An Asymmetric Clustered Processor based on Value Content

R. González, A. Cristal, M. Pericas and M. Valero
Universitat Politècnica de Catalunya
{gonzalez,adrian,mpericas,mateo}@ac.upc.edu

A. Veidenbaum
University of California, Irvine
alexv@ics.uci.edu

ABSTRACT

This paper proposes a new organization for clustered processors. Such processors have many advantages, including improved implementability and scalability, reduced power, and, potentially, faster clock speed. Difficulties lie in assigning instructions to clusters (steering) so as to minimize the effect of inter-cluster communication latency. The *asymmetric* clustered architecture proposed in this paper aims to increase the IPC and reduce power consumption by using two different types of integer clusters and a new steering algorithm. One type is a standard, 64b integer cluster, while the other is a very narrow, 20b cluster. The narrow cluster runs at twice the clock rate of the standard cluster.

A new instruction steering mechanism is proposed to increase the use of the fast, narrow cluster as well as to minimize inter-cluster communication. Steering is performed by a history-based predictor, which is shown to be 98% accurate.

The proposed architecture is shown to have a higher average IPC than its un-clustered equivalent for a four-wide issue processor, something that has never been achieved by previously proposed clustered organizations. Overall, a 3% increase in average IPC over an un-clustered design and a 8% over a symmetric cluster with dependence based steering are achieved for a 2-cycle inter-cluster communication latency.

Part of the reason for higher IPC is the ability of the new architecture to execute most of the address computations as narrow, fast operations. The new architecture exploits its early knowledge of partial address values to achieve a 0-cycle address translation for 90% of all address computations, further improving performance.

Categories and Subject Descriptors

C.1.1 [**Single Data Stream Architecture**]: RISC/CISC, VLIW architectures

General Terms

Performance, Design.

Keywords

Cluster Architectures. Content aware architectures.

ICS\'05, June 20-22, Boston, MA, USA.

1. INTRODUCTION

The scalability of high-performance, out-of-order processor design is made very difficult by the increasing clock frequencies and issue width. It has been shown in [24], [6] that the wakeup/select and the ALU bypass loops as well as the register file are some of the main sources of difficulties in such designs. The issue width of four or more instructions requires a large number of ports on instruction queues and register file as well as bypassing to all the execution units present. This slows down the operation and/or requires multiple pipeline stages to accomplish.

Clustering has been proposed as a solution to these problems. It has been successfully implemented in several processors, such as Digital's Alpha 21264 [19]. In a cluster, the instruction queue access, wakeup/select, register file access (except for inter-cluster update), and ALU bypass are limited to just the units in the cluster. Instructions are typically assigned to clusters after decoding and are placed into per-cluster instruction queues. A data communication path has to be provided between clusters, either via the register file or the bypass network. Arrival of operands from another cluster(s) activates the wakeup logic. The communication between clusters takes one or more clock cycles.

One of the key problems in clustered architectures is the dynamic assignment of instructions to clusters (steering) to minimize inter-cluster communication. At the same time, the steering mechanism has to achieve a balanced cluster utilization so as not to waste resources and lose IPC. A clustered organization has lower IPC than the same organization but implemented without clustering, if it were feasible. The work presented here proposes a way to speed up execution by using a fast cluster and to minimize the effect of inter-cluster communication latency by a different approach to steering.

A number of research groups has recently shown that many integer operand values are either "small" or contain many all-0 or all-1 bytes [5][37][2][8]. This is particularly true for 64-bit architectures, in which approximately 50% of all integer instruction can be executed in a 16b ALU [5]. It has also been shown that, in addition, a large number of 64b address values have a common, invariant high-order part [14]. The processor in [14] had the register file and the data path optimized to take advantage of this value "content" locality. This increased the number of instructions using the narrow ALU to significantly above 50% of all integer instructions that was achieved in [5]. It was shown that 75% or more of all instructions could be executed using the narrow registers and data path.

It was also pointed out in [14] that operand types used by an instruction were "clustered" so that 75% of all instructions only needed narrow values, while 17% needed only wide, 64b values.

This, it was suggested, can be used to build a clustered processor with narrow and wide clusters.

This paper presents such a clustered processor organization to take full advantage of a very high probability that an instruction needs only a narrow data path. The goal of this design is to increase performance of a clustered architecture. It uses a fast, narrow cluster and a slower standard, 64b-wide cluster. There several ways one could design and optimize the narrow cluster. This paper proposes a 20b narrow cluster running at twice the clock rate of the rest of the processor. It therefore needs only half of the ALUs needed in the standard cluster. The 20b data path is chosen to increase the number of instructions using the narrow cluster. It can be changed to affect the instruction distribution.

Consider a 4-wide issue baseline processor with two integer clusters. Each cluster contains two 64b ALUs, a separate register file, and is dual issue. The clusters share the front end and the data cache. Now imagine that one of the clusters is implemented using a 20b data path. This cluster will only execute instructions with 20b inputs and produce a 20b result. There are several reasons it is argued below, why such a narrow cluster can be twice as fast as the 64b cluster in wakeup/select, register file access, execution, and write-back stages:

1. The narrow cluster will need only one ALU to achieve the same performance, and thus will need half the ports on the instruction queue, register file, and bypass network
2. Its register file is narrow, speeding up its access and making it physically smaller.
3. Its ALU is 1/3 or less of the size of the 64b ALU and does not have to perform long, slow operations such as multiply. The addition is the slowest remaining operation and can be performed in 1/2 the time of a 64b add if one assumes a staggered 64b adder, as in Pentium 4 [15].
4. The bypass path is 1/3rd as narrow and six times shorter than in the 64b cluster with 2 ALUs.
5. The wakeup/select logic is faster because it is single-issue and uses a single-ported instruction queue.

Let us assume that such an implementation is indeed feasible and consider the asymmetric organization. The asymmetric cluster organization requires a new steering mechanism to take full advantage of the fast cluster. Such a mechanism using past execution history is proposed and shown to be very effective and fast. The history-based cluster predictor is PC indexed and records the most recent cluster assignment for a given instruction. Furthermore, it can start the prediction as early as fetch stage and thus has several cycles to complete. The "best" steering algorithm in literature, dependence-based steering [9], is more complex and is performed after renaming where it has to complete in one cycle.

Instructions using the narrow cluster for the most part use results produced in the narrow cluster. Thus steering based on this information is a simplified form of dependence-driven steering and minimizes inter-cluster communication. The new steering mechanism does not aim to achieve balanced instruction distribution, instead it maximizes the use of the fast cluster. The steering mechanism is shown to be competitive even for symmetric clustered organization.

The history-based prediction can be "corrected" after renaming for ready operands whose value type (size) is known at that time. A value type descriptor is associated with every register. Steering stage (performed after renaming) can check available source operand value type information to see if the prediction was correct. For instance, a narrow cluster prediction is wrong if a source operand is wide.

The asymmetric cluster organization with prediction-based steering will have "mis-predictions". This happens when an instruction assigned to the narrow cluster uses a 64b source operand or produces a 64b result. The register descriptor is then set to reflect the latter. The narrow cluster assignment is the only type of mis-prediction possible in this asymmetric organization, a wide cluster assignment cannot be wrong since any instruction can execute in the wide cluster.

A mis-predicted instruction is sent to the wide cluster for replay, but first it wakes up its dependents in the narrow cluster. They will issue and detect their own source operand problems, if any, and replay. Wakeup of dependent instructions in the other cluster occurs when a tag sent with the data to be written in the replicated register file arrives. This guarantees data is available locally when instruction issues. There is no need for instruction window cleanup and re-execution from mis-predicted instruction as on branch mis-prediction. There is a mis-prediction time penalty, however, in the narrow cluster.

A mis-predicted instruction replays from the scheduling step, as it has already (correctly) gone through the wake-up. A separate scheduler port is used for this and replay has highest priority. The instruction is not inserted in the instruction queue of the wide cluster, rather payload RAM information read in the narrow cluster is sent to the wide cluster. The replay proceeds to read operands in the wide cluster and execute, thus incurring further latency in addition to the inter-cluster communication latency.

It is possible that an instruction assigned to the wide cluster produces a narrow result. It is in our best interest to detect this and update the predictor or inform the steering mechanism so it can assign successor instructions to the fast cluster.

Both the narrow and wide clusters may need operands from the other cluster. A replicated register file is used for the sake of control simplicity which incurs a communication penalty on write-back. As explained below, only the narrow part of the register file is replicated in the narrow cluster (plus a small additional register file).

The asymmetric architecture proposed in this paper treats address calculations in load/store instructions in a special way that allows most of them to be performed in the narrow cluster. This is possible because the high-order part of an address is often unmodified by the addition of a 16b offset in address computation. A special, small register file is used to store the high-order address bits. These registers are called address registers (Addr), each holding the invariant, frequently used upper address bits.

The use of invariant Addr registers opens the possibility of using them for address translation. A direct-mapped TLB using Addr registers is proposed as a level-0 TLB, followed by a "normal" (level-1) TLB. The level-1 TLB is accessed when a Ld/St instruction is not using one of the Addr registers. The level-0 TLB is indexed by an Addr register number and stores a page (or a super-page) translation corresponding to the Addr register. The translation is performed assuming that the upper part of an

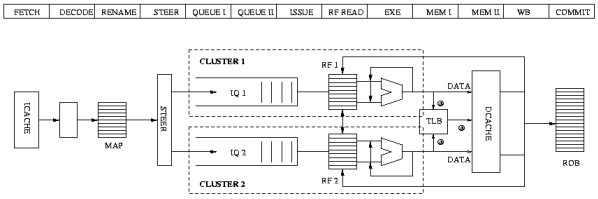

Figure 1. Symmetric Cluster Organization and Pipeline

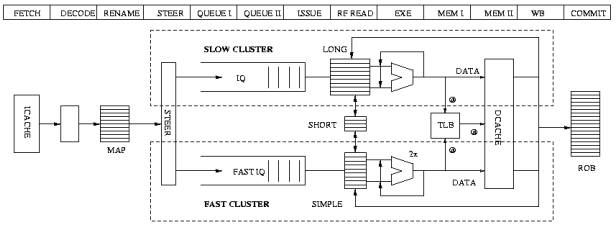

Figure 2. Asymmetric Cluster Organization and Pipeline

address stored in the Addr register remains unchanged after (base register + offset) computation. Translation time is thus shorter and is performed in parallel with address computation.

This paper makes several important contributions. First, it proposes an asymmetric clustered organization using a fast, narrow datapath cluster. Second, it introduces a fast and very accurate prediction-based steering algorithm. It also uses an asymmetric register file design, which enables a 0-latency address translation for a very high percentage of memory accesses. Overall, the proposed asymmetric architecture increases the utilization of the fast cluster. This significantly improves performance, which exceeds that of an un-clustered architecture, even with two (or even more) cycles of inter-cluster communication latency as well as two (or more) cycles of steering mis-prediction penalty.

The rest of this paper is organized as follows. Section II defines a standard, symmetric clustered architecture used as a baseline for performance comparison. Section III defines value content-based asymmetric clustering and its implementation used to evaluate performance, and discusses potential benefits of a shorter access time and area savings of narrow clusters. Sections IV presents a new steering mechanism and compares it with several existing algorithms on symmetric clustered architectures. Section IV also describes and analyzes the impact of fast narrow cluster design on performance. Related work is discussed in Section V. Finally, future work and conclusions are discussed in Section VI.

2. BASELINE ORGANIZATION

The symmetric 64b clustered processor organization used as the baseline is shown in Figure 1. It has two symmetric integer clusters with replicated 64b register files. Each write is sent to both register files and is posted to the remote register file one or more clock cycles later than to the local file. A write from the other cluster activates the local wakeup logic. Each cluster has a separate instruction queue and two ALUs for a combined issue width of four integer instructions.

The baseline pipeline is also shown in Figure 1. It has 13 stages to allow a fast clock. The front-end, e.g. stages up to and including QUEUE II, are shared by both clusters. Other parameters of the baseline processor organization are described in Table 1 and remain unchanged throughout the paper unless otherwise specified.

The baseline architecture requires a register file with 4 read and 2 write ports in each cluster with 2 ALUs. Two additional write ports are used for writes from the other cluster, plus 2 more write ports for memory access, for total of 4 read and 6 write ports.

Several different steering algorithms have been proposed for symmetric clustered architectures. They vary in the amount of inter-cluster communication and how "balanced" across clusters is the resulting execution. For example, the First-Fit algorithm [4] assigns consecutive instructions to the same cluster until its instruction queue (IQ) is full. It reduces the inter-cluster communication at the expense of balanced execution and good resource utilization. The MODn algorithm [4] assigns n consecutive instructions to the same cluster and then switches to the other cluster. Alpha 21264 used the MOD1 algorithm, while MOD3 has been shown to be one of the better algorithms in [4][3]. Instruction dependence based steering, DEPGRAPH, attempts to use data dependence knowledge in the assignment process to reduce communication [9]. It is a complex mechanism, so a modified version to reduce the hardware complexity is used in this paper. It is assumed in the rest of this paper that cluster assignment and instruction steering take only one cycle, a generous assumption for the DEPGRAPH algorithm.

The various algorithms differ in their complexity and the resulting IPC. The dependence based steering is the most complex to implement but results in the highest IPC among the proposed algorithms. Several steering algorithms will be evaluated on the baseline architecture assuming varying communication delay and compared to the new algorithm proposed in this paper.

3. ASYMMETRIC CLUSTERED PROCESSOR ORGANIZATION

The asymmetric clustered processor with two integer clusters, slow and fast, is shown in Figure 2. It has two ALUs in the 64b, slow cluster and one ALU with a 20b wide data path in the fast cluster. The floating-point cluster is not shown, as it is not changed in this architecture and is the same as in the baseline. Only Issue, RF READ, EXE, and WB stages run at the faster clock rate.

The key to understanding the asymmetric organization is the register file and the prediction-based steering mechanism, which are described next.

3.1 A Replicated Asymmetric Register File with a Content-aware Component

Figure 2 shows the details of each cluster's datapath and connections between them. Each cluster has a full-size (128 entry) physical register file, which is 20b wide in the narrow cluster. In addition, each cluster has an Addr register file (Short in the figure), which is not part of the ISA. It stores an upper part of a full memory address, with the low part stored in a narrow register. The Addr file can be as small as 8 entries.

The (replicated) physical register file in the narrow cluster is referred to as the Simple register file, while the register file in the wide cluster is called the Long register file. Each replicated physical register (Long or Simple) has a (replicated) 2-bit register descriptor, RD, associated with it. Possible value types recorded in RD are Simple, Long, or Addr. The replicated RD descriptor may need to be updated in both clusters on a write.

The use of RD allows us to determine what type of value is stored in a register (in each cluster). A physical register with RD type of Addr has an Addr register associated with it. n bits of a value (19 down to 19-n+1, where n is the size of Addr register), in a (Long or Simple) register entry are used as a pointer (PTR) to the corresponding Addr register.

The Long register file has four special features.

1. A write from the narrow cluster to this register file is performed with sign extension.
2. A 64b result in the wide cluster is checked to see if it is a 20b Simple value, i.e. if it has 20 or fewer significant bits. Only in this case is the result also written to the Simple register file in the narrow cluster.
3. Each result is checked to see if its value can use one of the valid Addr registers. In this case it is written to the Simple and the Addr register files in the narrow cluster and its type marked as Addr.
4. A Ld/St base address register value is checked to see if its high-order bits should be written to the replicated Addr file. This write is performed only from the wide cluster. Note that the base register is not updated in Ld/St operations.

The narrow cluster writes do not require any special treatment.

3.2 Addr value type and the Addr registers

Load/stores are very frequent and their treatment by this asymmetric architecture is one of its key features. The goal is to perform most Ld/St address computations in the fast cluster. This is accomplished by finding frequently used, invariant, 44b high-order bits in Ld/St base registers. These are stored in Addr registers while the low-order 20 bits go into a Simple register. The address is also still stored as a 64b value in the Long register (all registers are replicated in both clusters), but the register descriptor is changed to Addr type in this case.

An address computation adds a 16b immediate offset to the Simple part, which in the vast majority of cases does not result in a carry out. This means that the Addr part remains invariant. Also, an Addr part is very often the same for addresses produced by the same or by different Ld/St instructions. This allows the Addr register file to be as small as 8 entries and still achieve near-optimal performance. One can thus think of Addr registers as a cache for invariant partial addresses. In fact, each Addr (file) entry has a valid bit and a value.

Determining whether to save the 44b from the Ld/St base register in an Addr register is performed as following. First, a 64b base register in a Ld/St instruction is checked. If the upper part of this Long register matches a value in a corresponding valid Addr register (as determined by using the PTR bits), then the Long register descriptor RD is changed to Addr. A copy of the low-order part is also written to the Simple file so that it can be used in the fast cluster for future instances of this Ld/St or this memory "page". The steering predictor is updated to send this Ld/St to the fast cluster in the future.

However, if the corresponding Addr register was not valid (with one exception, see below) then it is written with the upper part of the Long register and marked valid. This is how Addr registers become valid and the only way they are written. The Long register descriptor is also marked Addr in this case, the update is propagated to the Simple file and the predictor is updated.

A replacement policy for a valid Addr entry is needed to make sure that only frequently used addresses are kept. The replacement can occur in two ways. A use bit keeps track of whether the entry has been used since it was written. A valid entry that has not been used is replaced by a write. In addition, a "background" algorithm frees entries that have not been used for a while. An entry that is marked used but which has not been accessed for a period proportional to 2 times the number of entries in the ROB is freed (see [14] for details).

Note that the 20b address arithmetic in the fast cluster can result in an overflow. The overflow, if detected, indicates that the Addr register can no longer be used for this Simple register and is a case of a mis-predicted cluster assignment. The RD descriptor and the predictor are updated to reflect this.

Based on the above discussion, the replicated Addr register file has 2 read and 2 write ports in the wide cluster and 1 read and 1 write port in the fast cluster.

Finally, when a register of type Addr is used by a Ld/St in the fast cluster, the corresponding Addr register is read in the Execute stage, after the PTR bits to index it become available. The good news is that the (invariant) Addr register value is not used in address computation. The low-order 20b of the address are computed using only the Simple register. By the end of the execute stage the narrow ALU produces a result and the Addr register value is read out. They are concatenated and the complete, 64b address is sent to memory (except when the 20b ALU overflow is detected).

3.3 Cluster Predictor and Instruction Steering

There are several possible predictor design choices, which affect prediction accuracy. A PC-indexed, history-based predictor was chosen after an extensive study (the details of which are not presented here due to lack of space). For each decoded instruction a cluster prediction is made based on previous execution of the same instruction. The predictor is an array of 1b cells indexed by the instruction PC and thus can be accessed very early in the pipeline. This history based cluster predictor was found more accurate than using individual predictors for each instruction operand's value type.

Other predictor parameters were chosen as following:

1. A tag-less design is used, as it gave more accuracy
2. The initial value points to the wide cluster to avoid mis-prediction
3. The predictor is updated in write-back

The effect of predictor size on prediction accuracy is discussed below.

One thing to keep in mind is that steering to the wide cluster has the lowest cost in terms of mis-prediction penalty but carries a performance penalty since clusters have different speed.

The register descriptor (RD) information is available in the rename stage for source registers that have their value and which have not been remapped. The steering prediction has already been made by this time, but is re-checked using a source operand RD value, if the source operand is ready. The prediction-based steering decision is "corrected" as following: the assignment is changed to the wide cluster if any source operand is long.

3.4 Instruction Pipeline

This sub-section describes new actions in the instruction pipeline necessary to implement asymmetric clustering. Figure 1 showed the 13-stage baseline instruction pipeline, this section describes what needs to be done in each stage for asymmetric clustering.

INSTRUCTION FETCH: The cluster predictor is accessed in this stage to hide its access latency.

DECODE/RENAME: assigns a Simple / Long register entry to every logical register. Re-checks steering prediction.

STEERING: the steering decision has been made by this time, so only the actual steering is performed

ISSUE: a register written from another cluster may cause wakeup. An instruction in the narrow cluster may be awaken by a write to a Long register. In this case the register tag and descriptor RD are sent from the wide to the narrow cluster, but not the value. This instruction will be replayed.

Re-play may be initiated in the wide cluster if a mis-predicted instruction arrives from the fast cluster. It has highest priority in the scheduler and issues as soon as possible.

RF READ: Type descriptor RD and the Addr register pointer PTR are read. A Long type read in the narrow cluster sets a mis-prediction flag and marks the instruction.

EXECUTE: The Addr register is read in this stage in parallel with ALU operation. 20b overflow in the narrow cluster sets a mis-prediction flag.

WRITE-BACK: writes are posted to the replicated register files, but write-back is different in narrow and wide clusters. Steering predictor update is performed.

1. **NARROW CLUSTER**: write if a mis-prediction flag was not set, otherwise prevent write-back and initiate a replay.
2. **WIDE CLUSTER**: write the Long destination register and send to the fast cluster for wakeup. Perform the following checks and initiate register descriptor update and fast cluster write, if needed

   a) Check if the result is a Simple value, i.e. the upper (64-20) bits are all 0 or all 1. The value is written to the narrow cluster if Simple. The time to do this is accounted for in the inter-cluster communication latency.

   b) A check for Addr type result is performed in parallel with Simple determination. The n-bit PTR field of the result (bits 19 downto 19-n+1) is used to read the Addr register file (this is fast since the Addr file is small). The result's upper 44 bits are compared to the (valid) value in Addr register. The result RD descriptor is set to type Addr if equal, or if the Addr register was not valid. The value is written to the narrow cluster if Addr.

   c) Memory load values are similarly checked prior to writing a register.

The additional delay to detect Simple or Addr values can potentially be hidden in the pipeline. For instance, cache data is typically available early (by 1 cycle in Alpha 21264) and can be checked. Only a comparator delay needs to be hidden for Simple value determination. The Addr register file access starts only in write-back before comparison can be performed. If this delay is a problem, value type can potentially be stored with the data in the L1 cache.

3.5 Steering Mis-prediction Recovery

Mis-prediction recovery in the narrow cluster is performed as follows. First, the instruction is replayed by sending it to the slow cluster. And second, steering predictor and the destination register are updated.

The replay is accomplished by sending the instruction's payload RAM content to the wide cluster. There a separate Ready port to the scheduler is used when the mis-predicted instruction arrives after inter-cluster communication latency. It has the highest priority. By this time any operands it may need have been replicated in the Long cluster (e.g. a narrow value written by a previous instruction) and they are read from the Long register file. The Execute and Write-back stages follow.

4. PERFORMANCE EVALUATION

This section presents performance evaluation of asymmetric

Table 1. Baseline clustered processor organization

Issue/Fetch/Commit width	4 instructions/cycle
Branch predictor	Combining ,16K entries
I-L1 size	32 KB, 4-way, 1 cycle
D-L1 size	32 KB, 4-way, 2 Rd/Wr ports, 3 cycle
L2 size	1MB, 4-way 10 cycle latency
Memory latency	100 cycles
Memory bus width	32 bytes
Clusters	2 Integer, 1 floating point
Integer Functional Units (lat.)	2 per cluster (latency 1)
FP Functional Units (lat.)	4 (latency 2)
Physical registers	128 Int. per cluster (4 Rd/6 Wr ports), 128 Floating Point
Reorder Buffer	128
Load/Store Queue	64
Integer Queues	20 in each cluster
FP Queue	64

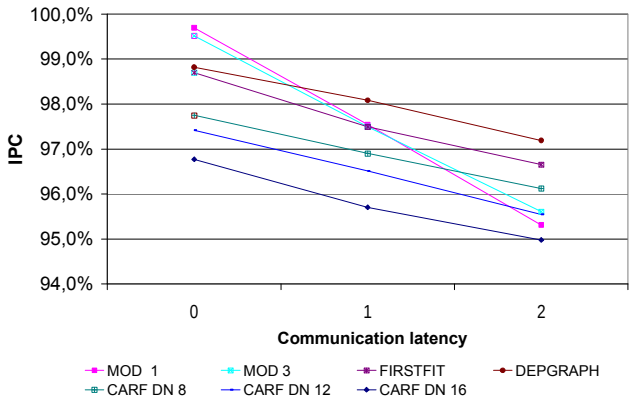

Figure3. Average IPC for SPEC2000 benchmarks relative to un-clustered organization

clusters and the impact of changes in the cluster organization with respect to a baseline architecture. The configuration of the baseline processor is shown in Table 1.

SPEC2000 benchmark suite was used to evaluate the performance of this approach. They were compiled for the Alpha ISA. The results presented are averages over both integer and floating point applications. All benchmarks have been simulated for 300 million representative instructions, where representative is defined following [31].

4.1 Steering

The IPC achieved with the new steering mechanism is first evaluated and compared to IPC for several existing steering algorithms. This is done using a symmetric clustered architecture for fair comparison. The goal is to show that the new algorithm can achieve good instruction distribution balance and reduce inter-cluster communication. The inter-cluster communication latency is varied from 0 to 2 cycles.

The algorithms used for comparison are MOD1, MOD3, FirstFit, and (simplified) Dependence Graph. All IPC results are normalized to those achieved by the un-clustered architecture and are shown in Figure 3 (averaged over all SPEC2000 benchmarks). The MOD1, MOD3 algorithms degrade rapidly after the communication latency increases beyond 1 cycle. The dependence graph and first fit algorithms perform better at higher latencies, with a maximum IPC loss of 4 to 5% compared to un-clustered results.

The new steering algorithm has an extra parameter that can be used to affect steering: the data path width W in the narrow cluster. If W were increased, more instructions would be steered to the narrow cluster. This changes the balance, but still keeps successors in the same (narrow) cluster. Results for three different values of W are shown in the figure: 8b, 12, and 16b. The 8 and 12b steering algorithms are 1 to 2% below the top competitors.

Data in Figure 3 is for symmetric clusters. The steering algorithm proposed in this paper is intended to improve performance of an asymmetric architecture. The goal there is different: steer as many as possible to execute in the fast cluster without increased communication and creating imbalance. In that setting, the effect of W is going to be different. A less competitive version in the symmetric cluster case, such as the 16b wide mechanism, will perform better than a well-balanced, 8b steering mechanism.

Figure 4 shows the instruction distribution balance actually achieved by the asymmetric clustered organization with a 20b datapath width. The results demonstrate that the new algorithm can steer most of the instructions to the fast cluster for improved performance, which is our goal.

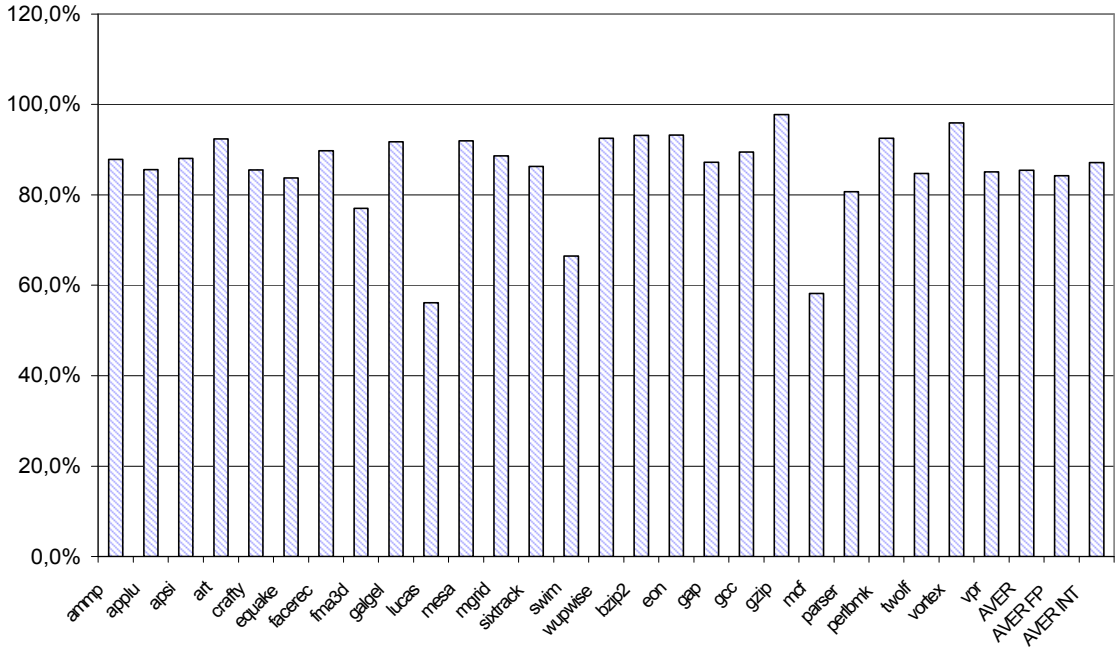

Figure 4. Instruction Distribution with a 20b asymmetric cluster (Narrow / (Narrow + Wide))

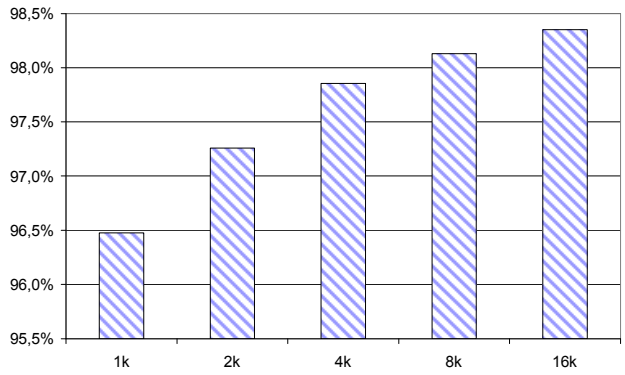

Figure 5. Steering Predictor Accuracy vs Predictor size

4.2 Cluster Prediction

This section presents the steering predictor evaluation. The steering prediction accuracy is shown in Figure 5. The results are presented averaged over the entire SPEC2000 suite. They are shown for different size predictors, from 1K to 16K entries. A prediction accuracy of 96% is achieved for a 1K-entry predictor. It increases to 98.5% for 16K entries. The difference between a 4K and 16K predictor accuracy is 0.5%, making a 4K predictor a good practical alternative. In all cases predictor output was "corrected" using source operand value types if they are available.

Two things should be noted about the predictor. First, the use of a large predictor is feasible in spite of additional delay because predictor access can be initiated early in the pipeline, as soon as the PC value is known. Second, not all mis-predictions have the same performance penalty. For instance, assigning a narrow ALU instruction to the wide cluster does not require re-execution in another cluster and thus does not incur a significant loss. Incorrectly assigning an instruction to the narrow cluster and then re-executing it in the wide cluster is costly.

The loss of accuracy with size decrease is due to two main reasons. First, aliasing increase in the tag-less predictor with size decrease, in particular on I-cache misses. And second, the predictor is updated after the write-back stage and thus during several cycles it may not be accurate.

4.3 Asymmetric Cluster Performance

Let us now examine asymmetric cluster performance focusing on IPC. In this section the steering mis-prediction penalty is set to 0 or 2 cycles, but the communication latency is always two (slow) cycles. The narrow cluster runs at 2x the clock speed of a wide cluster. Recall that instruction processing prior to the Issue stage is common to all instructions.

Fig. 6 presents the asymmetric cluster IPC results relative to the un-clustered architecture for prediction-based and dependence-graph based steering. Symmetric clustering with dependence based steering is shown for comparison. The 100% level corresponds to the un-clustered architecture. Cluster mis-prediction penalty is 0 cycles for both steering algorithms.

The results demonstrate that asymmetric clustering with a fast, 20b cluster (left bars) achieves a 3% IPC increase over un-clustered organization and an 8% increase over the best symmetric clustering (right bars) for integer programs. The increase is slightly smaller for f.p. codes. The increase over the un-realizable un-clustered architecture is quite remarkable and is due to the steering algorithm and the use of the fast narrow cluster.

Results for individual benchmark vary, with the majority demonstrating that asymmetric clustering performs better than the

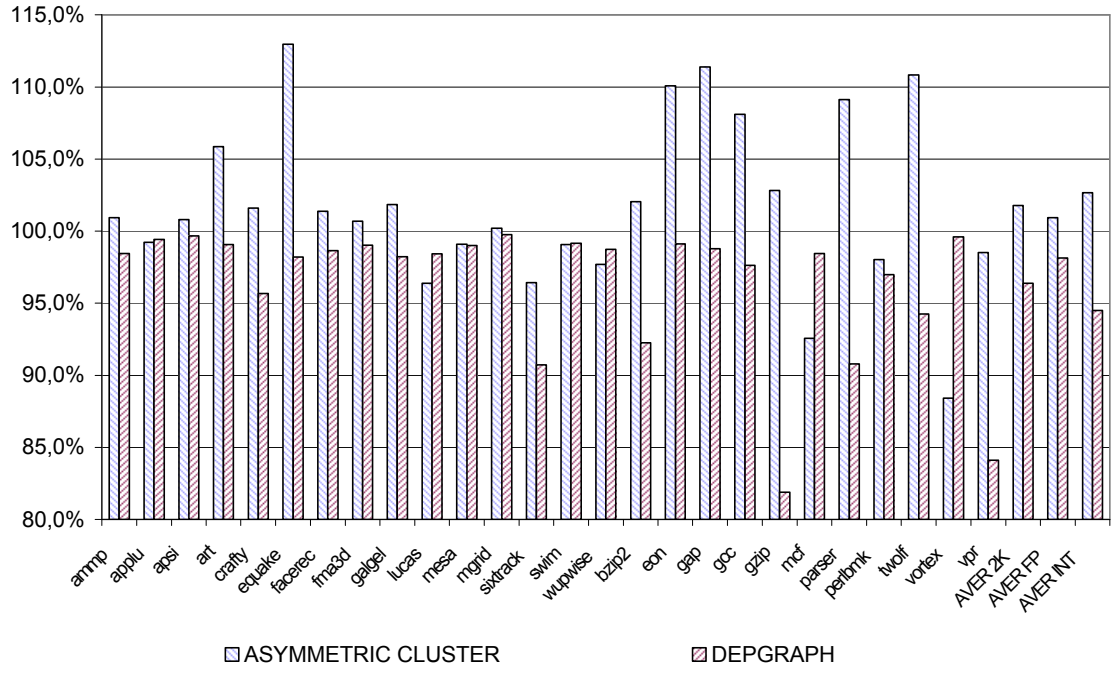

Figure 6. Relative IPC for asymmetric and symmetric clustered organizations (0-cycle mis-prediction and 2-cycle communication latencies)

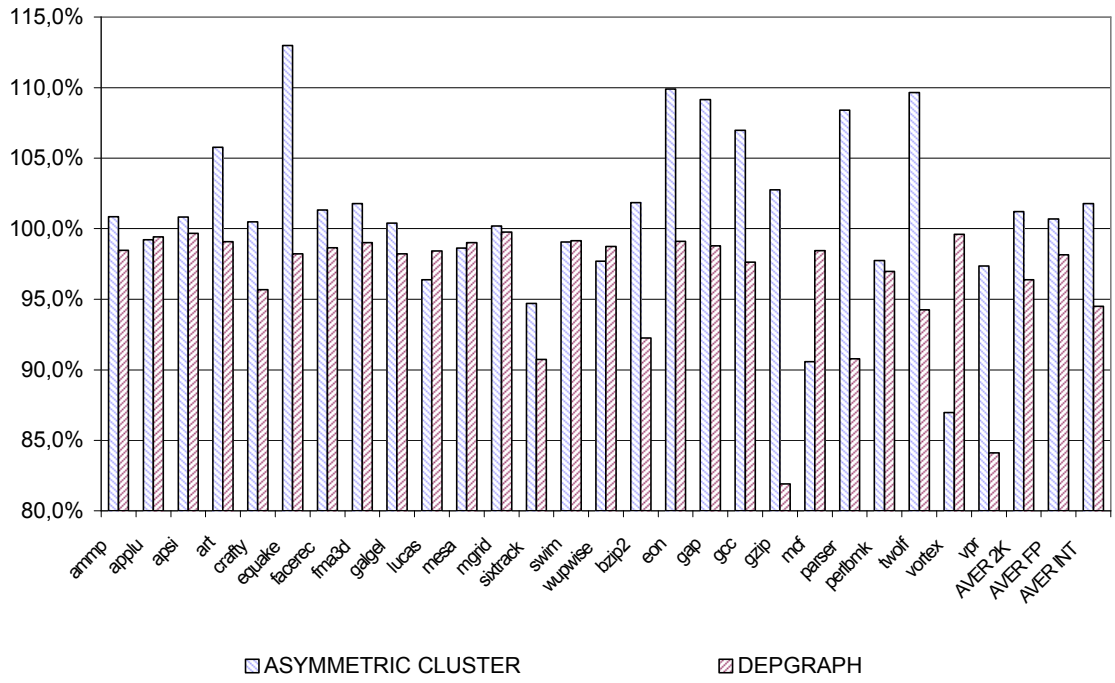

Figure 7. Relative IPC with mis-prediction and communication latency of 2 cycles.

symmetric clustering. Many exceed the un-clustered architecture performance (the 100%).

Figure 7 shows the IPC results for the mis-prediction penalty of 2 cycles. On average, the results are only slightly lower. As can be seen, many benchmarks still exceed the performance of un-clustered architecture because they execute most of the instructions in the double-speed fast cluster, including the majority of dependent instructions. This is something that does not happen on any symmetric clustered architecture,

One can possibly further improve asymmetric cluster performance by using criticality information. Fields et al [12] proposed criticality-based scheduling and steering, The former was applied to an un-clustered architecture and shown to improve its performance. Both were applied to clustered architectures and shown to exceed the performance of the un-clustered architecture, which used *"normal"* scheduling. Note that when both architectures used criticality-based scheduling the un-clustered architecture still performed better in [12].

4.4 0-delay TLB

This section examines the effect of using the Addr register file as a level-0 TLB. As each entry is added to this file, a copy of the page attributes and the physical address translation is stored in a parallel "TLB" structure. Let us assume here that the page size is chosen to be 32KB.

The new TLB is indexed by the Addr register "pointer" and is thus direct-mapped. As described above, the Addr register is accessed in the Execute stage, thus the same is true for the new TLB. As a result, the translation is available at the end of the Execute stage making it a 0-delay TLB.

In case of a miss in the level-0 TLB, a standard level-1 TLB is accessed. It can be larger than in the traditional architecture, if level-0 TLB can translate the majority of accesses.

The effect of this level-0 TLB can be seen in Fig. 8 . It presents results for 8-, 16- and 32-entry level-0 TLBs, showing the percentage of Ld/St instructions translated by this TLB. The results are presented for mis-prediction and communication latencies of 2 cycles. They demonstrate that approximately 90% of all translations are taken care by this new TLB.

5. RELATED WORK

Related work can be divided into several major groups: data-based optimization, critical structures, clustered architectures, and register file design and instruction scheduling.

Data based optimization. Brooks et al. [5] studied the data width of operands in a 64-bit architecture. They found that over 50% of operands could be executed with only 16 bits. As a result, it was shown that a 45% to 60% reduction in functional unit power consumption can be achieved. The power reduction estimate was based on a study by Nagendra et al [23], which explained in detail the impact of narrow width on power, area and timing.

Loh [21] proposed a Multi-Bit-Width micro-architecture, which attempts to accurately predict instruction data-width. It uses the width information to simultaneously execute multiple narrow-width instructions on a 64b datapath, increasing the effective issue width.

A more recent study by Gupta et al [39] showed that a small group of frequent values is used repeatedly in programs. (It was based on this idea that we originally developed a mechanism to get "proximate" frequent values in addresses which lead to the approach in [14])

The idea of fast and slow ALUs and schedulers was used in Pentium 4 [15], but without clustering. ALU selection was based on instruction type and only a very limited subset of instruction types was allowed to use the fast ALU. The fast ALU performed a 32b add in a pipelined (staggered) fashion allowing a successor instruction to be started as soon as the end of the first 16b operation. The operation of the fast cluster described in this paper is somewhat similar to the P4, except that it uses a 20b narrow and a 64b wide ALU.

Critical structures. Palacharla [24] identified critical structures present in processors and their effect on performance. Structures such as the register file, bypasses, and instructions queues are shown to be the main performance bottlenecks in processor architecture. A number of more recent studies have examined each of these in more detail. For instance, Sassone et al [28] studied the impact of multi-cycle bypass networks. [25][17][7][13][33] proposed ways to speedup instruction wakeup and scheduling or reduce its energy consumption.

Register file design. There are a number of interesting studies of register file design ([20][30][36][40][26][18][2][3][34][22][10][35]) aiming to reduce power or improve performance. The main difference of the approach in [14] and, in part of the work presented here, is the use of partial value locality, which allowed the register file to be partitioned into three separate structures. One of these structures only needed 8 registers and one was only 20b wide. This allowed a faster access and an energy reduction.

Clustered architectures. Palacharla et al [24] study the complexity and delay on structures for processors based on future technologies. They identify the main parts of processors that will have problems and more impact over throughput. Horowitz et al. [16] showed that design of present/future processors will have to adapt to higher communication costs by further partitioning some of the critical structures. And Aggarwal et al [1] studied the effects of high clock rates on IPC.

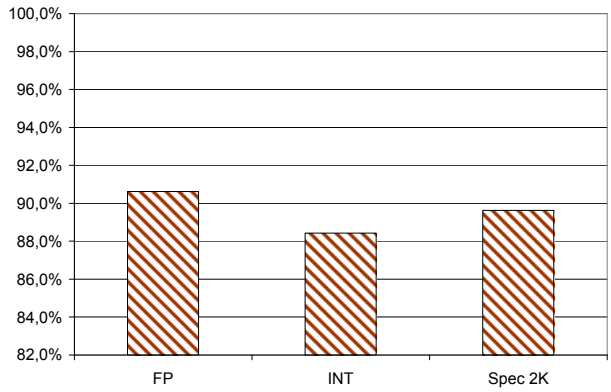

Figure 8. % Load/Stores Translated by the Level-0 TLB

Farkas et al proposed as a clustered architecture [11], which divides functional units, registers and instruction windows between two clusters. They used a static instruction scheduling to handle dependencies. Rotenberg et al. proposed Trace processors [27] which dynamically divide program instructions among multiple processing elements, each with private functional units. Sohi et al. proposed Multiscalar Processors [32] that divides a program into tasks guided by software, with each task executed on a small cluster (processing unit).

Baniasadi et al evaluated various queue clustering schemes for a single threaded processor [4]. Their work focused on queue assignment for “split” instruction queue. Balasubramonian et al investigated the impact of clustering a single-threaded processor into up to 16 clusters [3], and proposed a dynamic scheme for selectively disabling clusters in order to reduce communication costs. Aggarwal et al explored the use of different algorithms for assigning instructions to clusters [1]. They investigated structures most likely to affect critical timing path for wide issue processors. The impact of dividing the traditional out-of-order queue into a series of instruction FIFOs was presented.

Canal et al proposed several instruction queue clustering schemes [9]. The Alpha 21264 [19] had two clusters of functional units with replicated register files that were kept consistent by hardware.

Wakeup/Scheduling Weiss and Smith [38] and Sato [29] proposed a pointer-based instruction wakeup mechanism using a pointer to successor instructions. The pointer-based mechanism can be used for wakeup of dependent instructions in the fast cluster. Fields et al [12] proposed criticality-based scheduling, which they applied to clustered architectures to show that it can mitigate the effect of inter-cluster communication.

6. CONCLUSIONS

This paper introduced the idea of asymmetric clustering and presented an asymmetric clustered processor organization based on operation and operand value type. A 20b narrow cluster operating at twice the speed of a standard, 64b cluster was presented and shown to execute, on average, over 80% of all integer instructions. The result was a higher average performance than that of an un-clustered organization as well as that of previously proposed symmetric clusters using the best steering algorithms.

Prediction-based steering was introduced and shown to achieve up to 98% accuracy, depending on its size. The accuracy is not equal to performance loss, however, as mis-prediction may simply not produce a performance gain when an instruction is mistakenly sent to the slow cluster.

Fast cluster organization with a single, double-speed ALU allows the Instruction Queue and register file to have fewer ports and the bypass network to be very short justifying a speed increase, simplifying the hardware, and saving power. But other ways of organizing asymmetric clusters are also possible and may be advantageous under different conditions.

The last contribution of the paper is a 0-delay TLB implemented via the Addr register file. 90% of all memory access instructions are translated this way resulting in 1 to 2% additional average IPC

increase. This approach also saves power by bypassing a large TLB. Last but not least, this allows a physically addressed cache to be used given a 0-delay translation. Further research will investigate ways to improve the coverage of the 0-delay TLB.

7. ACKNOWLEDGMENTS

This work has been supported by the Ministry of Education of Spain under contract TIN-2004-07739-C02-01, the HiPEAC European Network of Excellence, and the Barcelona Supercomputing Center and in part by the National Science Foundation under Grant No. NSF CCR-0311738.

8. REFERENCES

[1] A. Aggarwal and M. Franklin. "*An empirical study of the scalability aspects of instruction distribution algorithms for clustered processors"*. In ISPASS, Nov. 2001.

[2] S. Balakrishnan and G. Sohi. "*Exploiting value locality in physical register files*". Research report, University of Wisconsin, 2002

[3] R. Balasubramonian, S. Dwarkadas, and D. Albonesi. "*Dynamically managing the communication-parallelism trade-off in future clustered processors*". In 30th ISCA, June 2003.

[4] A. Baniasadi and A. Moshovos. "*Instruction distribution heuristics for quad-cluster, dynamically-scheduled, superscalar processors*". In 33rd International Symposium on Microarchitecture, Dec. 2000.

[5] D. Brooks and M. Martonosi. "*Dynamically exploiting narrow width operands to improve processor power and performance*". In HPCA, pages 13–22, 1999.

[6] E. Borch, E. Tune, S. Manne and J. Emer . "*Loose Loops Sink Chips*" .Proceedings of the Eighth HPCA 2002

[7] A. Buyoktusunoglu, S. E. Shuster. D. Brooks, P. Bose, P. W. Cook, and D. H. Albonesi, "*An Adaptive Issue Queue for Reduced Power at High Performance*", Workshop on Power Aware Computer Systems, in conjunction with ASPLOS-IX, November 2000.

[8] R. Canal, A. González, and J.E. Smith. "*Very low energy pipelines using significance compression*". In 33rd MICRO, 2000.

[9] R. Canal, J.-M. Parcerisa, and A. Gonzalez. "*Dynamic cluster assignment mechanisms*". In Proceedings of the 6 HPCA, Jan. 2000.

[10] J. L. Cruz, A. González, M. Valero, and N. P. Topham. "*Multiple-banked register file architectures*". In 27th ISCA, 2000.

[11] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic. "*The multicluster architecture: Reducing cycle time through partitioning*". In 30th International Symposium on Microarchitecture,Dec. 1997.

[12] B. Fields and S. Rubin and Rastislav. "*Focusing processor policies via critical-path prediction*", Proceedings of the 28th annual international symposium on Computer architecture, 2001

[13] D. Folegnani and A. González, "*Energy Effective Issue Logic*", Proceedings of 28th ISCA, 2001. Göteborg Sweden.

[14] R. González, A. Cristal, D. Ortega, A. Veidenbaum and M. Valero. "*A Content Aware Integer Register File Organisation*". In 31th ISCA, June 2004

[15] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, P., "*The Microarchitecture of the Pentium 4 Processor*" Intel Technology Journal Q1, 2001.

[16] M. Horowitz, R. Ho, and K. Mai. "*The future of wires*". In Seminconductor Research Corporation Workshop on Interconnects for Systems on a Chip, May 1999.

[17] M. Huang, J. Renau and J. Torrellas, "*Energy-Efficient Hybrid Wakeup Logic*", Proceedings of ISLPED August 2002 Page(s): 196-201, Monterrey California, USA.

[18] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, and A. Yoaz. "*A novel renaming scheme to exploit value temporal locality through physical register reuse and unification*". In 31 MICRO, 1998

[19] R. Kessler. "*The alpha 21264 microprocessor*". In IEEE Micro,March/April 1999.

[20] M. H. Lipasti, B. R. Mestan2, and E. Gunadi1. "*Physical Register Inlining*". In 31th ISCA, June 2004.

[21] G. H. Loh. "*Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth*". In Proceedings 35th Intl. Symposium on Microarchitecture, Pages: 395 – 405, 2002

[22] M. Moudgill, K. Pingali, and S. Vassiliadis. "*Register renaming and dynamic speculation: an alternative approach*". In 26th MICRO.

[23] Nagendra, M. Irwin., and R. Owens, "*Area-time-power tradeoffs in parallel adders*". IEEE Trans. Circ. Syst. 43, 10, 689-702. 1996

[24] S. Palacharla, N. P. Jouppi, and J. E. Smith. "*Complexity effective superscalar processors*". In 24th ISCA, June 1997.

[25] M. A. Ramírez, A. Cristal, A. V. Veidenbaum ,L. Villa, M. Valero. "*A Simple Low-Energy Instruction Wakeup Mechanism*", (ISHPC-IV), Oct. 2003

[26] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J. Owens. "*Register organization for media processing*". In 6th HPCA, , Washington - Brussels - Tokyo, January 1999. IEEE.

[27] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. Smith. "*Trace processors*". In 30th MICRO, Dec. 1997.

[28] P. G. Sassone D. Scott Wills. "*Multicycle Broadcast Bypass: Too Readily Overlooked*". WCED ISCA 2004

[29] T. Sato et al, "*Revisiting Direct* Tag *Search Algorithm on Superscalar Processors*", Workshop on Complexity-Effective Design (in conj. with ISCA), 2001.

[30] A. Seznec, E. Toullec, and O. Rochecouste. "*Register write specialization register read specialization: a path to complexity-effective wide-issue superscalar processors*". In MICRO 2002.

[31] T. Sherwood, E. Perelman, and B. Calder. "*Basic block distribution analysis to find periodic behavior and simulation points in applications*". In Proceedings of the Intl. Conference on Parallel Architectures and Compilation Techniques pages 3–14, Sept. 2001

[32] G. Sohi, S. Breach, and T. Vijaykumar. "*Multiscalar processors*".In 22nd Annual International Symposium on Computer Architecture, June 1995.

[33] J. Stark and M. D. Brown and Y. N. Patt, "*On pipelining dynamic instruction scheduling logic*", International Symposium on Microarchitecture, pp. 57-66, Dec. 2000.

[34] N. Sung Kim, T. Mudge. "*Reducing Register Ports Using Delayed Write-Back Queues And Operand Pre-Fetch*". In of 17th Conference on Supercomputing. June 2003.

[35] J. H. Tseng, K. Asanovic. "*Banked multiported register files for high-frequency superscalar microprocessors*". In 30th intl. symposium on Computer architecture, June 2003.

[36] J. Tseng and K. Asanovic. "*Energy-efficient register access*". In 13th (SBCCI'00), September 2000.

[37] L. Villa, M. Zhang, and K. Asanovic. "*Dynamic zero compression for cache energy reduction*". In 33rd MICRO, 2000.

[38] S. Weiss, J. E. Smith, "*Instruction Issue Logic for Pipelined Supercomputers*", Proc.11th ISCA, pp.110-118, 1984.

[39] Y. Zhang, J. Yang, and R. Gupta. "*Frequent value locality and value centric data cache design*". In Proceedings of the 9th ASPLOS, pages 150-159. ACM Press, 2000.

[40] V. Zyuban and P. Kogge. "*The energy complexity of register files*". In Intl. Symposium On Low Power Electronics And Design, 1998.