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

OLIVIER ROCHECOUSTE

IRISA/INRIA

GILLES POKAM

University of California, San Diego

and

ANDRÉ SEZNEC

IRISA/INRIA

The analysis of program executions reveals that most integer and multimedia applications make heavy use of narrow-width operations, i.e., instructions exclusively using narrow-width operands and producing a narrow-width result. Moreover, this usage is relatively well distributed over the application. We observed this program property on the MediaBench and SPEC2000 benchmarks with about 40% of the instructions being narrow-width operations. Current superscalar processors use 64-bit datapaths to execute all the instructions of the applications. In this paper, we suggest the use of a width-partitioned microarchitecture (WPM) to master the hardware complexity of a superscalar processor. For a four-way issue machine, we split the processor in two two-way clusters: the main cluster executing 64-bit operations, load/store, and complex operations and a narrow cluster executing the 16-bit operations. 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 and the bypass network). The dynamic interleaving of the two instruction types allows maintaining the workload balanced among clusters. WPM also helps to reduce the complexity of the interconnection fabric and of the issue logic. In fact, since the 16-bit cluster can only communicate narrow-width data, the datapath-width of the interconnect fabric can be significantly reduced, yielding a corresponding saving of the interconnect power and area. We explore different possible configurations of WPM, discussing the various implementation tradeoffs. We also examine a speculative steering heuristic to distribute the narrow-width operations among clusters. A detailed analysis of the complexity factors shows using WPM instead of a classical 64-bit two-cluster microarchitecture can save power and silicon area with a minimal impact on the overall performance.

Categories and Subject Descriptors: C.1.4 [Processor Architectures]: Parallel Architectures; C.4 [Performance of systems]:

Authors' addresses: Olivier Rochecouste and André Seznez, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France; emails: orocheco@irisa.fr, seznez@irisa.fr. Gilles Pokam, Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114; email: gpokam@cs.ucsd.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2006 ACM 1544-3566/06/0900-0295 \$5.00

General Terms: Clustered Microarchitecture, Complexity-Effective Design, Narrow-Width Operations

Additional Key Words and Phrases: Power analysis

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 [Ho et al. 2001] 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 [Palacharla et al. 1997]. 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 [Palacharla et al. 1997; Farkas et al. 1997; Balasubramonian et al. 2003]. 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 [Magen et al. 2004]. It is, therefore, desirable to minimize the number of intercluster communications while also keeping balanced the workload among clusters.

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 run-time data, such as the seminal observation made by Brooks and Martonosi [1999] 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 [Brooks and Martonosi 1999; Canal et al. 2000; Pokam et al. 2004; Kondo and Nakamura 2005] or to improve performance [Sato and Arita 2000; Nakra et al. 2000; Loh 2002; Ergin et al. 2004; Lipasti et al. 2004; Gonzalez et al. 2005]. While they do actually help to reduce 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 using a width-partitioned microarchitecture (WPM) to master the hardware complexity of superscalar processors. In a recent and independent study, Gonzalez et al. [2005] examined a clustered design sharing some similarities with WPM, as it is based on the narrow-width data property of programs. A key difference with our approach is that these authors aim at improving performance, while our main concern is to reduce processor complexity and power consumption.

In 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; 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 two-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. WPM is described in detailed in Section 3, while their complexity analysis is discussed in Section 4. The instructions steering mechanism is presented in Section 5. Results are presented in Section 6, while Section 7 discusses the related work. We conclude in Section 8.

2. MOTIVATIONS

In recent work, several authors [Brooks and Martonosi 1999; Loh 2002; Pokam et al. 2004; Ergin et al. 2004] 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 and Martonosi [1999] 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. [2004] considered the basic-block granularity to define narrow-width regions in a program. We formulate a different

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

assumption that considers a narrow-width operation to be an operation where *no operand exceeds 16 bits*, including the destination operand.

2.1 Characterizing Narrow-Width Operations

We have quantified the number of occurrences of these narrow-width operations across the MediaBench and SPEC2000 benchmarks. Our bit-width analysis is exclusively devoted to operations processed through the integer functional unit, including the address calculation. We also considered the operations which execute with operands in the two's complement form by measuring the bit-width of their absolute value. 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 and F for a full-width operand. We use a three-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 that processes a narrow-width along with a full-width source operand and produces a narrow-width result. For the monadic operations, we consider that all of the source operands feature the same width.

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 and Martonosi [1999] 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 significantly reduce 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 to 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

Fig. 2. Distance between narrow-width operations and the other program instructions (and vice versa) at runtime.

heuristics balance the workload among clusters while minimizing intercluster communications.

2.2 Intercluster Communications

Figure 1 also provides an estimate of the average number of communications that take place within WPM. An intercluster 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). Note that this is only valid when every NNN operations are steered to the same cluster. As shown in Figure 1, this concerns roughly 20% of the integer operations. For WPM, this might translate into the worst-case scenario where one operation out of five triggers an intercluster 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 intercluster communications is far below this bound.

2.3 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 WPM might be subject to as follows. For each operation, we have collected the distance separating a narrow-width operation from a full-width one and the distance separating a full-width operation from a narrow-width one. 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 because of 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 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 (WPM) that can efficiently operate on narrow-width operations, as well as on the other program instructions, with reduced complexity. This section describes the implementation of such a four-way WPM design.

3.1 Baseline Model

Our baseline model is derived from the Alpha 21264 [Kessler 1999]. 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. Similar to the Alpha 21264, 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., four 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 Canal and Gonzalez [2000] that 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 intercluster communications are implicitly done by propagating every result to the local and the remote RF. For the producing cluster, data are bypassed in the same cycle to allow 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

Fig. 3. Baseline organization.

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 and the full-width cluster.

Fig. 4. WPM organization.

3.2.1 Intercluster Communications. The need to communicate data between the narrow-width 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 intercluster 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

Fig. 5. Intercluster communication scenario. The suffix indicates the width of the operand.

operations in the full-width cluster will be contributing to the intercluster 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.

3.2.1.1 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 write-back register coherent 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 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. An 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.

3.2.1.2 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 results. 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 (3%); 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 [Parcerisa et al. 2002]. 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 operations processing 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 Intercluster Connectivity. We also explored a scheme with limited intercluster 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 labeled 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 labeled 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 intercluster communications as only a small fraction (3%) 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 [Parcerisa et al. 2002] 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 compare the implementation complexity of the baseline partitioned processor presented in Section 3 with 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 available onto the 16-bit duplicate RF of *WPM basic* from 3 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 [Wilton and Jouppi 1996], which we modified appropriately to model a register file.¹ For all the results presented in this section, we assume a 0.13- μm CMOS process technology for the register cell implementation.

4.1.1 Silicon Area. For a conventional multiported register file featuring N_{read} ports and N_{write} ports, a total of N_{read} bit-lines, N_{read} word-lines, $2 \times N_{write}$ bit-lines, along with N_{write} word-lines, 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 [Zyuban and Kogge 1998].

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

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 I, 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 I) is about 34% for the basic WPM and 43% for the limited-connectivity WPM.

¹The tag path has been omitted.

Table I. Estimate of RF Complexity^a

	Conventional		WPM Basic		WPM Limited	
Cluster # (C)	C0	C1	C0	C1	C0	C1
nb of RFs	1	1	2	1	2	1
reg. width (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)
Area ($\times \omega^2$)	491520	491520	245760 + 76800	89600	245760 + 30720	89600
Area reduction	—	—	34%	81%	43%	81%
Access time (ns)	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%
Energy(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%

^aEnergy consumption is average energy per read/write access.

4.1.2 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 I. 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 in smaller access time (6% less than the baseline model). Therefore, WPM still proves to be more complexity-effective as compared to the conventional processor.

4.1.3 Power Consumption. The register file layout as well as the number of ports attached to it have a dramatic impact on the overall processor power consumption. The power dissipated in this structure can, indeed, account for 10 to 25% of the total chip power consumption [Balasubramonian et al. 2001; Ergin et al. 2004]. 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 bit-width 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 I. The wires length increases with the number of ports, significantly raising the wire capacitance. With WPM, the number of write ports on the 64-bit RF is halved. Hence, the wire capacitance is reduced, which explains the energy savings of the full-width RF (21 to 71%), as shown in Table I. 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.

$$T_{bypass} = 0.5 \times R_{metal} \times C_{metal} \times L^2 \quad (2)$$

The delay of the bypass logic is primarily determined by the amount of time it takes to forward the result value onto the corresponding wire. In other words, it is strongly dependent on the wire length, which is conditioned by the chip layout. Equation (2) displays the delay of the bypass logic, where R_{metal} and C_{metal} correspond to the resistance and parasitic capacitance of metal wires per unit length, respectively, and L is the length of the result wires [Palacharla et al. 1997]. Increasing the fan-in of the operand muxes increases the amount of capacitance C_{metal} on the result wires, thus adding extra delay. Palacharla et al. [1997] do, however, mention that this component of the delay is likely to become less significant as device features become reduced. Assuming so would, therefore, help mitigating the extra complexity associated with the basic WPM design in the bypass network. With the limited-connectivity WPM design, the bypass complexity of both approaches becomes 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, four possible wakeup sources must be monitored on the full-width cluster, compared to three on the narrow-width cluster. With the conventional processor, the number of distinct wakeup points is four. 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 four to three.

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

Fig. 6. Wire layout.

clusters. Recent research in this line reveal that up to 50% of the total dynamic power consumption is because of the interconnects [Magen et al. 2004], while a significant performance degradation can be attributed to interconnection delays as device features become smaller [Theis 2000; Ho et al. 2001]. In this section, we show how WPM can help to tackle these issues.

4.4.1 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 because of 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 [Theis 2000]. In the conventional model, the data transfers between clusters involve sending 64-bit of data. Hence, each intercluster connection requires 64 wires. Using WPM, the number of interconnect wires is reduced by four.

4.4.2 Interconnect Power. The power dissipated by wires [Ho et al. 2001] 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. By decreasing the value of N_{wire} by four, WPM provides a significant reduction in the interconnect power consumption. Note that WPM is also likely to reduce the switching activity factor a . By referring to our analysis (see Section 2.1), transfers of 64-bit and 16-bit data are interleaved for a conventional architecture. A transfer on 64 bits followed by a transfer of 16 bits always induces a switching activity on the 48 highest-order bits. As this scenario never occurs in WPM, our scheme might, therefore, provide another opportunity to reduce the energy dissipated in the interconnect fabric.

4.4.3 Opportunity for Reduced Delays. Thus 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 [Balasubramonian et al. 2005]. The main idea is to take advantage of the area reduction obtained with WPM (see paragraph above) to design wires with appropriate characteristics that may accelerate the

data communication time between narrow-width and full-width clusters.

$$R_{wire} = \frac{\rho}{(Height - barrier)(width - 2 * barrier)} \quad (3)$$

$$C_{wire} = \epsilon_0 \left(2K \epsilon_{horiz} \frac{Height}{Spacing} + 2\epsilon_{vert} \frac{Width}{lspacing} \right) + fringe(\epsilon_{horiz}, \epsilon_{vert}) \quad (4)$$

To see how this may be achieved, consider a wire with resistance R_{wire} and capacitance C_{wire} , shown in Eqs. (3) and (4), respectively [Ho et al. 2001]. 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 transferred 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 such a purpose well since, usually, interconnect wires of less than 20 bits are considered for this type of implementation [Balasubramonian et al. 2005]. 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 [Balasubramonian et al. 2005]. 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 WPM provides a strong motivation for the deployment of such heterogeneous interconnects.

5. INSTRUCTION STEERING MECHANISM

Various steering schemes [Baniasadi and Moshovos 2000; Canal and Gonzalez 2000; Farkas et al. 1997; Palacharla et al. 1997; Balasubramonian et al. 2003] 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 [Loh 2002; Pokam et al. 2004] 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

Fig. 7. Misprediction rates for 3-bit bimodal data-width predictors. Each bar corresponds to 4, 8, and 16 K entries.

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 leads 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 bit-width 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. 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 prediction accuracy of the data-width predictor at the cost of missing some narrow-width operations.

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 misprediction does not affect the execution and reflects the number of optimization opportunities that 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. 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. Tables ranging from 4 to 16 K entries have been considered. The results

Table II. Predictor Coverage for a 4K-entry 3-Bit Bimodal Configuration

MediaBench	jpeg	epic	g721	ghostscript	mesa	mpeg2	pegwit
Predictor coverage	.95	.98	.98	.95	.99	.84	.99
Spec2000	bzip2	gcc	gzip	mcf	parser	vortex	vpr
Predictor coverage	.98	.84	.99	.93	.97	.84	.95

discriminate between conservative 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. For this data-width predictor configuration, we have also measured the data-width prediction coverage, i.e., the fraction of predicted narrow-width operations to actual number of narrow-width operations. Table II reports that a high data-width coverage, on average 94%, is realized with a 4K-entry predictor table. From a WPM viewpoint, these results mean that using a 4K-entry predictor table is sufficient to uncover a large number of narrow-width operations with reasonable impacts on the processor complexity [Parikh et al. 2002]. Note, however, that a few benchmarks (*gcc*, *vortex*) encounter a significant conservative misprediction rate along with a poor data-width coverage, 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: (1) the location of the source operands and (2) the decision of the data-width predictor. We assume for doing (1) that the renaming process is aware of the register file affiliation to clusters. For instance, this can be done through an explicit numbering of the physical register addresses, e.g., odd/even numbering. After assigning clusters to instructions, the renaming logic allocates a free physical register for each instruction producing a result. Note that as there exists only one load/store unit, memory operations are scheduled on the main cluster. Considering this simple heuristic permits to greatly simplify the steering logic. It would have been 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.

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 have to be written back onto the 64-bit RF.

5.3 Recovery Mechanism

We suggest a simplistic mechanism to recover from any data-width misprediction. We assume that a data-width misprediction is detected at the execution stage by means of a zero detection logic, which is available in many implementations [Brooks and Martonosi 1999]. Note that, unlike Loh [2002], simply relying on the ALUs overflow detection logic is not sufficient in our context, since we have to predict the bit-width of data fetched from memory as well. For the recovery mechanism, we consider a refetch replay scheme [Kim and Lipasti 2004]. Upon detecting a data-width misprediction, all instructions fetched after the misspeculated instruction are flushed from the pipeline and the fetch process starts over with the instructions following the misspeculated instruction. Prediction tables are then updated accordingly so that the recently fetched instructions could be assigned to a more appropriate cluster. Our recovery mechanism is similar to the one used for recovering from a branch misprediction. Hence, we could assume that the logic required for these recovery mechanisms can be shared so that the hardware complexity is made negligible. Note, that implementing a more advanced recovery mechanism [Gonzalez et al. 2005] would likely reduce the impacts on performance, but at the cost of increasing the overall hardware complexity.

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 [Larson et al. 2001]. 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 interconnects, 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 Alpha 21264 [Kessler 1999]. Table III 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 Balasubramonian et al. [2005] and depicted in Table IV. 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.

Table III. Simulated Machine Parameters

Parameter	Configuration	Parameter	Configuration
Fetch width	4	FUs (16-bit)	2 ALUs
Decode width	4	Register file size	80 per RF
Issue width	2/cluster	ROB size	64
Retire width	4	LSQ size	16
Fetch queue size	16	Branch predictor	16k-entry GShare
Issue queue size	16/cluster	Data-width predictor	4k-entry 3-bit bimodal
FUs (64-bit)	ALU + LD/ST	Misprediction penalty	min. 6 cycles

Table IV. Delay, Energy, and Power Parameters Assumed for Experimentations

Homogeneous interconnect delay	2 cycles
Heterogeneous interconnect delay	1 cycle
Homogeneous interconnect relative energy	1.0
Heterogeneous interconnect relative energy	0.84
64-bit RF (4R/4W ports) leakage power	46.50 μ W
64-bit RF (4R/2W ports) leakage power	27.08 μ W
16-bit RF (2R/4W ports) leakage power	11.16 μ W
16-bit RF (4R/3W ports) leakage power	9.19 μ W
16-bit RF (2R/2W ports) leakage power	6.30 μ W
4k-entry 3-bit data-width predictor (1R/1W ports) energy	0.112 nJ

Table V. Benchmark Applications

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

We conducted our evaluation with several benchmarks collected from MediaBench and SPEC2000. All applications were compiled for SimpleScalar PISA instruction set with gcc 2.7.2.3 using `-O2` and `-funroll-loops`. Note that some applications were omitted because of difficulties compiling them for SimpleScalar PISA as they use extra libraries that are not part of the SimpleScalar toolset. Table V presents the benchmarks along with the input data sets used for collecting the performance numbers. The applications were simulated until completion or for a maximum of 300 million instructions after skipping the first 1 billion 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. A possible metric for estimating the workload balance of a cluster could be the number of instructions in the instruction queue [Canal and Gonzalez 2000]. However, this metric is not fully appropriate as it does not consider the parallelism among instructions. Canal and Gonzalez [2000] proposed a more suitable metric that takes into account both the number of instructions and the parallelism among them. They consider that the workload is balanced when both clusters have the same number of ready instructions; therefore, the difference in the number of ready instructions could be used as a metric of the workload imbalance. We relied upon this metric to estimate the workload balance of the baseline and WPM implementations. For instance, a zero difference identifies a perfect balance scenario, whereas a difference of one means that a cluster has one more instruction than the other, etc.

Figure 8 presents the results of the workload balance distribution for the baseline and WPM architectures over the MediaBench and Spec2000 benchmarks. Figures 8a and b depict the workload balance distributions of the baseline architecture. We considered the balanced RBMS heuristic [Canal and Gonzalez 2000] 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. Figures 8c and d display the workload balance distributions of our basic WPM implementation featuring a 4K-entry 3-bit bimodal data-width predictor. Overall, it can be seen that the workload balance distributions are very similar for both the baseline and WPM architectures. This similarity mainly stems from the fact that nonhomogeneous designs are considered in our study. On average, the various workload balance distributions reveal that the clusters workload is very well-balanced for about 60% 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 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

For a conventional clustered microarchitecture, the performance degradation—measured in terms of overall IPC—primarily depends upon the workload distribution and the number of intercluster communications. For 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 bit-width predictor is very close to that of the conventional clustered model that uses a complex steering mechanism. We can,

Fig. 8. Workload balance distribution for MediaBench and Spec2000 benchmarks.

however, notice that some applications (*epic*, *mesa*, *mpeg2*) perform better on the basic WPM featuring an oracle width-predictor than on the baseline model. We observed that this is because of the fact that the workload is very unbalanced on these applications when considering the baseline steering heuristic.

Considering a 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*,

Fig. 9. Performance impact assuming a 4k-entry predictor table.

vortex) exhibit a high number of conservative mispredictions. However, this does not 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 because of 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 interconnects. 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.

6.3.1 Assuming Heterogeneous Interconnects. Finally, we also considered an implementation with heterogeneous interconnects, which are able to propagate the data twice as fast as the baseline clustered model [Balasubramanian et al. 2005]. With this scheme, Figure 10 depicts that a significant peak performance gain is achieved over some benchmarks. For instance, performance improvements of 27 and 23% are realized over *epic*, using the basic WPM and the limited WPM, respectively. On average, we could notice a performance gain of 6% with the basic WPM and a performance degradation of 1% with the limited WPM, as compared to the baseline model. These results point out that the application's performance is very tight to the number and the latency of communications in a clustered microarchitecture. Hence, implementing an heterogeneous interconnect fabric appears to be a very promising approach to mitigate the performance degradations of WPM. Note, however, that these results

Fig. 10. Performance impact assuming *heterogeneous interconnects* and a 4k-entry predictor table.

consider a very conservative heterogeneous interconnects delay. For instance, as for comparison, Balasubramonian et al. [2005] considers that the wire delay of the heterogeneous interconnects can be reduced by a factor of 3.

6.4 Power Estimation

With respect to a conventional superscalar processor, a clustered architecture can achieve significant energy savings because of 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 data-path 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 Pokam et al. [2004]. This has to be taken into consideration as integer units contribute up to 10% of the total chip power in the Alpha 21264 [Gowan et al. 1998]. As discussed in Section 4.1, the energy consumption of the register files is dramatically lowered because of 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 significantly reduced. Energy savings in the latter structure result from both the reduction in the number of wires used and the infrequent occurrence of intercluster communications.

6.4.1 Dynamic Power Reduction. Figures 11, 12, and 13 report the energy savings realized by the basic and the limited-connectivity WPM implementations as compared to the baseline model. Figure 11 shows 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 limitedconnectivity model by refining the steering heuristic so that it minimizes the intercluster communications. Next, we can observe in Figure 12 that up to 50% of the RF energy consumption is saved with both the basic and the limited WPM

Fig. 11. Energy savings achieved in functional units.

Fig. 12. Energy savings achieved in the register file.

Fig. 13. Energy savings achieved in the interconnect fabric.

Fig. 14. Leakage energy savings realized in the register file.

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, as depicted in Figure 13, 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 consider lower-power wires that can further reduce the energy dissipation by a factor of 3 [Balasubramanian et al. 2005]. Considering that modern microprocessors dissipate a large amount of power in the interconnect fabric (50% in the Pentium 4 [Magen et al. 2004]), our WPM proposal might have, therefore, a significant impact on the overall microprocessor power consumption.

6.4.2 Static Power Reduction. As technology scales to deeper submicron processes, static power is becoming a dominant fraction of the total chip power consumption. Using narrow-width structures, WPM helps reducing leakage power in several datapath components: RF, FUs and interconnect fabric. Balasubramanian et al. [2005] 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. With Hotleakage [Zhang et al. 2003], we approximated the amount of leakage power dissipated by the register file. Table IV reports the leakage power values for the different RF configurations considered in this study, assuming a 0.13μ process. As displayed, reducing the number of read/write ports and the width of registers has a significant impact on the register file leakage power consumption. Leakage power is, indeed, almost reduced by a factor 2, as the number of RF write ports is halved. It is further reduced by a linear factor, i.e., 4, as the width of registers gets narrower, i.e., 16-bit. Nonetheless, using WPM may involve higher leakage energy consumption because of the longer execution time induced by wrong data-width predictions. We measured the leakage energy consumed over a program run by the basic and limited WPM schemes. Figure 14 reports the leakage energy savings realized in the register file. On average, we witness a 50% decrease of the leakage energy consumption

for both the basic and the limited WPM schemes. Leakage energy savings are better in the basic than in the limited scheme. This is because of longer execution time using a limited-connectivity scheme, which offsets the reduction of the number of write ports on the register file.

6.4.3 Power Overhead. WPM may involve additional power consumption because of resorting to a speculative scheme and the use of extra multiplexers in the bypass network. We approximated the power overhead because of the 4K-entry 3-bit bimodal data-width predictor by means of CACTI [Wilton and Jouppi 1996]. Our power estimations do not take into account the energy consumed by the components of the tag path as we consider a tagless structure. We simulated a predictor table provided with two access ports (one read port, one write port) assuming that four consecutive 3-bit counters can be accessed simultaneously. We consider that the predictor lookups and updates are done at fetch and commit time, respectively. Table IV depicts the per access energy cost of the considered data-width predictor. The lack of an appropriate tool to measure the overall processor power consumption makes it hard to report the data-width predictor power overhead. In order to have a rough estimate, we have compared the data-width predictor power consumption with that of a four read/four write ports 64-bit register file. On average, we measured that accessing the data-width predictor represents a 6% overhead relative to the power requirements of the register file over a program run. This derives from the fact that the number of data-width predictor lookups and updates are of inferior magnitude as compared to the number of register file access, i.e., 12% on average. Therefore, the power overhead of the data-width predictor is negligible as compared to the savings obtained by the other WPM components. It should be noted that we could also have used a multibanked predictor table to further mitigate the power overhead.

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 [Palacharla et al. 1997; Farkas et al. 1997; Balasubramonian et al. 2003]. 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 can be of order of magnitude higher than that of the cluster, the instruction steering logic should strive to minimize the intercluster communications, while, at the same time, balancing the workload among clusters. Multiple heuristics for the instruction steering logic have been proposed in the literature [Banasadi and Moshovos 2000; Canal and Gonzalez 2000; Farkas et al. 1997; Palacharla et al. 1997; Balasubramonian et al. 2003]. This paper contrasts with these previous work 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 [Kessler 1999]. 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. [2002] improves on that by further clustering the RF among group of clusters, thereby reducing the number of write ports. 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 reduce the complexity more aggressively.

Other studies proposed a nonhomogeneous clustered microarchitecture to either improve performance [Fields et al. 2002; Albonesi and Dropscho 2004; Gonzalez et al. 2005] or reduce power consumption [Seng et al. 2001; Sato et al. 2002; Baniasadi and Moshovos 2002; Ramani et al. 2004]. In these studies, the processor core is generally distributed among clusters that can operate at different clock rates. Diverse steering strategies were adopted to exploit this asymmetric design. Seng et al. [2001] suggested using slower functional units for performing non-critical instructions. Albonesi and Dropscho [2004] proposed to steer instructions toward clusters in order to better match the applications ILP.

In a recent and independent proposal, Gonzalez et al. [2005] examined a nonhomogeneous clustered design. This design shares some similarities with our proposal, since it is also based on the narrow-width property of programs. However, while we exploit this property to reduce the complexity of some critical datapath components and to reduce power consumption, Gonzalez et al. [2005] focused on a performance-oriented design. An asymmetric processor organization that distributes the execution core between 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 toward the narrow cluster for enabling performance gains. A conventional data-width predictor is used for that purpose. In contrast with our approach, these authors also considered processing address computations with invariant high-order bits. Doing so enables about 75% of the instructions to be executed on the narrow-width cluster but at the cost of a significant workload imbalance. Although marginal gains are observed for their partitioned design, it is much more difficult to assert its impacts on the processor complexity. Doubling the narrow-core frequency also

involves doubling 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 artifacts are also required, as, for instance, the replay logic in case of width misprediction and the TLB mechanisms. Neither complexity nor power analysis were discussed throughout the paper. Unlike their study, we provide a detailed analysis of the processor complexity factors demonstrating the feasibility of the WPM model. Our work also introduces many unique features (e.g., 16-bit RF duplicate into the 64-bit cluster), which have been shown to further reduce the processor complexity with only little impact on performance.

7.2 Exploiting Narrow-Width Operands

The observation that most of the applications only need part of the datapath-width to execute is due to Brooks and Martonosi [1999]. They designated as narrow-width operands any data that can be represented with less than 16-bits. 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 efficient 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 Brooks and Martonosi [1999], Canal et al. [2000, 2004], and Pokam et al. [2004]. Other approaches have considered instead using the empty bit-width slices on the datapath to increase the effective issue width by allowing several narrow-width data to share the datapath [Sato and Arita 2000; Loh 2002; Nakra et al. 2000].

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. [2004] 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 the early freeing up of 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 an 8-bit 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. [2004] proposed the exploitation of narrow-width data by means of register packing. Similar to the SIMD programming model [Larsen and Amarasinghe 2000], the authors propose packing 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 16-bit each. A narrow-width value is allowed to be represented using 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. [2004] 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 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 [Flautner et al. 2002] to save static energy. This scheme requires significant modifications to the memory cells.

The proposal by Kondo and Nakamura [2005] is somewhat similar to Pokam et al. [2004] and Ergin et al. [2004]. 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 bit-partition can hold a different narrow-width data. Hence, this approach is somewhat similar in complexity to Ergin et al. [2004].

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, etc.) has become mainstream. Therefore datapaths on current processors are 64-bit wide. However, the analysis of workloads show that applications also contain 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. Featuring a full 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 that WPM significantly reduces the area and the power overhead of the register file and the interconnect fabric,

giving rise to more aggressive implementations. In addition, we also demonstrated that WPM allows the break down of 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 WPM is very promising. Our evaluation showed that by using a WPM architecture instead of a classical 64-bit two-cluster architecture, more than 50% of the power consumption can be saved on the register file and the interconnect fabric, with a performance overhead of less than 6%. Moreover using narrow-width may allow 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 WPM. For instance, the sensitivity to the bit-width predictor has not been studied in depth, although it is very likely that more aggressive bit-width 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 looking 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

- ALBONESI, L. C. D. AND DROPSHO, S. 2004. Dynamically matching ILP characteristics via a heterogeneous clustered microarchitecture. In *Proceedings of IBM Watson Conference on the Interaction Between Architecture, Circuits, and Compilers*.
- BALASUBRAMONIAN, R., DWARKADAS, S., AND ALBONESI, D. 2001. Reducing the complexity of the register file in dynamic superscalar processor. In *Proceedings of the 34th International Symposium on Microarchitecture*.
- BALASUBRAMONIAN, R., DWARKADAS, S., AND ALBONESI, D. 2003. Dynamically managing the communication-parallelism trade-off in future clustered processors. In *Proceedings of the 30th International Symposium on Computer Architecture*.
- BALASUBRAMONIAN, R., MURALIMANOHAR, N., RAMANI, K., AND VENKATACHALAPATHY, V. 2005. Microarchitectural wire management for performance and power in partitioned architectures. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*.
- BANIASADI, A. AND MOSHOVOS, A. 2000. Instruction distribution heuristics for quad-cluster, dynamically-scheduled, superscalar processors. In *Proceedings of the 33th International Symposium on Microarchitecture*.
- BANIASADI, A. AND MOSHOVOS, A. 2002. Asymmetric-frequency clustering: A power-aware back-end for high-performance processors. In *ISLPED '02: Proceedings of the 2002 international symposium on Low power electronics and design*.
- BROOKS, D. AND MARTONOSI, M. 1999. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*.
- CANAL, R. AND GONZALEZ, J.-M. P. A. 2000. Dynamic cluster assignment mechanisms. In *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*.
- CANAL, R., GONZALEZ, A., AND SMITH, J. E. 2000. Very low power pipelines using significance compression. In *Proceedings of the 33th International Symposium on Microarchitecture*.
- CANAL, R., GONZALEZ, A., AND SMITH, J. E. 2004. Software-controlled operand-gating. In *Proceedings of the International Symposium on Code Generation and Optimization*.

- ERGIN, O., BALKAN, D., GHOSE, K., AND PONOMAREV, D. 2004. Register packing: Exploiting narrow-width operands for reducing register file pressure. In *Proceedings of the 37th International Symposium on Microarchitecture*.
- FARKAS, K. I., CHOW, P., JOUPPI, N. P., AND VRANESIC, Z. 1997. The multicluster architecture: Reducing cycle time through partitioning. In *Proceedings of the 30th International Symposium on Microarchitecture*.
- FIELDS, B., BODIK, R., AND HILL, M. D. 2002. Slack: Maximizing performance under technological constraints. In *Proceedings of the 29th annual international symposium on Computer architecture*.
- FLAUTNER, K., KIM, N. S., MARTIN, S., BLAAUW, D., AND MUDGE, T. 2002. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th International Symposium on Computer Architecture*.
- GONZALEZ, R., CRISTAL, A., VEIDENBAUM, A., PERICAS, M., AND VALERO, M. 2005. An asymmetric clustered processor based on value content. In *Proceedings of the 19th ACM International Conference on Supercomputing*.
- GOWAN, M. K., BIRO, L. L., AND JACKSON, D. B. 1998. Power considerations in the design of the alpha 21264 microprocessor. In *DAC '98: Proceedings of the 35th annual conference on Design automation*.
- HO, R., MAI, K. W., AND HOROWITZ, M. A. 2001. The future of wires. *Proceedings of the IEEE* 89, 4 (Apr.), 490–504.
- KESSLER, R. E. 1999. The Alpha 21264 Microprocessor. *IEEE Micro* 19, 2 (Mar.), 24–36.
- KIM, I. AND LIPASTI, M. H. 2004. Understanding scheduling replay schemes. In *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*.
- KONDO, M. AND NAKAMURA, H. 2005. A small, fast and low-power register file by bit-partitioning. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*.
- LARSEN, S. AND AMARASINGHE, S. 2000. Exploiting superword level parallelism with multimedia instruction sets. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*.
- LARSON, E., CHATTERJEE, S., AND AUSTIN, T. 2001. MASE: A novel infrastructure for detailed microarchitectural modeling. In *Proceedings of the 2001 International Symposium on Performance Analysis of Systems and Software*.
- LIPASTI, M. H., MESTAN, B. R., AND GUNADI, E. 2004. Physical register inlining. In *Proceedings of the 31th International Symposium on Computer Architecture*.
- LOH, G. 2002. Exploiting data-width locality to increase superscalar execution bandwidth. In *Proceedings of the 35th International Symposium on Microarchitecture*.
- MAGEN, N., KOLODNY, A., WEISER, U., AND SHAMIR, N. 2004. Interconnect-power dissipation in a microprocessor. In *Proceedings of the 2004 International Workshop on System Level Interconnect Prediction*.
- NAKRA, T., CHILDERS, B. R., AND SOFFA, M. L. 2000. Width-sensitive scheduling for resource-constrained VLIW processors. In *Proceedings of the 3th ACM Workshop on Feedback-Directed and Dynamic Optimization*.
- PALACHARLA, S., JOUPPI, N. P., AND SMITH, J. E. 1997. Complexity-effective superscalar processors. *ACM SIGARCH Computer Architecture News* 25, 2 (May), 206–218.
- PARCERISA, J.-M., SAHUQUILLO, J., GONZALEZ, A., AND DUATO, J. 2002. Efficient interconnects for clustered microarchitectures. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*.
- PARIKH, D., SKADRON, K., ZHANG, Y., BARCELLA, M., AND STAN, M. R. 2002. Power issues related to branch prediction. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*.
- POKAM, G., ROCHECOUSTE, O., SEZNEC, A., AND BODIN, F. 2004. 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*.
- RAMANI, K., MURALIMANO HAR, N., AND BALASUBRAMONIAN, R. 2004. Microarchitectural techniques to reduce interconnect power in clustered processors. In *Proceedings of the 5th Workshop on Complexity-Effective Design*.

- SATO, T. AND ARITA, I. 2000. Table size reduction for data value predictors by exploiting narrow width values. In *Proceedings of the 14th international conference on Supercomputing*.
- SATO, T., CHIYONOBU, A., AND ARITA, I. 2002. Energy reduction via critical path prediction. In *Proceedings of the 5th Workshop on Complexity-Effective Design*.
- SENG, J. S., TUNE, E. S., AND TULLSEN, D. M. 2001. Reducing power with dynamic critical path information. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*.
- SEZNEC, A., TOULLEC, E., AND ROCHECOUSTE, O. 2002. Register write specialization register read specialization: A path to complexity-effective wide-issue superscalar processors. In *Proceedings of the 30th International Symposium on Microarchitecture*.
- THEIS, T. N. 2000. The future of interconnection technology. *IBM Journal of Research and Development* 44, 3.
- WILTON, J. E. AND JOUPPI, N. P. 1996. CACTI: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits* 31, 5 (May), 677–688.
- ZHANG, Y., PARIKH, D., SANKARANARAYANAN, K., SKADRON, K., AND STAN, M. 2003. Hotleakage: A Temperature-aware model of subthreshold and gate leakage for architects. Tech. Rep. CS-2003-05, University of Virginia, Department of Computer Science. March.
- ZYUBAN, V. AND KOGGE, P. 1998. The energy complexity of register files. In *Proceedings of the International Symposium on Low Power Electronics and Designs*.

Received August 2005; revised December 2005 and April 2006; accepted May 2006