

Reducing Wire Delay Penalty through Value Prediction

Joan-Manuel Parcerisa and Antonio González
Dept. d'Arquitectura de Computadors, Universitat Politècnica de Catalunya
c/. Jordi Girona, 1-3 Mòdul C6
08034 Barcelona, Spain
{jmanel,antonio}@ac.upc.es

Abstract

In this work we show that value prediction can be used to avoid the penalty of long wire delays by predicting the data that is communicated through these long wires and validating the prediction locally where the value is produced. Only in the case of misprediction, the long wire delay is experienced.

We apply this concept to a clustered microarchitecture in order to reduce inter-cluster communication. The predictability of values provides the dynamic instruction partitioning hardware with less constraints to optimize the trade-off between communication requirements and workload balance, which is the most critical issue of the partitioning scheme. We show that value prediction reduces the penalties caused by inter-cluster communication by 18% on average for a realistic implementation of a 4-cluster microarchitecture.

1. Introduction

Recent studies point out that two major problems for scaling-up current superscalar microarchitectures will be the growing impact of wire delays [1, 2, 13], and the increasing complexity of some critical components, such as the issue logic, the bypass, the register file and the rename logic [16], since they may have a direct influence on the clock cycle time.

One of the proposed solutions to this problem is based on clustering. In a clustered microarchitecture some of the critical components are partitioned into simpler structures, and the impact of wire delays is reduced as far as signals are kept local within the clusters. In a clustered architecture, deciding which instructions are executed in each cluster becomes a key issue. We will refer to this task as code partitioning. A code partitioning scheme determines how the dynamic instruction stream is split among the different clusters. Data dependences among instructions in different partitions correspond to inter-cluster communications, which use long wires and have a high associated latency. In

this work we will focus on dynamic mechanisms for partitioning the instruction stream, implemented through a small hardware that steers instructions to clusters, the steering logic.

Minimizing the impact of inter-cluster communication delays is one of the main objectives of any code partitioning scheme. The solution that we propose in this work is to eliminate data dependences that cross the partition boundaries by predicting the values that flow among them. Value prediction has been largely investigated in the context of superscalar processors, and it is not our purpose to design another predictor but to investigate its potential to reduce slow inter-cluster communications on a clustered architecture, and to provide a new source of performance improvements. In this paper we show that value prediction can significantly improve the performance of the steering logic by providing a less dense data dependence graph which results in less communication requirements and better opportunities to balance the workload.

It is known that the IPC of a clustered architecture is lower than that of an equivalent centralized organization without the inter-cluster communication delays. It is also expected that value prediction may increase the IPC in both cases. However, we show that the clustered architecture benefits from value prediction more than a centralized one, since value prediction removes some inter-cluster communications. In particular, we show that the IPC degradation caused by inter-cluster communications can be reduced by 18% through a simple value prediction scheme when the steering logic is designed to take advantage of the value predictor.

The rest of this paper is organized as follows. Section 2 presents the assumed clustered architecture. Section 3 presents the partitioning heuristic implemented by the steering logic, and its specific adaptations to take advantage of value prediction, and provides a performance evaluation. In Section 4, a sensitivity analysis regarding communication latency, communication bandwidth and predictor table size is performed. Section 5 reviews other related

work. And finally, the main conclusions are summarized in Section 6.

2. Microarchitecture

The target processor microarchitecture is a clustered implementation of an 8-way out-of-order issue superscalar processor with a 6 stage pipeline (fetch, decode, issue, execute, writeback and commit). The processor front-end (fetch and decode stages) is a centralized structure, and we assume that it has an aggressive instruction fetch mechanism to stress the instruction issue and execution subsystems. The processor core is divided into N homogeneous clusters: each cluster has its own instruction queue, a physical register file, a set of functional units, and the corresponding data bypasses among these functional units. We experiment with several configurations (targeting different technologies and clock rates) having 1, 2 and 4 clusters. While the register file access time and issue time are assumed to be constant in all cases, structure sizes are scaled down with the degree of clustering. Therefore, register files have respectively 128, 80 and 56 physical registers per cluster, and instruction queue lengths are 64, 32 and 16 entries. The reorder buffer length (128 entries), the total number of functional units (8), and the total issue width (8) is kept constant through all the configurations. The main architectural parameters are described in Section 2.4.

Local bypasses within a cluster are responsible for forwarding result values produced in the cluster to the inputs of the functional units in the same cluster. A local bypass takes 0 cycles, i.e. a value produced in cycle i can be an input of a local functional unit in cycle $i+1$. Inter-cluster bypasses are responsible for forwarding values among functional units of different clusters. Since inter-cluster bypasses require long wires, they will likely take several cycles in future technologies [1]. Therefore, we have assumed a one-cycle latency for inter-cluster bypasses in the basic configurations, although we also evaluate the effects of longer latencies. Latency is not the only penalty of inter-cluster communications. Also bandwidth is relevant, since it directly affects the number of register file write ports, and the complexity of the bypass logic. We first assume an unbounded number of interconnection paths in order to isolate our experiments from the effect of possible bandwidth bottlenecks, and then we evaluate the effects of having a limited inter-cluster communication bandwidth.

2.1. Handling register copies

In a processor with N clusters, instructions are renamed at the decode stage, by means of a register map table with N fields per logical register, that allows up to N different mappings of the same register. An additional bit per

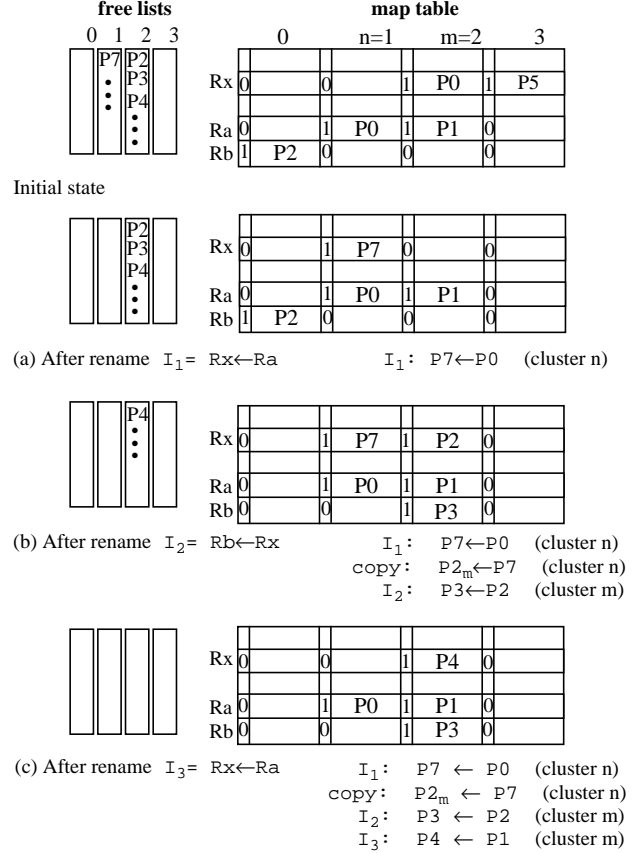


Figure 1. Example of renaming 3 instructions. I_2 requires to copy Rx from cluster n to m

field indicates whether the mapping is valid and if so, it points to a physical register in the corresponding cluster. All logical registers must have at least one valid mapping. Each cluster has a free pool of physical registers from where they are allocated when needed.

When an instruction I_1 is decoded (see Figure 1(a)), and it is assigned to cluster n by the steering logic, its source operands are renamed by looking at the field n of the map table. If the instruction has a destination register Rx, a new free physical register is allocated from the free-list of cluster n , the new mapping is written in the field n of the map table, and the other fields are set invalid, to denote that Rx is not currently mapped to any physical register in these clusters.

Let us assume that a subsequent dependent instruction I_2 , that reads register Rx, is decoded and steered to cluster m , different from n (see Figure 1(b)). When I_2 is renamed, the field m of the map table entry for register Rx is found invalid. Normal instructions are not allowed to access the register files of remote clusters. Instead, they require remote operands to be copied from one register file to another by means of special *copy* instructions generated on demand during the renaming stage. Therefore, in this example a

new physical register is allocated in cluster m , to store the copy of register Rx for future reuse, and its mapping is written in the field m of the map table entry for Rx. This field becomes valid, and its mapping is used to rename the source operand of I_2 . Then, a *copy* instruction is dispatched to cluster n . This instruction will forward the value of the physical register in cluster n to the physical register in cluster m . This *copy* instruction will be handled by the issue logic as any other instruction, i.e., it will be executed once its source operand and the needed resources are available.

Note that during the renaming of an instruction, just one physical register for its destination register is allocated. Additional physical registers to store copies of it in other clusters are only allocated on demand if they are required by subsequent instructions that do not execute in the same cluster. All these physical registers will be freed by the first subsequent instruction that writes to the same logical register, when it is committed (see instruction I_3 , in Figure 1(c)). This scheme requires some degree of register replication which dynamically adapts to the program requirements and is much lower than replicating the whole register file. Compared with a full replication scheme, it has also less communication requirements and thus, less inter-cluster bypass paths and less register file write ports.

If the processor has a limited number of inter-cluster bypass paths, they must be reserved by the issue mechanism like any other resource. Copy instructions provide a simple mechanism to allocate the required bypasses and schedule inter-cluster communications. They also provide a simple method for precise state recovery, since copy instructions are inserted in the reorder buffer like normal instructions. However, since a copy instruction makes the dependence chain one node longer, it increases by one cycle the total effective latency between the producer and the remote dependent instruction (in addition to the bus latency). A particular implementation could optimize this, either by shortening the tags propagation delay between clusters or by implementing specific hardware that avoids generating copy instructions. However, we have not assumed any of these optimizations in this work.

2.2. Value prediction

The microarchitecture implements a stride value predictor [8, 9, 19] that predicts the source operands of the instructions. There is a value prediction table indexed by the PC and the operand order (left/right). We first assume a very large table (128K entries) to isolate the results from the effects of a limited table size, and we later evaluate the impact of a table with sizes ranging from 1K to 16K entries. Each entry contains the last value, the last observed stride and a 2-bit counter that assigns confidence to the prediction. Since each prediction involves a table access and an

addition, we assume that value predictions are available 1 cycle after the fetch, i.e. at the decode stage. Table updates are done at decode time.

When a source operand is not yet available at decode time, and its predicted value is confident (the counter value is greater than 1), the instruction is dispatched speculatively and may use the predicted value. The instruction that will produce this value is identified, and it is assigned the task of verifying that its output matches the prediction. The verification occurs during the writeback stage of the producer instruction, and it takes one cycle. If it fails, the dependent misspeculated instruction is invalidated and reissued.

We have assumed a selective invalidation and reissue mechanism [17], i.e. after the mispredicted instruction is reissued and executed, a new value is produced and propagated to dependent instructions, which in turn reissue, and so on. Only the instructions that depend on the mispredicted instruction are invalidated. The mechanism is in fact the existing issue mechanism, and therefore we have assumed no additional penalty for each instruction restart.

For a clustered architecture, this speculation procedure is further extended, in order to reduce inter-cluster communications. The extension apply to the case when a source operand is not currently mapped on the cluster where the instruction is being dispatched. In this case, the operand is predicted regardless of whether it is available, the instruction is dispatched speculatively, and a special *verification-copy* instruction is dispatched to the cluster where the operand is produced. When issued, the verification-copy compares locally the operand with the predicted value, and just in case of mismatch, it forwards the correct value through an inter-cluster bypass, and the remote misspeculated instruction is reissued.

2.3. Steering logic

Code partitioning can be done at compile time (static) or at run time (dynamic). The first method relies on the compiler, which allocates each static instruction to a cluster, while the second method is based on a specialized hardware that decides where to distribute each dynamic instruction. The main advantage of a static partitioning is that it requires minimal hardware support, but its downside is that it requires to recompile the applications because it extends the ISA for encoding the steering information. Furthermore, it will require to recompile for each new microarchitecture generation that changes the number of clusters.

In contrast, a dynamic partitioning method does not require to recompile, because it makes clustering transparent to the compiler. In addition, the information used by the dynamic steering logic (workload balance, data dependences) is obtained directly from the actual pipeline state, rather than estimations of the compiler. Therefore, a dynamic

steering scheme is more effective than a static approach because it is more adaptable to the actual processor state. This work focuses on this type of steering.

In order to maximize performance, the dynamic steering logic must address two main goals: to minimize inter-cluster communications (or their associated penalties) and to maximize the workload balance.

On one hand, inter-cluster communications introduce delays between dependent instructions, which may result in a performance loss if they stay in the critical path of execution. Determining whether a communication is critical is a hard problem, therefore a more simple goal for the steering heuristic is to minimize the number of communications.

On the other hand, when there are more ready instructions in a cluster than functional units to execute them, the excess of instructions are forced to wait, incurring an additional delay. If at the same time, another cluster has idle functional units, this additional delay would have been avoided if the steering logic had sent some instructions to a different cluster. We refer to this situation as a workload imbalance among clusters, and since it may potentially degrade the performance, a major goal of the steering logic is to prevent it from happening.

Intuitively, both goals (reducing communications and balancing workload) are sometimes conflicting, and therefore a good steering algorithm must find the optimal trade-off between them. We outline below how these two issues are addressed by the steering logic from a conceptual standpoint. Particular steering techniques are defined in Section 3

2.3.1. Communication. The valid bit associated to each field of the map table indicates whether the logical register may be directly read in the corresponding cluster without requiring a communication. Therefore, the steering logic uses this information to minimize communications by choosing a cluster where all or most of the source operands of an instruction are currently mapped. In some cases, when an operand is mapped in more than one cluster due to previously dispatched copy instructions, but the value is not yet available, the choice of clusters should be narrowed to the cluster where the value will be available sooner, to avoid the instruction being needlessly delayed by a communication.

2.3.2. Workload balance. To improve the workload balance, the steering logic must detect when there is a workload imbalance and how much unbalanced it is, and must also determine which is the least loaded cluster. There are many alternatives to determine at run-time the individual workloads of the clusters and their relative workload imbalance. In other words, there are several figures that can be used to measure these features. From the description

given above, we intuitively define the workload imbalance at a given instant of time as the total number of ready instructions that cannot issue, due to having exceeded the issue width in their respective clusters, but could have issued in other clusters since they have idle functional units. This figure (we will refer to it as metric NREADY), is what we report in our experiments as “workload imbalance”, because it corresponds to our definition. However, we also experimented several other imbalance figures to guide the steering decisions, and found the following scheme (we will refer to it as metric DCOUNT) to give the best performance:

- The processor has a signed counter in each of the N clusters that measures its workload. Its value is initially zero, and it is updated in the following way: for every instruction dispatched to a cluster, the corresponding counter in that cluster is increased by $N-1$, while the other $N-1$ counters are decreased by 1 (i.e. the sum of the counters is kept always zero). Therefore, the value stored in the counter of a given cluster is N times the difference between the total number of instructions dispatched to that cluster and the average number of instructions dispatched per cluster. The workload imbalance is calculated as the maximum absolute value of the workload counters. Note also that in the case of two clusters, a single counter will suffice.

The NREADY figure matches more exactly our definition of workload balance. However, when it is used by the steering logic, the actions taken to compensate a workload imbalance (sending instructions to the least loaded cluster) may not update immediately the NREADY figure, if some of the steered instructions are not ready. When this occurs, the corrective action may result disproportionate, and cause an imbalance in another direction or some unnecessary inter-cluster communications. This does not happen with the DCOUNT figure, since it varies instantly and in proportion to the steering decisions, which allows the steering logic to gauge more accurately the actions to compensate a workload imbalance. Thus, the steering logic uses the DCOUNT figure to determine balancing actions and we use the NREADY figure to measure and report workload balance.

2.4. Experimental framework

We perform our microarchitectural timing simulations with a modified version of the SimpleScalar tool set [3], version 3.0. It was extended to include register renaming through a physical register file, instruction queues (separate integer and FP), stride value prediction, steering logic, and a clustered processor core.

Three different configurations were simulated, having 1, 2 and 4 clusters respectively. Each was simulated with and without value prediction. The total issue width, number

Table 1. Main architecture parameters, for configurations with 1, 2 and 4 clusters

Parameter	1 Cluster config.	2 Clusters config.	4 Clusters config.
Fetch, decode & retire width	8 instructions		
Branch Predictor	Combined predictor of 1K entries with a Gshare with 64K 2-bit counters, 16 bit global history, and a bimodal predictor of 2K entries with 2-bit counters.		
ROB size	128		
Instruction queue size	64	32	16
Functional units	8 int (4 include mul/div) 4 fp (2 include fp mul/div)	4 int (2 include mul/div) 2 fp (include fp mul/div)	2 int (1 include mul/div) 1 fp (includes fp mul/div)
Issue width	8 int/ 4 fp	4 int/ 2 fp	2 int/ 1 fp
	Out-of-order issue. Loads may execute when prior store addresses are known		
Communications	1-cycle latency. Communications consume issue width and instruction queue entries		
Register file sizes	128	80	56
I-cache L1	64KB, 2-way set-associative. 32 byte lines, 1 cycle hit time, 6 cycle miss penalty		
D-cache L1	64KB, 2-way set-associative. 32 byte lines, 1 cycle hit time, 6 cycle miss penalty, 3 R/W ports		
I/D-cache L2	256 KB, 4-way set associative, 64 byte lines, 6 cycle hit time.		
Memory	8 bytes bus bandwidth to main memory, 18 cycles first chunk, 2 cycles interchunk.		

Table 2. The Mediabench benchmark suite¹

program	input	instr. count (millions)	description
cjpeg	testimg.ppm	18.8	image
djpeg	testimg.jpg	6.0	image
epicdec	test_image.pgm.E	11.1	image
epicenc	test_image.pgm	70.6	image
g721dec	clinton.g721	421.1	audio
g721enc	clinton.pcm	440.6	audio
ghostscript	tiger.ps	899.5	PS interpreter
gsmdec	clinton.pcm.gsm	115.1	audio
gsmenc	clinton.pcm	307.1	audio
mesamipmap	m.ppm	75.2	3D graphics
mesaosdemo	o.ppm	29.7	3D graphics
mesatextgen	t.ppm	129.4	3D graphics
mpeg2dec	test.m2v	12.6	video
mpeg2enc	test.par	222.0	video
pgpdec	pgptest.pgp	108.6	encryption
pgpenc	pgptest.plain	130.6	encryption
rasta	ex5_c1.wav	26.4	audio
rawcaudio	clinton.pcm	8.7	audio
rawdaudio	clinton.adpcm	7.1	audio

(1) Many program names are renamed for convenience

of functional units, and reorder buffer length was kept constant along the configurations while the sizes of register files and instruction queues were scaled down with the number of clusters. The most relevant architecture parameters are detailed in Table 1.

We simulate all Mediabench benchmark programs [14, 12], (except for Pegwit, which could not be compiled). We chose this benchmark suite because they are representative of modern multimedia applications, which is a growing segment of the commercial workloads. In addition, they exhibit a quite high ILP, which makes them suitable for testing a wide issue superscalar architecture like the one we present here. Table 2 lists briefly program input data and

run lengths. All the benchmarks were compiled for the Alpha AXP using Compaq’s C compiler with the -O4 optimization level, and they were run till completion.

We define a new metric to evaluate the performance of a clustered configuration relative to that of a centralized one, with similar characteristics: the normalized N-clusters IPC Ratio ($IPCR_N$ for short) is the quotient $IPC_{N-clusters}/IPC_{1-cluster}$. It indicates the IPC degradation caused by inter-cluster communication delays on a clustered architecture, and its maximum value is 1. This metric is useful to evaluate the impact of a particular technique (e.g. value prediction) on a clustered architecture, by comparing the IPCR obtained with and without implementing the technique. An IPCR increase would indicate that the technique produces higher IPC improvements in the clustered architecture than in the centralized one, thus measuring the benefits that are exclusive to the clustered architecture, isolated from other more general improvements that affect both configurations.

3. A steering scheme for value prediction

In this section, we first introduce a steering scheme that is very effective but does not include any technique to leverage value prediction. Then we present a steering mechanism that exploits value prediction as a way to reduce communication requirements.

3.1. The baseline steering algorithm

We have evaluated several steering strategies described in previous works [4, 5], and variations of them. Finally, the best performance was obtained with an enhanced version of the “Advanced RMBS” heuristic [4], generalized for an arbitrary number of clusters, which will be the Baseline scheme considered in this paper. This algorithm

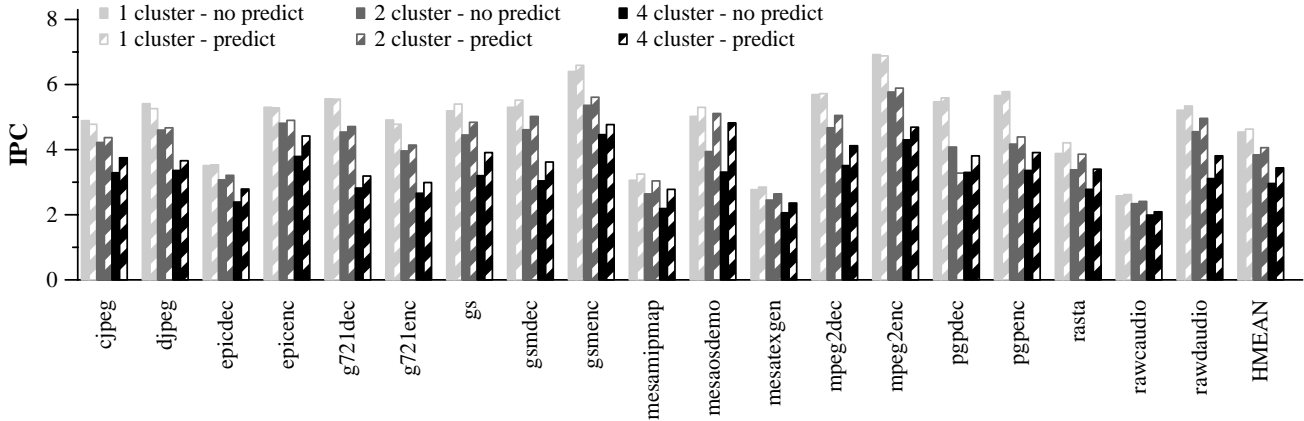


Figure 2. IPC of a 1, 2 and 4-cluster configurations (baseline steering), with value prediction, and without it

applies the criteria discussed above in the following way: in most cases, as a primary rule, it gives the highest priority to the reduction of communication penalties, and as a second rule, it tries to improve the workload balance. However, in some cases, when the workload imbalance is considered too high, the balance criterion takes precedence. The algorithm is described next, in more detail.

1. If the workload imbalance is higher than a given threshold, the current instruction is sent to the least loaded cluster.
2. Else, the clusters that will cause minimum communication penalties are identified:
 - 2.1. If any source operand is not available at dispatch time, select the cluster(s) where the pending operand(s) are to be produced.
 - 2.2. If all source operands are available, select the clusters that have the greatest number of operands currently mapped.
 - 2.3. If it has no source operands, select all clusters.
3. Finally, choose the least loaded cluster among those selected in step 2.

The threshold mentioned in rule 1 was set experimentally to $DCOUNT=32$ and $DCOUNT=16$ on a 4-cluster and a 2-cluster configurations, respectively.

Figure 2 shows the IPC obtained with the baseline steering algorithm for 1, 2 and 4 clusters, with value prediction and without it. The IPCs are higher when value prediction is implemented, although the improvement is rather low for the centralized configuration (2% on average, and negative for several benchmarks). The benefits are higher for the clustered organization (5% and 16% for the 2 and 4-cluster configurations respectively).

The two leftmost bars in each group in Figure 3 depict other interesting figures from the same previous experiments. Graph c shows the IPCR ratio increase provided by value prediction, which is a performance improvement spe-

cific to each clustered architecture, as discussed in Section 2.4. This graph shows a notable increase of the IPCR ratios when value prediction is implemented (in spite of a slight increase in the average workload imbalance, see graph a) which is due to a drastic communication reduction (graph b), especially for the 4-cluster configuration, where communications are also higher: $IPCR_4$ increases by 14%, from 0.65 to 0.74, and the communications rate is reduced by 44%, from 0.22 to 0.12.

3.2. Enhancing the partitioning scheme through value prediction

In this paper we focus on how value prediction may improve the performance of the steering logic in a clustered processor. We propose some modifications of the Baseline steering heuristic, based on the assumption that the predicted source operands will never cause communications or delays, and thus the steering may concentrate on improving the workload balance. The assumption is true if the prediction does not fail, but it may not hold otherwise. However, as far as the misprediction rate is kept low, these modifications may improve significantly the workload balance. The first two modifications to the steering strategy are described below, in more detail:

First, when the source operand of an instruction is predicted and it is not yet available, the steering algorithm considers it as available. By doing so, the algorithm does not force to steer the instruction to the cluster where the operand is going to be produced (if it is the only operand, rule 2.1 is not applied).

The second modification consists on considering any predicted source operand to be mapped in all clusters because, regardless of the cluster it is sent to, it will not cause any additional inter-cluster communication (unless the prediction fails and the operand is remote). In consequence, communication issues do not impose any restriction on the

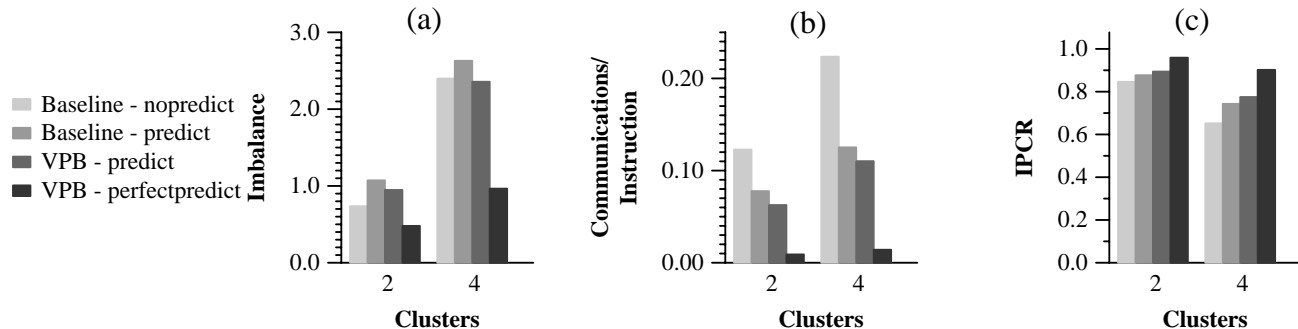


Figure 3. Comparison of 4 configurations: Baseline without and with prediction, VPB with prediction and VPB with perfect prediction. (a) Workload Imbalance (b) Communications/Instruction (c) Normalized IPCR

choice of clusters (i.e. this operand does not constrain the set of candidate clusters, if rule 2.2 is applied).

In summary, these two modifications to the baseline steering algorithm eliminate in some cases the constraints imposed by communications/delays issues (in rule 2), so that the algorithm has better opportunities for balancing the workload (since rule 3 selects one cluster from a wider choice of clusters).

We evaluated the impact of these two modifications on a 4-cluster configuration, and found that they produce a negligible average performance improvement over the baseline scheme. The average workload balance is reduced by 31% and, since imbalance correction actions (which ignore communication issues) are less frequent, one would expect also to have less communications. However, the communications ratio (which mostly determines the IPC) remains constant because there is also a communications increase due to an indiscriminate use of the optimistic initial assumptions. More specifically, if an instruction that uses a predicted source operand is sent to a cluster where it is not mapped, and the prediction fails, then this instruction will be re-issued non-speculatively, and a communication will be required to read the correct operand from a remote cluster.

3.3. The VPB steering scheme

In consequence, to minimize the above mentioned communications increase, the second modification to the Baseline steering scheme should only apply to those cases in which there is a potential for improving the workload balance. In particular, we propose that the steering logic considers predicted source operands to be mapped in all clusters only when the workload imbalance is higher than a given threshold (that we set empirically to $DCOUNT=16$ and $DCOUNT=8$, for a 4-cluster and a 2-cluster configurations, respectively). In other words, if the workload is very well balanced, the steering does not rely on value prediction to improve workload balance, since it may increase the

communication requirements. We refer to this technique as the Value Prediction Based scheme (VPB).

Figure 3 compares workload imbalance, communication rate and IPCR for 4 different configurations: the Baseline without and with value prediction, the VPB scheme, and VPB with perfect prediction. Comparing the results for a 4-cluster configuration with value prediction, the VPB scheme has 12% less communications than the Baseline and a 10% lower workload imbalance, which results in a significant performance improvement ($IPCR_4$ increases from 0.74 to 0.77).

The rightmost bar in each group in Figure 3 show an upper bound for the VPB scheme, assuming a perfect predictor. Communications are not zero because of fp values, that are not considered by our predictor. IPCR ratios are 0.90 and 0.96 for a 4- and a 2-cluster configurations respectively, which suggests that the performance of the VPB scheme may significantly be improved by a more effective predictor.

So far, all the reported experiments assumed that the rename/steering logic takes a single cycle. However, due to the additional complexity introduced by the steering logic, it might require 2 cycles, for some particular technology. We simulated a 2-cycle rename/steer stage and obtained that, for a 4-cluster configuration with VPB, the IPC is degraded by less than 2%.

In summary, we observe that value prediction produces significant performance improvements for a cluster organization, which are higher than those observed for a centralized one, especially when adequate steering techniques are implemented. In particular, we have found that for a 4-cluster configuration the $IPCR_4$ ratio increases on average by 18%, from 0.65 to 0.77, and for a 2-cluster configuration $IPCR_2$ increases by 5%, from 0.85 to 0.89. This is due to the drastic 50% reduction of the communication rate (from 0.22 to 0.11 for 4 clusters, and from 0.12 to 0.06 for 2 clusters). We can thus conclude that value prediction is a very effective technique to reduce the communication requirements of clustered processors.

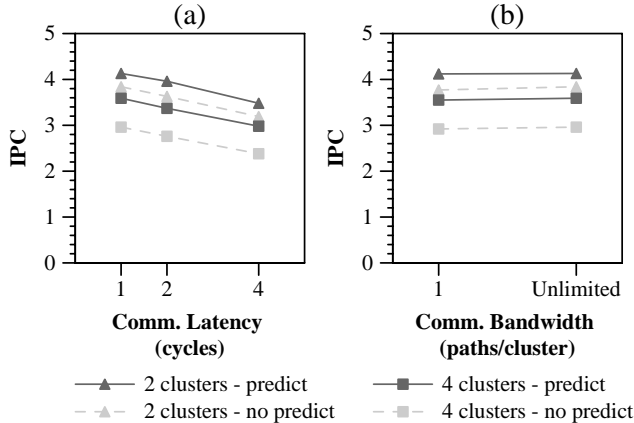


Figure 4. Impact of (a) communication latency and (b) communication bandwidth, on the IPC

The overall benefits of value prediction translate into an increase in IPC of 21% on average (from 2.96 to 3.59) for a 4-cluster architecture, and a smaller increase for a 2-cluster configuration (8%, from 3.84 to 4.14), whereas for a centralized processor the benefits are almost negligible (2%, from 4.54 to 4.63). Note that we assumed a simple value predictor and the results will likely be better with more complex and effective predictors

4. Sensitivity analysis

In future technologies the widening gap between the relative speeds of gates and wires will decrease dramatically the percentage of on-chip transistors that a signal can travel in a single clock cycle [1]. Using high clock rates will require not only to reduce the capacity of many components like register files and issue windows, but also to pipeline more deeply the access to other structures.

In this work we focus on the inter-cluster communication bypasses. In the previous sections we have assumed that these communications take 1 cycle (there is a 1 cycle “bubble” between the copy instruction and the dependent instruction, in another cluster). In this section, we study the sensitivity of clustered architectures to the communication latency, measured by the IPC degradation caused by a communication latency of 1, 2, and 4 cycles. In all cases we assume that communications are fully pipelined, that is, for a given bypass path, one communication may begin per cycle regardless of its total latency. We also analyze the impact of the communication bandwidth and value predictor table size on the performance of the processor.

4.1. Communication latency

Figure 4(a) shows that there is a significant performance degradation when the communication latency in-

creases from 1 to 4 cycles. For instance, on a 4-cluster configuration, the IPC decreases by 17% (and by 20% without prediction, because of its higher communication requirements). Similar trends are observed on a 2-cluster configuration, although the performance degradation is slightly smaller (16% with prediction and 17% without prediction).

4.2. Communication bandwidth

The inter-cluster communication bandwidth has a direct impact on the complexity and delay of the register files [7, 16] and the bypass network, since it determines the number of register file write ports devoted to remote accesses, the number of bypass multiplexer’s inputs coming from remote clusters, and the number of outputs from the bypass network to the interconnection network. Furthermore, the inter-cluster communication bandwidth also determines the number of tags that are broadcast to the instruction queues of remote clusters. Therefore, it has a direct impact on the complexity and delay of the wake-up logic, which depends quadratically on the total number of tags crossing its CAM cells [15, 16].

So far, we have assumed an unbounded bandwidth for the interconnection network to isolate our results from possible communication bandwidth bottlenecks. Here we study the impact of having a limited bandwidth. For an N-cluster configuration, we assume a simplified model with $N \times B$ independent paths. Each path is implemented through a pipelined bus where any cluster can send a value and each bus is connected to the write port of a single cluster register file. Therefore, we assume that each register file has B write ports for inter-cluster communications. Any cluster may allocate one of these paths to write a value to a remote register file, and holds it during a single cycle, since the communication is fully pipelined. Obviously, this model is somewhat idealized, since it omits the complexities due to the pipelining, arbitration, or variable latencies dependent on the topology, but it may provide a first order approach to evaluate the problem.

Figure 4 (b) shows that when the communication bandwidth is limited to a single path per cluster there is very little performance degradation compared to the unbounded model. For instance, on a 4-cluster configuration, the IPC decreases only by 1% (1.4% without value prediction), and a small IPC decrement is also observed for a 2-cluster configuration (0.2% and 1.8% respectively). In consequence, for inter-cluster communications in a cost-effective architecture, it may suffice just a single write port in each register file, a single incoming tag per issue window, and a single remote bypass attached to the input multiplexers of the functional units.

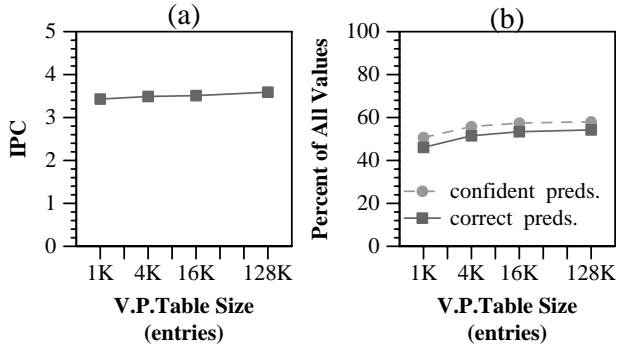


Figure 5. Impact of value predictor table size for 4 clusters on (a) IPC (b) predictor accuracy.

4.3. Value predictor table size

The predictor table size determines the prediction accuracy, which has a significant influence on the performance. We have evaluated the impact of the predictor table size on a clustered architecture. Figure 5(a) shows that on average, for a 4-cluster configuration, there is less than 4.5% IPC degradation when the predictor table size is reduced from 128K to just 1K entries.

Figure 5(b) shows the predictor accuracy for the same range of predictor sizes. We can observe that for 42% of the values, the predicted value was not used because it was not confident. The percentage of non-confident predictions is a bit high because we chose a rather simple value predictor. In addition, the hit ratio (correctly predicted values over predicted values) decreases from 93.4% to 90.9% when the predictor size is reduced from 128K to 1K.

5. Related work

The main contribution of this work is realizing that value prediction can eliminate many of the long wire communication penalties in the context of a clustered architecture. We have also presented a new steering algorithm, the VPB scheme, that takes advantage of value prediction to further reduce inter-cluster communications. Moreover, this paper extends the techniques used in previous works [4, 5] (for register renaming, dynamic steering of instructions and forwarding values among different clusters) from a configuration with 2 heterogeneous clusters to a more general design with an arbitrary number of homogeneous clusters.

Other relevant works on dynamically scheduled clustered processors are the Dependence-based, the Multicluster and the Pews architectures. In the Dependence-based paradigm [15, 16], instructions are steered to several FIFO queues instead of a conventional issue window, according

to a heuristic that ensures that two dependent instructions are only queued in the same FIFO if there is no other instruction in between. This heuristic lacks of any explicit mechanism to balance the workload, which is instead adjusted implicitly by the allocation algorithm of new free FIFO queues. This allocation algorithm generates many communications when it assigns a FIFO to a non-ready instruction, since it does not consider in which cluster the operands are to be produced [5].

The Multicluster architecture [6] also used run-time generated copy instructions for inter-cluster communication. In that architecture the register name space is partitioned into two subsets, and program partitioning is done at compile time without any ISA modification, by the appropriate logical register assignment for the result of each instruction. Both the workload balance and inter-cluster communication are estimated at compile time. The same authors proposed a dynamic scheme [7] that adjusts run-time excess workload by re-mapping logical registers. However, they found most heuristics to be little effective since the re-mapping introduces communication overheads that offset almost any balance benefit.

Kemp and Franklin proposed the Pews clustered architecture [11] where instructions are assigned to clusters based on register dependences. However, since they assume a centralized register file, the steering scheme only needs to group two dependent instructions in the same cluster when the value from the producer is not still available at the time the consumer is decoded. This simple scheme is not suitable for our “distributed” register file, and in addition, it does not address the load balancing problem.

The Alpha 21264 [10] is also a 2-cluster organization that duplicates the integer register file, one copy in each cluster. The two register file copies are kept consistent by writing any result in both clusters. Instructions are dynamically steered at issue time by a central instruction queue, that sends an instruction to the cluster where its operands will be available earlier. This organization does not reduce the number of register write ports nor the number of registers per cluster. Besides, it does not reduce the complexity of the issue logic, although it requires a simpler partitioning scheme.

The Trace Processors [17, 20] dynamically partition the code sequence into chunks of consecutive instructions, called traces. Instruction steering to clusters is then performed at run-time in a per-trace basis. This partitioning may result in an acceptable workload balance since traces have similar sizes but it is likely to result in many inter-cluster communications since they are not taken into account by the partitioning scheme.

Sastry, Palacharla and Smith proposed a static code partitioning technique [18]. The partitioning scheme is constrained to dispatching loads, stores and complex inte-

ger instructions to the same cluster. In addition, it requires some extensions to the ISA in order to specify to the hardware the target cluster of each instruction. Moreover, their scheme is less flexible and less effective than a dynamic approach as shown elsewhere [4], since all dynamic instances of the same static instruction are executed in the same cluster regardless of run-time conditions, such as the workload balance, that are difficult to estimate at compile time.

6. Conclusions

Future microprocessors are likely to be communication bound due to the increasing penalty of wire delays. In this paper we show that value prediction can be an effective instrument to improve communication locality. In particular, we have presented an approach to reduce inter-cluster communication by means of a dynamic steering logic that leverages value prediction. Values produced in a cluster and consumed in another one may not require long wire delays to propagate from the producer to the consumer if the consumer can correctly predict the value. The validation required by the prediction is locally performed in the producer cluster.

We have shown that value prediction removes communications even for previously proposed steering schemes not specially designed to exploit value prediction. However, performance is higher if the steering logic exploits the predictability of values. We have presented a novel steering scheme (VPB), and we have shown that it outperforms previous proposals. This benefit mainly comes from a 50% reduction in the amount of communications. We observed that value prediction reduces the penalties caused by inter-cluster communications by 18% on average. Moreover, whereas value prediction increases the IPC of a centralized architecture by just 2%, the same predictor increases the performance of a 4-cluster microarchitecture with VPB steering by 21%.

Acknowledgements

We thank the anonymous referees for their valuable comments. This work was developed using the resources of the CEPBA, and is supported by the Ministry of Education of Spain under contract CYCIT TIC98-0511.

References

[1] V. Agarwal, M.S. Hrishikesh, S.W. Keckler and D. Burger. "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures", in *Proc. of the 27th Annual Int. Symp. on Comp. Architecture*, June 2000.

[2] Bohr, Mark T. "Interconnect Scaling - The Real Limiter to High Performance ULSI". in *Proc. of the 1995 IEEE Int. Electron Devices Meeting*, pp. 241-244, 1995.

[3] D. Burger, T.M. Austin, S. Bennett. "Evaluating Future Microprocessors: The SimpleScalar Tool Set", *Tech. Report CS-TR-96-1308*, Univ. Wisconsin-Madison, 1996.

[4] R. Canal, J-M. Parcerisa, A. González. "A Cost-Effective Clustered Architecture". In *Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques (PACT 99)*, Newport Beach, CA, pp. 160-168, Oct. 1999.

[5] R. Canal, J-M. Parcerisa, A. González. "Dynamic Cluster Assignment Mechanisms". In *Proc. of the 6th. Int. Symp. on High-Performance Computer Architecture*, pp.132-142, Jan. 2000.

[6] K.I. Farkas, P. Chow, N.P. Jouppi, Z. Vranesic. "The Multi-cluster Architecture: Reducing Cycle Time Through Partitioning", in *Proc of the 30th. Ann. Symp. on Microarchitecture*, pp. 149-159, December 1997.

[7] K.I. Farkas. "Memory-system Design Considerations for Dynamically-scheduled Microprocessors", *Ph.D. thesis*, Department of Electrical and Computer Engineering, Univ. of Toronto, Canada, January 1997.

[8] F. Gabbay and A. Mendelson. "Speculative Execution Based on Value Prediction", TR. #1080, Technion, 1996.

[9] J. González and A. González. "Memory Address Prediction for Data Speculation". *Tech. Report UPC-DAC-1996-50*, Univ. Politècnica de Catalunya, Spain. 1996.

[10] L. Gwennap. "Digital 21264 Sets New Standard", *Microprocessor Report*, 10 (14), Oct. 1996.

[11] G.A. Kemp, M. Franklin, "PEWs: A Decentralized Dynamic Scheduler for ILP Processing", in *Proc. of Int. Conf. on Parallel Processing*, pp. 239-246, August 1996.

[12] C. Lee, M. Potkonjak and W. H. Mangione-Smith, "Media-bench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems", *Proc. of the Int. Symp. on Microarchitecture (Micro 30)*, pp. 330-335, Dec. 1997.

[13] D. Matzke, "Will Physical Scalability Sabotage Performance Gains", *IEEE Computer 30(9)*: 37-39, Sept. 1997.

[14] Mediabench Home Page. URL: <http://www.cs.ucla.edu/~leec/mediabench/>

[15] S. Palacharla, N.P. Jouppi, and J.E. Smith, "Complexity-Effective Superscalar Processors" in *Proc of the 24th. Int. Symp. on Comp. Architecture*, pp. 1-13, June 1997.

[16] S. Palacharla. "Complexity-Effective Superscalar Processors". Ph.D. thesis, Univ. of Wisconsin-Madison, 1998.

[17] E. Rotenberg, Q. Jacobson, Y. Sazeides and J.E. Smith, "Trace Processors", in *Proc of the 30th. Ann. Symp. on Microarchitecture*, pp. 138-148, December 1997.

[18] S.S. Sastry, S. Palacharla and J.E. Smith, "Exploiting Idle Floating-Point Resources For Integer Execution", in *Proc. of the Int. Conf. on Programming Lang. Design and Implementation*, pp. 118-129, June 1998.

[19] Y. Sazeides, S. Vassiliadis, J.E. Smith. "The Performance Potential of Data Dependence Speculation & Collapsing", *Proc. of Int. Symp. on Microarchitecture*, pp. 238-247, 1996.

[20] S. Vajapeyam and T. Mitra, "Improving Superscalar Instruction Dispatch and Issue by Exploiting Dynamic Code Sequences", in *Proc. of the Int. Symp. on Computer Architecture*, pp. 1-12, June 1997.