ABSTRACT

Clustered microarchitectures are becoming a common organization due to their potential to reduce the penalties caused by wire delays and power consumption. Fully-distributed architectures are particularly effective to deal with these constraints, and besides they are very scalable. However, the distribution of the data cache memory poses a significant challenge and may be critical for performance. In this work, a distributed data cache VLIW architecture based on an interleaved cache organization along with cyclic scheduling techniques are proposed. Moreover, the use of Attraction Buffers for such an architecture is introduced. Attraction Buffers are a novel hardware mechanism to increase the percentage of local accesses. The idea is to allow the movement of some data towards the clusters that need it.

Performance results for 9 Mediabench benchmarks show that our scheduling techniques are able to hide the increased memory latency when accessing data mapped in a remote cluster. In addition, the local hit ratio is increased by 15% and stall time is reduced by 30% when using the same scheduling techniques with an interleaved cache clustered processor with Attraction Buffers. Finally, the proposed architecture is compared with a state-of-the-art distributed architecture such as the multiVLIW.

Results show that the performance of an interleaved cache clustered VLIW processor with Attraction Buffers is similar to that of the multiVLIW architecture, whereas the former has a lower hardware complexity.

Categories and Subject Descriptors: C.1.1 [Processor Architectures]: Single Data Stream Architectures - Pipeline processors, RISC/CISC, VLIW architectures.


Keywords: clustered microarchitectures, VLIW processors, distributed cache, Attraction Buffers, modulo scheduling.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1. INTRODUCTION

Nowadays processors are moving from capacity-bound to communication-bound due to the increasing impact of wire delays [23][1]. One approach to deal with this problem is to distribute some critical resources of the processor into semi-independent units while others remain centralized. Each one of these units is commonly referred to as a cluster. For example, a processor can consist of a number of clusters and each one can be made of a local register file and a set of local functional units while the issue engine and the data cache remain centralized. Local communications in a cluster are fast, while global communications among the clusters are slow. Clustering following this strategy has been used in the last years by superscalar processors (e.g., Compaq Alpha 21264 [11]), but this trend is even more noticeable in embedded/DSP VLIW processors (e.g., HP/ST’s Lx [8], TI’s TMS320Cox [28], Analog’s Tighershare [9] and Equator’s MAP1000 [22]).

Whereas the clustering of functional units and the register file is common in current commercial microprocessors, some recent works advocate for the clustering of other critical components such as the memory hierarchy [29][24][27]. In this work, we focus on this type of microarchitecture. In particular, we investigate alternative approaches to distribute the L1 data cache memory. We use instruction scheduling techniques for cyclic code (loops) to exploit the features of the proposed interleaved data cache organization. We demonstrate that it is feasible to partition the data cache among clusters and still have a competitive performance.

In addition, based on the observation that most remote accesses are to data that has been previously accessed by the same cluster, a hardware enhancement called Attraction Buffers is proposed and evaluated. These buffers allow a small portion of data to be replicated in all clusters, increasing the number of local accesses. Coherence is kept by the construction of what we call memory dependent chains. While the use of interleaved memories is not new, its use in a clustered VLIW processor, the proposed scheduling algorithms and the use of the Attraction Buffers are novel ideas in the field of static instruction scheduling.

Finally, an interleaved cache clustered VLIW architecture with Attraction Buffers is compared with a state-of-the-art clustered VLIW architecture such as the multiVLIW [27]. Statistics for
the Mediabench benchmark suite show that such an architecture offers almost the same performance as the multiVLIW with lower hardware complexity.

The rest of this paper is organized as follows. Section 2 describes other approaches that have been used for clustered VLIW processors. Section 3 presents the basic interleaved cache clustered VLIW architecture along with the use of Attraction Buffers. Next, in Section 4, the scheduling algorithms are described. Finally, Section 5 and Section 6 describe the performance evaluation and the main conclusions respectively.

2. RELATED WORK

Multiple-banked interleaved memories have been used for long as a low-complexity solution to implement “semi”-multi-ported memories [7][19]. The idea is that each memory bank or module can serve one (or more) request at a time and hence, if all memory requests happen to go to different banks, they can be served in parallel. This design has also been used in the memory system of vector processors with the goal of increasing the throughput and bandwidth of memory for strided accesses [16][13].

On the other hand, different proposals exist in the literature for clustered VLIW architectures with a unified cache [21][26][12][5]. We use for comparisons the organization used in [26], which is depicted on the left side of Figure 1. In such architecture a cluster consists of a local register file and a set of local functional units (FUs). All clusters share a L1 cache memory and this is why the cache is said to be unified. Cluster resources can be homogeneous among clusters or heterogeneous. Homogeneous clusters will be considered in this work for simplicity. Clusters communicate register values through a set of buses which are called register-to-register buses. Hence, the compiler is responsible for adding an explicit copy instruction whenever it schedules two register flow dependent instructions in different clusters. A copy instruction broadcasts the desired value and any cluster can forward this value to a local functional unit or store it in a local register.

The main drawback of a clustered architecture with a unified cache is that scalability is still restricted. If more ports are added to this cache in order to increase ILP with more memory functional units, access time, area and power consumption are affected. Moreover, some studies point out that access time (in number of cycles) to memory structures is likely to increase with future technologies, even when their capacity is kept constant [1]. This suggests that short latency memory structures should be even smaller than they are today. Because of these two reasons, we consider that a distributed cache memory architecture is key for performance of future microarchitectures.

A few proposals of clustered architectures with a distributed data cache can be found in the literature. In [27], Sánchez et al. proposed to divide the L1 cache among clusters. In such an architecture, a cluster consists of a local register file, a set of local functional units and a L1 cache module. The contents of all cache modules may differ from one another. Hence, some coherence mechanism must be employed to guarantee correctness. The authors used a snoopy MSI coherence protocol found in many commercial multiprocessor systems (for this reason they called this approach multi-VLIW processors). A diagram of a multiVLIW architecture is depicted on the right side of Figure 1. A multiVLIW architecture has the main advantage that data tends to move towards the clusters that make use of it. However, data replication may limit the effective capacity of the cache. In the best case, the effective capacity of all first level caches is C (the sum of all individual cache capacities) if all caches’ contents differ. On the other hand, in the worst case, the effective capacity is C/N (where N is the number of clusters) in case the contents are all the same. In addition, there is the added complexity of the cache coherence protocol.

The RAW architecture proposed in the MIT [2][29] has an organization similar to the architecture proposed in this paper. A RAW machine is composed of several identical units (which they call processor tiles) organized as a 2-dimensional mesh. Each tile has its own functional units, register file and both instruction and data memories. Tiles are interconnected through a network which provides static and dynamic interconnection. The former has the property that it is very fast and the latency can be determined at compile time. The latter is slower and has a non-constant latency, and it is used when the target of the communication is not known. In this architecture, as in the one proposed in this paper, the memory is distributed in an interleaved fashion. The three main differences

![Diagram of a unified cache clustered VLIW and a multiVLIW architectures.](image-url)
3. PROPOSED ARCHITECTURE

In this section the proposed architecture is explained, which is a cluster VLIW architecture with a partitioned cache. First, the basic design is shown, while in Section 3.2 the use of Attraction Buffers is introduced. These buffers will increase the percentage of local accesses.

3.1. Basic Architecture

In this paper we propose to distribute the L1 data cache among the clusters in an interleaved fashion. A diagram of such an architecture is shown in Figure 2, where only one memory bus is assumed. The register file and register-to-register communication buses are not pictured for clarity purposes.

In such an architecture, a cache block will be distributed among the different clusters and each line of a cache bank will hold some words of the block, depending on the interleaving factor. We will use the term subblock to identify the words of the same block that are mapped to the same cluster. The size of a subblock is B/N (where B is the size of a cache block and N is the number of clusters) and it may contain contiguous words depending on the interleaving factor. For example, given a 4 cluster architecture, a cache block of 8 words and an interleaving factor of one word, words 0 and 4 of the block will form subblock 1 and will be mapped into cluster 1. We distribute the cache in this fashion so that it will be easier for the scheduling algorithm to increase local accesses. We will use the terms cache module or cache bank to identify the local portion of the cache in one cluster.

A miss in the cache will bring the whole block from the next memory level to the cache modules. The block will be split into sub-blocks and each subblock will be stored in its appropriate cache module. In such a cache organization, each subblock will reside in only one cluster so there is no data replication at all. However, tags must be replicated in all caches so that the cache system has local identifiers for its contents.

In an interleaved cache clustered architecture, a memory access can be classified into four different types:

1) **local hit**: when the address of the access references the local cache module and the requested data is present in it. The access is satisfied with a local access latency.

2) **remote hit**: when the address of the access references a remote cache module and the requested data is present there. The latency of the access is the sum of sending the request over the memory bus, performing a cache access in the remote cache module and sending the reply back to the original cluster.

3) **local miss**: when the address of the access references the local cache module and the requested data is not present in it. In this case, the latency of the access is the sum of performing a cache access (to detect the miss), sending the request over the memory bus, performing a next memory level access, and sending the reply back through the memory bus from the next memory level to the cache.

4) **remote miss**: when the address of the access references a remote cache module and the requested data is not present there. This is the most expensive operation because it requires to send a request over the memory bus to the remote cache module, perform a cache access there and detect the miss, send the request to the next memory level, perform a next memory level access, send the reply over the memory bus from the next memory level to the remote cache module, and send the request back over the memory bus from the remote module to the local one.

As can be seen in Figure 2, memory requests issued by a functional unit enter the cache pipeline if the address is local or are put in the out bus buffer otherwise. Requests that exit the cache pipeline may directly feed the functional units in case of a local hit or may be sent to the out bus buffer in case of a remote hit reply, a local miss or a remote miss. In case of a local miss or a remote access, the request is inserted in the Miss Status Holding Register (MSHR) if
it is a load request, while in the case of a store, it is inserted in the MSHR only if it is a local miss.

Eventually, requests in the out bus buffers get a free bus and end up accessing a remote cache module or the next memory level. Again, replies from the next memory level or from remote cache modules are stored in the in bus buffers waiting to feed a functional unit (waking up some instructions in the MSHR) and/or entering the cache pipeline to write their contents in the local cache module.

Each cache module implements its own independent replacement policy so that no distributed coordinated action is needed. Thus, it may turn out that some subblocks of a given block may be present in the cache while others are missing. This implies that misses may bring data from the next memory level that is already present in some clusters. If the subblocks that are already present in a cache module have been modified, the new contents brought from the next memory level must be discarded. In addition, a block replacement from the L1 data cache does not consist of a single bus transaction but it consists of N individual bus transactions (where N is the number of clusters) that may happen at different times.

Regarding the memory bus system, note that each cluster does not need full read access to the whole bus, but only to the bus lines of its corresponding subblock. For instance, in Figure 2, only one memory bus is shown for clarity purposes, which has been divided into four lines because each cluster needs read access to a subblock of it. Hence, when interchanging data among clusters, the appropriate data lines must be used. However, each cluster needs write access to the whole bus but its bit lines in order to be able to send data to other clusters. The logic labeled MUX (multiplex) logic in Figure 2 is responsible to put data in the appropriate bit lines of the bus.

From now on, such an architecture will be referred to as a basic interleaved cache architecture.

3.2. Interleaved Caches with Attraction Buffers

One approach to reduce remote accesses is to use buffers in each cache module to hold some remote data. We will use the term Attraction Buffers to refer to them because data is attracted where needed. The idea is that when a remote access is performed, not only the requested word is sent to the requesting cluster, but the whole remote subblock is. The remote subblock is stored in the cluster’s Attraction Buffer since chances are high that it will be used in a near future by it (due to the fact that our scheduling algorithm tries to maximize memory instructions with a stride multiple of NxI, where N is the number of clusters and I is the interleaving factor, see Section 4). Subsequent accesses to that subblock will be satisfied locally by the Attraction Buffer. A remote access that brings a complete subblock only needs one bus transaction as before because each cluster has access to B/N bit lines of the bus. Thus, no additional bus traffic is added by this mechanism.

We assume that the local cache module and its associated attraction buffer form one hardware structure instead of being two separate ones. Thus, such structure contains two main blocks: the local data block, and the attraction buffer itself, as it can be seen in Figure 3. Both structures share the same access ports and since it is known beforehand whether an address is local to a cluster or not, the signal that feeds the data selector logic and the hit selector logic can be computed through the local logic combinational system pictured on the left hand. In the example of the figure, a subblock with words 2 and 6 is stored in the Attraction Buffer. Note that if \( \log_2 N \) bits are not enough to index a set in the Attraction Buffer, uncontiguous bits will be used as the index, since the field denoted by W determines the accessed word inside a subblock.

However, the use of Attraction Buffers has three main drawbacks. First, the access time of a cache module is increased because of the extra logic to derive the data and hit lines, as it can be seen in Figure 3. In addition, a memory operation always enters the local cache pipeline either if its address is local or not. Hence, the remote hit and remote miss latencies are increased (note that the local hit and local miss latencies are not modified).

Finally, the use of Attraction Buffers may generate multiple copies of the same data and some coherency mechanism must be used to guarantee correctness. We propose a low complexity solution due to the fact that given a loop, our scheduling algorithm schedules its memory dependent instructions in the same cluster (see Section 4). The solution is based on flushing the contents of these buffers between loops since coherency is guaranteed by the instruction scheduler within a loop but not between loops. Modified data must be updated in the original cache module when the Attraction Buffers are flushed or when some of their contents are replaced.
4. SCHEDULING ALGORITHMS

In this section the static instruction scheduling algorithms are explained. We have used three different algorithms: one for a clustered architecture with a unified cache, and two for an interleaved cache clustered architecture. All algorithms perform modulo scheduling on hyperblock loops. Due to space restrictions, the main points of the algorithm are discussed. The reader is referred to [10] for further details.

4.1. Background on Modulo Scheduling

Modulo scheduling is an effective technique to extract instruction-level parallelism (ILP) on loops by overlapping the execution of successive iterations of the original loop without unrolling it [14][15]. It is a well-understood technique used by many current compilers.

The parameters that most affect the performance of a modulo scheduled loop are the Initiation Interval (II), the Stage Count (SC) and the register pressure. The II is the number of cycles between the initiation of consecutive iterations. For loops with a high trip count, execution time is almost inversely proportional to the II. The Stage Count determines the number of overlapped iterations. Finally, register pressure can have an important effect on performance in those cases that the schedule requires more registers than are available. This may require the insertion of spill code or an increase on the II, which in both cases may reduce performance.

Another important factor in static scheduling techniques is the scheduling of memory operations. Memory operations have a variable latency which makes them more difficult to schedule. If they are scheduled too late, they may unnecessarily increase the register pressure, the II and/or the SC. On the other hand, if they are scheduled too early, they may cause pipeline stalls [25].

4.2. Algorithm for a Unified Cache Clustered Architecture

A diagram of the steps of the algorithm for a clustered architecture with a unified cache can be seen in Figure 4. The main goal of the algorithm is to end up with a compromise between balancing the workload and minimizing the number of register-to-register communications. Loop unrolling is performed because it helps the scheduling process [26] and profiling information is used to schedule memory instructions with the cache hit latency or the next memory level latency in order to reduce stall time. The algorithm is similar to the one proposed in [26].

The algorithm orders the nodes (instructions) of a given loop following the heuristic called Swing Modulo Scheduling [18]. We use such an heuristic because it has a good performance in terms of II values and register pressure [6]. Once the nodes are ordered, the algorithm schedules nodes one at a time. For each node, it computes the set of possible clusters where this node can be scheduled into based on resource usage. Then, this set is ordered so that clusters that minimize register-to-register communications and that balance the workload are selected first. Finally, the algorithm tries to schedule the node in all possible clusters by following the order computed previously. Memory nodes are scheduled with the cache hit or miss latency based on their hit rate, which is obtained through profiling. Note that no backtracking is used: whenever a node can not be scheduled, the value of the II is increased.

This algorithm is also the base algorithm for the other architectures with some changes, so we will call it BASE algorithm.

4.3. Algorithm for an Interleaved Cache Clustered Architecture

The next algorithm we have implemented is an algorithm targeted to an interleaved cache clustered architecture. Prior to scheduling, this algorithm unrolls the loop in order to increase the number of memory instructions with a stride multiple of N×I (where N is the number of clusters and I is the interleaving factor). Memory instructions with such a stride access the same cluster in all iterations once a loop is entered.

The heuristic to schedule non-memory instructions is the same as the one used for a unified architecture: try to schedule the instructions in the clusters where register-to-register communications are minimized and the workload is balanced. However, we use two different heuristics to schedule memory operations. The first heuristic, called IBC (Interleaved Build Chains, see Section 4.4), treats memory instructions like any other instruction: it schedules them in the node with the best trade-off between register-to-register communications and workload balance. The second heuristic, called IPBC (Interleaved Pre-Build Chains, see Section 4.4), schedules memory instructions in the cluster that they access most (we call it the preferred cluster). This information is gathered through profiling. Finally, it should be pointed out that memory instructions are scheduled with the local hit latency, the remote hit latency or the miss latency based on their hit rate.
latency based on their hit rate and their stride (for further details, refer to [10]).

4.4. Handling Memory Dependent Instructions

Care must be taken when scheduling memory dependent instructions in an interleaved cache clustered architecture because the latency of such instructions is unknown. For example, a load may read a stale value from memory if a previous dependent store scheduled in another cluster is still sending the updated value through the memory bus when the load is issued. Hence, a mechanism must be used to ensure program correctness. Our solution to this problem is conservative, but very easy to implement. Our scheduling algorithm for an interleaved cache clustered architecture schedules memory dependent operations in the same cluster. Thus, one step of the algorithm is to identify what we call memory dependent chains, mark them, and end up scheduling all nodes in the same memory dependent chain in the same cluster. We use the memory dependence analysis implemented in the IMPACT environment tool to perform memory disambiguation [4]. Note that when the compiler is not able to disambiguate memory references it always stays on the conservative side: it adds dependences between unresolved memory accesses. Thus, memory dependences in the Data Dependence Graph indicate true dependences and false unresolved dependences.

The scheduling algorithm using the IBC (Interleaved Build Chains) heuristic builds a memory dependent chain while it is about to schedule the first node of it. It then chooses the cluster where register-to-register communications are minimized for that node and marks all other nodes in the chain to be scheduled in the same cluster. On the other hand, the scheduling algorithm using the IPBC (Interleaved Pre-Build Chains) heuristic computes the memory dependent chains prior to scheduling and chooses the average preferred cluster for all nodes in the same memory dependent chain.

5. PERFORMANCE EVALUATION

In this section the evaluation methodology and the results are presented. First, the benchmarks and the configuration parameters are discussed. Next, results for the interleaved cache clustered architecture are shown. Finally, these results are compared with a state-of-the-art clustered processor like the multiVLIW, and the impact of the memory dependent chains is evaluated.

5.1. Tools and Configurations

The IMPACT compiler [3] has been used as the base infrastructure to compile the benchmarks, optimize them, and build hyperblocks [20]. The benchmarks we have used are a subset of the Mediabench suite [17] because they represent real workloads that can be found in media or embedded processors like DSPs. The benchmarks have been simulated completely.

<table>
<thead>
<tr>
<th>Table 1. Basic configuration parameters used for simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of clusters</td>
</tr>
<tr>
<td>Functional Units</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Cache parameters</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Interleaving factor for interleaved caches</td>
</tr>
<tr>
<td>Register-to-register communication buses</td>
</tr>
<tr>
<td>Memory buses</td>
</tr>
<tr>
<td>Next Memory Level parameters</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
The performance of three different clustered VLIW architectures has been evaluated: a cluster architecture with a unified cache, a cluster architecture with an interleaved cache, and a multiVLIW architecture. Each architecture consists of 4 clusters. The BASE algorithm has been used to modulo schedule hyperblock loops in a clustered architecture with a unified cache. Both heuristics (IBC and IPBC) of the algorithm presented in Section 4.3 have been used to modulo schedule the same loops for an interleaved cache clustered architecture and a multiVLIW architecture. Note that a multiVLIW architecture has also a distributed non-deterministic latency cache and we use the same solution we have used for an interleaved cache clustered processor to guarantee correctness: the construction of what we have called memory dependent chains. The IBC heuristic tends to minimize compute time since it tries to reduce register-to-register communications, while the IPBC heuristic tends to reduce stall time because memory instructions are scheduled in their preferred cluster. Since both heuristics generate valid schedules for an interleaved cache clustered architecture and a multiVLIW architecture, the best one in terms of better cycle count performance has been considered for each benchmark/architecture.

The basic configuration parameters we have used for the three architectures are summarized in Table 1. Note that a cache access latency of one cycle is assumed regardless of the number of ports to it and/or its parameters. For example, in the case of an interleaved cache, each cache module has two read/write ports: one for the local memory functional unit and another for requests from the memory buses. On the other hand, in the case of a unified cache, the cache has 5 read/write ports. In addition, the size of a cache module in an interleaved cache clustered architecture is smaller than the size of the unified cache leading to different access times. The goal of our study is to quantify performance based on cycle count and not execution time. The impact on cycle time will be considered in a future work.

5.2. Evaluation of an Interleaved Cache Clustered Architecture

In Figure 5, memory accesses are characterized for an interleaved cache clustered architecture. In the left hand of the figure, memory accesses are classified into local hits, remote hits, local misses, remote misses and combined accesses. Combined accesses are accesses to subblocks that have been already requested and are still pending, and hence the second request is not issued. These combined accesses can derive in hits or misses and they have just been counted as a separate group. The y-axis represents the ratio of all memory accesses. The left bar for each benchmark in the x-axis represents a basic interleaved cache clustered processor, while the right bar represents an interleaved cache clustered processor with 2-way set associative Attraction Buffers of 16 entries. We have chosen Attraction Buffers with such parameters because they show the best trade-off between capacity and performance as it will be seen latter on. Note that on average, the local hit ratio (the proportion of local hits) is increased by 15% when using such buffers. Thus, the use of Attraction Buffers is an effective technique to decrease remote accesses and in consequence, reduce bus traffic.

On the right hand of Figure 5 remote hits have been characterized to understand why they occur and how to deal with them. In this figure, remote hits have been divided in three groups for the same two configurations: a basic interleaved cluster architecture, and an interleaved cluster architecture with Attraction Buffers. The three groups are as follows:

- **CLOC (Current Loop Other Cluster):** remote hits that access a subblock that has been previously accessed during the current loop by clusters different to the one issuing the request.
- **CLSC (Current Loop Same Cluster):** remote hits that access a subblock that has been previously accessed during the current loop by the same cluster that is issuing the request.
- **OTHERS:** remote hits that access a subblock that has not been accessed previously during the current loop.

The y-axis represents the ratio of remote hits normalized to the basic interleaved cluster architecture. Hence, it can be observed that the number of remote hits decreases when Attraction Buffers are used because remote accesses in general are also decreased.

On average, 50% of the remote hits are due to memory accesses to subblocks that have been accessed previously in the current loop by the same cluster (group CLSC). This indicates a large reuse of subblocks by each cluster except in benchmarks epicenc, mpeg2dec and mpeg2enc. CLSC accesses are due to data that is reused
between iterations and to memory instructions with a stride multiple of \( N \times I \) (where \( N \) is the number of clusters and \( I \) the interleaving factor). This is the primary group targeted by the use of Attraction Buffers and this is why we propose such an enhancement. As can be seen in Figure 5, CLSC remote hits are reduced by 35% when Attraction Buffers are used, while the rest of the groups remain almost the same.

Finally, cycle count results for a basic interleaved cache architecture (Basic) and an interleaved cache architecture with Attraction Buffers (ABuffers) can be seen in Figure 6. The y-axis represents the number of execution cycles normalized with respect to a basic interleaved cache architecture. These cycles have been divided in compute and stall time. Stall time is mainly due to memory instructions that have been scheduled too close to their consumers. We have considered 2-way set-associative Attraction Buffers with 8, 16 and 32 entries.

The main conclusion that can be drawn from this figure is that the proposed scheduling algorithms do a very good job in scheduling memory instructions with an adequate latency (local hit, remote hit, local miss, remote miss) since stall time is small for almost all benchmarks in comparison with compute time. In the case of a basic interleaved cache architecture, stall time accounts only for 9% of the total cycle count execution time, while compute time accounts for 91%.

Secondly, it can be observed that the use of Attraction Buffers is a low complexity solution to reduce stall time. Stall time is reduced by 30% on average and total cycle count execution time is reduced by an average 3%. Total cycle count is especially reduced for benchmarks gsmenc and rasta, where the global benefit is 11% and 5% respectively. In particular, the reduction on stall time for gsmenc is spectacular for two reasons. First, the main loops of the benchmark access 2 byte values. Hence, each subblock that is attracted to a cluster has four 2-byte elements and the subblock is reused more times than if the elements were 4 bytes long. In addition, the small amount of replicated data that Attraction Buffers offer (128 bytes in each cluster in case of Attraction Buffers with 16 entries) is enough to capture the entire working set of these loops.

Finally, results for Attraction Buffers with 8 entries are similar to results for 16 entries, and bigger Attraction Buffers result in better performance as expected. An average 10% overall speedup has been observed if only loops with a big trip count (greater than 10) are considered. From now on, only Attraction Buffers with 16 entries will be considered because they show a good trade-off between capacity and performance.

### 5.3. Comparison with a MultiVLIW Architecture

In Figure 7, results for an interleaved cache architecture with 16-entry 2-way set associative Attraction Buffers and for a multiVLIW architecture with the same amount of resources are shown. Two different configurations have been evaluated: one where the memory bus runs at half the processor frequency (a bus access takes 2 cycles), and one where the bus runs at a fourth the processor frequency (a bus access takes 4 cycles). Results are normalized to the number of cycles of a clustered architecture with a unified cache and a bus running at half the processor clock.

Results show that an interleaved cache clustered architecture has similar cycle count results as a multiVLIW architecture and close to that of a unified cache architecture. Thus, it is feasible to partition a cache in an interleaved manner and still have a competitive performance. The average performance degradation with respect to the multiVLIW is 5% and 8% for 2-cycle and 4-cycle bus latencies respectively, which is heavily biased by two benchmarks: epicdec and rasta. However, an interleaved cache clustered architecture is a lower hardware complexity solution. First, a multiVLIW uses a complex coherence protocol that is not present in an interleaved cache architecture. In addition, the lower effective capacity of the cache in a multiVLIW processor may have an impact on performance for bigger benchmarks. In particular, the same benchmarks have been simulated with smaller caches (2KB) so that the working sets do not fit so well in the cache and in some cases an interleaved cache architecture outperformed the multiVLIW.

Finally, it should be pointed out that since the multiVLIW tends to move data towards the clusters that needed it and its lower effective capacity is enough to hold the benchmarks’ working sets, this architecture is more tolerant to an increase in the bus latency for
these programs as it can be seen by the smaller average performance degradation with 4-cycle buses.

5.4. The Impact of Memory Dependent Chains

Some mechanism must be used to ensure the order of memory accesses when the cache is partitioned. We have proposed the use of memory dependent chains and have scheduled the nodes inside a chain in the same cluster. The use of such a “conservative” technique has an impact in the presented results and hence, this impact is quantified in this section. In Figure 8, results for a basic interleaved cache clustered architecture and a multiVLIW architecture are compared to those gathered when not building memory dependent chains. On the left side of the figure, the normalized number of cycles with respect to a clustered processor with a unified cache is depicted, while on the right side, the local hit ratio is plotted only for an interleaved cache clustered processor.

In the case of an interleaved cache clustered architecture, memory dependent chains have a twofold disadvantage: not only they may unbalance the workload, but they may also increase the percentage of remote accesses. This is clearly important in the epicdec and rasta benchmarks. This impact is even greater if only loops with a big trip count (greater than 10) are considered. However, it seems that even the solution to build memory dependent chains may seem conservative, its performance is still within a very reasonable average slowdown of 5%.

On the other hand, in the case of a multiVLIW architecture, memory dependent chains tend to unbalance the workload. However, they sometimes decrease the amount of coherency traffic inside a loop and increase the effective capacity of the cache by scheduling instructions that access the same data together. Thus, it is not surprising that for some benchmarks (epicdec, jpegdec and rasta), the construction of memory dependent chains reduces the amount of stall time.

Finally, we should add that even though the performance loss due to memory dependent chains is reasonable in an interleaved cache clustered processor, we will also focus on alternative mechanisms to guarantee program correctness in a future work. Special attention will be paid to the epicdec and rasta benchmarks since these are the programs with bigger cycle count degradations.

6. CONCLUSIONS

In this paper, an interleaved cache clustered VLIW processor has been presented and evaluated for 9 Mediabench programs. Effective static scheduling techniques for cyclic code have been developed for such an architecture. Results show that the proposed scheduling algorithms do a very good job in scheduling memory accesses with the appropriate latency, thus generating a small amount of stall time. In the proposed algorithms, coherence is kept by scheduling all memory dependent instructions in the same cluster. Results for such a conservative solution indicate that the cycle count loss is relatively small except for the epicdec benchmark, where the cycle count degradation is 38%. The results for this benchmark advocate for future research in this area.

In addition, the use of Attraction Buffers has been proposed. These buffers act as small caches that hold remote data near the clusters that make use of it. 2-way set associative Attraction Buffers with 16 entries have shown the best trade-off between capacity and performance. Such a hardware enhancement increases the local hit ratio by 15% for the evaluated benchmarks, which results in a reduction of 30% on stall time, a reduction of 3% on global execution time, and a reduction of 10% on global execution time in loops with high trip counts (greater than 10).

Finally, an interleaved cache clustered architecture with Attraction Buffers has been compared with a state-of-the-art clustered processor like the multiVLIW. Results show that such an interleaved cache clustered architecture has almost the same performance as the multiVLIW with lower hardware complexity.

7. ACKNOWLEDGEMENTS

This work has been partially supported by STMicroelectronics and by El Ministerio de Ciencia y Tecnología and the European Union (FEDER funds) reference TIC2001-0995-C02-01. It has been developed using the resources of CESCA and CEPBA.

Enric Gibert would like to thank Josep Maria Codina at UPC, Dan Connors at University of Colorado and all IMPACT group members at University of Illinois for their help.
8. REFERENCES


