the power consumption of the functional units designed to treat the narrow-width operations decreases by a linear factor as noted in [24]. As discussed in section 4.1, the energy consumption of the register files is dramatically lowered due to a decrease in the number of access ports and the width of registers. As a result of exclusively communicating narrow-width values, energy dissipated in the interconnect fabric is also reduced in a significant way. Energy savings in the latter structure result from both the reduction in the number of wires used and the infrequent occurrence of inter-cluster communications. Note that WPM may still involves additional power overhead due to resorting to a speculative scheme and the use of extra multiplexors in the bypass network. For minimizing the power impact of the data-width predictor, we considered a relatively small history table which dissipates a tolerable amount of power [23]. Overall, we believe that the resulting power overhead is negligible compared to the energy savings obtained by the other WPM components.

Figure 10 reports the energy savings realized by the basic and the limited-connectivity WPM implementations as compared to the baseline model. The first bar represents the energy gain realized by the functional units. On average, it can be seen that using the basic WPM implementation yields a 20% energy reduction, whereas considering the limited-connectivity WPM up to 13% of the ALU energy can be saved. This difference stems from the fact that the limited-connectivity model solicits more the use of functional units for processing the copy instructions. One may therefore optimize the energy consumption of the limited-connectivity model by refining the steering heuristic so that it minimizes the inter-cluster communications. Next, we can observe that up to 50% of the RF energy consumption is saved with both the basic and the limited WPM implementations. The difference in the energy savings is not significant as the treatment of the copy instructions in the limited WPM scheme consumes extra energy when the operand value that needs to be communicated is accessed. Power savings realized in the interconnect fabric are the most significant in our approach. Indeed, up to 60% of the energy is saved with the basic WPM implementation, while 80% of energy reduction is achieved with the limited-connectivity model. Note that we did not considered lower-power wires that can further reduce the energy dissipation by a factor of 3 [2]. Considering that modern microprocessors dissipate a large amount of power in the interconnect fabric (50% in the Pentium 4 [19]), our WPM proposal might have therefore a significant impact on the overall microprocessor power consumption. It should be noted that employing a WPM also helps reducing the leakage power in the data-path components. For instance, Balasubramonian et al. [2] denote that the static power in the interconnect fabric can be reduced by a factor ranging from 1.26 to 3 when using a small number of wires. Using narrow-width structures also helps tackling the static energy of the RF and FUs.

7 Related Work

Several techniques have been proposed to tackle the complexity growth associated with scaling up the performance of modern superscalar processors. One approach, partitioning, consists of arranging the resources of a processor into clusters. The other approach considered in this paper consists of tackling the complexity growth problem by exploiting narrow-width data. Each of these approaches are examined in the next sections.

7.1 Partitioned microarchitectures

In a partitioned microarchitecture, the critical processor components are arranged into smaller computational units, called clusters. A cluster represents the complexity-effective counterpart of a centralized design; it can therefore be amenable to sustain higher clock rates. In addition, a clustered microarchitecture can scale to larger issue-width since the parallelism can be distributed across the clusters. Several research papers discussed variants of this type of architecture [21, 9, 1]. Unlike a centralized design, data produced on one cluster may be communicated to another cluster using an interconnect fabric. Since the latency of the interconnect fabric is higher than the intra-cluser latency, the instruction steering logic should try to minimize the inter-cluster communications, while at the same time balancing the workload among the clusters. Multiple heuristics for the instruction steering logic have been proposed in the literature [3, 7, 9, 21, 1]. This paper contrasts with these previous works by considering a new heuristic for the instruction steering logic based on narrow-width data which also proves to balance the workload among clusters.

Attempts to exploit partitioning to reduce the complexity of the register file include the Alpha 21264 [13]. The Alpha 21264 provides each cluster with a copy of the register file (RF). This approach reduces the number of read ports, but requires each RF copy to have the same number of write ports as there are functional units. Seznec et al. [26] improved this approach by reducing the number of write ports through write specialization. Our approach considers a different clustering motivation to tackle processor complexity, i.e. the narrow-width operations property of programs, but still can take advantage of this technique to further reduce the complexity.

In an independent study, Gonzalez et al. [11] recently examined a partitioned design which shares some similarities with our proposal, as it is also based on the narrow-width property of programs. However, while we exploit narrow width to reduce the complexity of some critical datapath components and to reduce power consumption, Gonzalez et al [11] focused on a performance-oriented design. An asymmetric processor organization that distributes the execution core among two clusters is proposed. It features a regular 64-bit cluster and a narrow 20-bit cluster with limited resources but running at twice the clock frequency. For this design, it is therefore desirable to steer most of the program instructions towards the narrow cluster for enabling performance gains. A conventional data-width predictor is used for that purpose. A first difference with our approach is that address computations with invariant high-order bits are also considered; thereby enabling about 75% of the instructions to be executed on the narrow cluster but at the cost of a significant workload imbalance. Despite this partitioned design is shown to slightly improve performance, it is much more difficult to assert its impacts on the processor complexity. Doubling the narrow core frequency also involve to double the frequency of other component logics (e.g. wake-up and issue logic) and leads to a corresponding increase of the power consumption. Many complex artefacts are also required, as for instance the replay logic in case of width misprediction and the TLB mechanisms. No complexity nor power analysis were discussed throughout the paper. Unlike their study, we provide a detailed analysis of the processor complexity factors which demonstrate the feasibility of the WPM model. Our work also introduces many unique features (RF duplicate) which have been shown to further reduce the processor complexity with only little impacts on performance.

7.2 Exploiting narrow-width operands

The observation that most of the applications can be excuted by a narrow datapathwas due to Brooks et al. [4]. They coined as narrow-width operands data that can be represented with less than 16-bits datapath. This section is concerned with optimization schemes that directly make use of narrow-width operands.

7.2.1 Narrow-width optimizations

A microarchitecture which is not aware of the narrow-width data property would exercise the full datapath-width upon each instruction execution, irrespective of the size of the data. Hence, one approach to make efficiently use of the narrow-width data is to optimize a processor for power-efficiency, reducing the effective number of transitions that takes place on the datapath. Implementations of this approach include [4, 5, 6, 24]. Other approaches have considered instead using the empty bitwidth slices on the datapath to increase the effective issue width by allowing several narrow-width data to share the datapath [25, 18, 20].

7.2.2 Register file optimizations

There are two major contributors to the register file complexity: the access time, which is strongly correlated with the number of physical registers, and the area/power which is largely determined by the number of available read/write ports. This section is concerned with some of the recent proposals that exploit narrow-width data to tackle these issues.

Lipasti et al. [17] addressed the case of reducing the pressure on the register file by making effective use of the available physical registers. The idea is based on the observation that the time between the last read and the release dominates a physical register lifetime, whereas mostly only few bits are required to represent the data values stored in these registers. Hence, the authors proposed to early freeing up registers containing narrow-width data by storing their content in the ID field of the register map table. The range of narrow-width values that can be covered by this scheme is therefore strongly dependent on the bit-width of the ID field. For a typical register file of size 64, only 8-bits index would be available in the register map table. To address a larger range of narrow-width values, the size of the map table would have to be scaled accordingly. It is obvious that this is not without consequences on the microarchitecture.

Ergin et al. [8] proposed to exploit narrow-width data by means of register packing. Similar to the SIMD programming model [15], the authors propose to pack several narrow-width data into a single register, making effective use of available registers; thus reducing the pressure on the register file. For a conventional 64-bit register file, for instance, each register is divided into four partitions of size 16-bit each. A narrow-width value is allowed to be represented with any partition combination. This scheme has the potential to complicate the microarchitecture (e.g. the register read stage) as a narrow-width value may now occupy any partition combination inside a register.

Pokam et al. [24] proposed the byte-slice register file to reduce the energy consumption of a register file. A 32-bit conventional register file is partitioned into three slices of size 8-, 8-, and 16-bit each. A data can be placed into the lowest 8-bit slice or into the first two 8-bit slices to represent a narrow-width value of size 8-bit or 16-bit, respectively. When operating in one of these narrow-width modes, the unused upper slices of the register file are put into a drowsy-mode [10] to save static energy. This scheme requires significant modifications to the memory cells.

The proposal by Kondo et al. [14] is somewhat similar to [24] and [8]. They presented a detailed implementation of a bit-partition register file that takes advantage of narrow-width data by dividing a conventional register file into bit-partitions of equal size. Each such bit-partition can hold a different narrow-width data. Hence, this approach is somewhat similar in complexity to [8].

We believe it is not always obvious to assess the impact of these various proposals on the microarchitecture. The register file is at the heart of the processor performance, such that any change it undergoes is likely to have a detrimental effect on the cycle time. On the other hand, clustered architectures have already demonstrated their potential to reduce the complexity growth. Our approach thus naturally combines these two proposals and proves, in effect, to be very effective to eliminate most of the overhead found in previous work.

8 Conclusions

Using 64-bit ISAs in general-purpose computing (PCs, servers, ..) has become mainstream. Therefore datapaths on current processors are 64-bit wide. However, the analysis of workloads show that applications are also containing a very significant proportion of narrow-width operands. Moreover, the use of these narrow-width operands is often evenly distributed over the overall execution. To address this issue, we introduced a new design, called *width-partitioned microarchitectures* (WPM), to help master the hardware complexity of superscalar processors. Through featuring a full-width cluster and a narrow-width cluster, WPM exploits the natural distribution of the narrow-width and the larger data-width operations found in programs to balance the workload among the clusters.

We showed that such a partitioning approach greatly reduces the complexity of existing microarchitectures. We showed indeed that WPM significantly reduces the area and the power overhead of the register file and the interconnect fabric, giving thus rise to more aggressive implementations. In addition, we also demonstrated that WPM allows to break down the complexity of several critical processor structures including the register file, the wakeup and the select logic, and the bypass network. Overall, the performance of WPMs are very promising. Our evaluation showed that, using a WPM architecture instead of a classical 64-bit 2-cluster architecture, more than 50% of the power consumption can be saved on the register file and the interconnect fabric with only a performance overhead of less than 6%. Moreover using narrow-width may allow to use more aggressive cluster interconnection implementations and may even result in performance improvement as illustrated in Section 6.3.

Further research is needed to investigate ways to improve WPMs. For instance, the sensitivity to the bitwidth predictor has not been studied in depth; although it is very likely that more aggressive bitwidth predictors will improve the capability of WPM. In addition, it would be equally interesting to study the scalability of WPM. In particular, an interesting question is how should the narrow-width clusters scale with increasing issue width? On the other hand, this study has only considered the integer functional units for purpose of simplicity. However, it is very likely that other structures may also benefit from WPM as well. Potential structures like these which may be worth to look at include the data cache. Indeed, since the load-store unit can be distributed among clusters, it will be interesting to investigate the issues of partitioning the data cache along with the narrow-width cluster.

References

[1] Balasubramonian, R., Dwarkadas, S., Albonesi, D. Dynamically Managing the Communication-parallelism Trade-off in Future Clustered Processors. In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003.

[2] Balasubramonian, R., Muralimanohar, N., Ramani, K., and Venkatachalapathy, V. Microarchitectural Wire Management for Performance and Power in Partitioned Architectures . In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*, February 2005.

[3] Baniasadi, A., and Moshovos, A. Instruction Distribution Heuristics for Quad-Cluster,Dynamically-Scheduled, Superscalar Processors. In *Proceedings of the 33th International Symposium on Microarchitecture*, Dec. 2000.

[4] Brooks, D., and Martonosi, M. Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*, January 1999.

[5] Canal, R., Gonzales, A., and Smith, J. E. Very Low Power Pipelines Using Significance Compression. In *Proceedings of the 33th International Symposium on Microarchitecture*, December 2000.

[6] Canal, R., Gonzales, A., and Smith, J.E. Software-Controlled Operand-Gating. In *Proceedings of the International Symposium on Code Generation and Optimization*, March 2004.

[7] Canal, R., Parcerisa, J-M., Gonzalez, A. Dynamic Cluster Assignment Mechanisms. In *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, Jan. 2000.

[8] Ergin, O., Balkan, D., Ghose, K., and Ponomarev, D. Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure. In *Proceedings of the 37th International Symposium on Microarchitecture*, December 2004.

[9] Farkas, K. I., Chow, P., Jouppi, N. P., and Vranesic, Z. The Multicluster Architecture: Reducing Cycle Time Through Partitioning. In *Proceedings of the 30th International Symposium on Microarchitecture*, December 1997.

[10] Flautner, K., Sung Kim, N., Martin, S., Blaauw, D., and Mudge, T. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *Proceedings of the 29th International Symposium on Computer Architecture*, May 2002.

[11] Gonzalez, R., Cristal, A., Veidenbaum, A., Pericas, M. and Valero, M. An Asymmetric Clustered Processor based on Value Content. In *Proceedings of the 19th ACM International Conference on Supercomputing*, June 2005.

[12] Ho, R., Mai, K. W., and Horowitz, M. A. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, Apr. 2001.

[13] R. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro*, 19(2):24–36, March 1999.

[14] Kondo, M., and Nakamura, H. A Small, Fast and Low-Power Register File by Bit-Partitioning. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, February 2005.

[15] Larsen, S., and Amarasinghe, S. Exploiting Superword Level Parallelism with Multimedia Instruction Sets. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2000.

[16] Larson, E., Chatterjee, S., and Austin, T. Mase: A novel infrastructure for detailed microarchitectural modeling. In *Proceedings of the 2001 International Symposium on Performance Analysis of Systems and Software*, November 2001.

[17] Lipasti, M. H., Mestan, B. R., and Gunadi, E. Physical Register Inlining. In *Proceedings of the 31th International Symposium on Computer Architecture*, June 2004.

[18] Loh, G. Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth. In *Proceedings of the 35th International Symposium on Microarchitecture*, November 2002.

[19] Magen, N., Kolodny, A., Weiser, U., and Shamir, N. Interconnect-power Dissipation in a Microprocessor. In *Proceedings of the 2004 International Workshop on System Level Interconnect Prediction*, 2004.

[20] Nakra, T., Childers, B.R., and Soffa, M.L. Width-Sensitive Scheduling for Resource-Constrained VLIW Processors. In *Proceedings of the 3th ACM Workshop on Feedback-Directed and Dynamic Optimization*, December 2000.

[21] Palacharla, S., Jouppi, N. P., and Smith, J. E. Complexity-effective Superscalar Processors. *ACM SIGARCH Computer Architecture News*, 25(2):206 – 218, May 1997.

[22] Parcerisa, J-M., Sahuquillo, J., Gonzalez, A., and Duato, J. Efficient Interconnects for Clustered Microarchitectures. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2002.

[23] Parikh, D., Skadron, K., Zhang Y., Barcella, M., and Stan, M.R. Power issues related to branch prediction. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, 2002.

[24] Pokam, G., Rochecouste, O., Seznec, A., and Bodin, F. Speculative Software Management of Datapath-width for Energy Optimization. In *Proceedings of the 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, June 2004.

[25] Sato, T., and Arita, I. Table Size Reduction for Data Value Predictors by Exploiting Narrow Width Values. In *Proceedings of the 14th international conference on Supercomputing*, May 2000.

[26] Seznec, A., Toullec, E., and Rochecouste, O. Register Write Specialization Register Read Specialization: A Path to Complexity-effective Wide-Issue Superscalar Processors. In *Proceedings of the 30th International Symposium on Microarchitecture*, December 2002.

[27] Theis, T. N. The Future of Interconnection Technology. *IBM Journal of Research and Development*, 44(3), 2000.

[28] Wilton, J. E., and Jouppi, N. P. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.

[29] Zyuban, V., and Kogge, P. The Energy Complexity of Register Files. In *Proceedings of the International Symposium on Low Power Electronics and Designs*, Aug. 1998.