Chained In-Order/Out-of-Order DoubleCore Architecture

Miquel Pericàs[†⋆], Adrian Cristal[†⋆], Ruben González[†], Daniel A. Jimenez[‡] and Mateo Valero[†⋆]

[†]Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya
{mpericas,adrian,gonzalez,mateo}@ac.upc.edu

[⋆]Barcelona Supercomputing Center

[‡]Department of Computer Science, Rutgers University
djimenez@acm.org

Abstract

Complexity is one of the most important problems facing microarchitects. It is exacerbated by the application of optimizations, by scaling to higher issue widths and, in general, by increasing the size of microprocessor structures.

This paper presents a new microarchitecture, the Chained In–Order/Out–of–Order DoubleCore Architecture (CIO2), designed to attack the problems of complexity and energy. The CIO2 architecture reorganizes the microarchitecture using the concepts of a centralized register file and the Future File. The resulting architecture decouples that program state from the execution units.

The simplicity of the architecture enables the implementation of three optimizations with little effort: register file banking, writeback filtering and instruction pre–execution. These optimizations allow a reduction of up to 75% in register file energy consumption. Instruction pre–execution further allows around 40% of all integer instructions to execute in the in–order front–end, considerably reducing the activity of the power–hungry issue queues in the out–of–order back–end. Moreover, these improvements are achieved with a negligible performance loss.

1. Introduction

During the last decade, processor design has seen a dramatic increase in complexity and in the power consumption of chips in order to maintain acceptable instructions-per-cycle (IPC) levels while continuing to decrease the clock cycle. Many recent studies have shown how most structures in modern processors are unnecessarily abused leading to the current levels of power consumption. This paper focuses on two of these structures: the register file and the instruction queues.

The register file is often on the critical path and normally has high power consumption. The inefficiency of current register files comes from several sources. In a microarchitecture like the MIPS R10000 [21], register files need to be large and heavily multiported to provide operands for all instructions in the instruction window.

The Future File [18] solves this bottleneck by keeping a small register file in the front–end that represents the program state at the time of instruction decode. Only a subset of the full processor state is available at that time. The remaining operands are obtained while the instruction is waiting in the reservation stations. But this design requires laying out additional buses, adding complexity. In addition, the Future File scheme for recovering from exceptions or branch mispredictions requires the excepting instruction to commit before recovery can start. This can have a large impact on performance.

Instruction queues are another source of inefficiency in processors. They are needed to implement out–of–order execution, a technique devised to tolerate the latency issues that appear when mixing operations with different latencies. As memory latencies increase, more entries are required in the instruction queues. But instruction queues are complex and require the use of large content-addressable memory (CAM) structures. This can be very costly.

The proposal in this paper decouples program state from logic and data necessary for execution. In this paper, *program state* refers to the set of registers necessary for execution plus the set of registers necessary for recovery. All program state is kept in the front–end while execution logic is kept in the back–end as usual. First, the front–end is optimized to reduce its power consumption. Next, pre–execution is added to the design. The new machine is able to execute early all single–cycle integer instructions that have their operands ready in the front-end and it executes the remaining instructions out of program order in the back–end. Thus an in-order core cascades to an out-of-order core. This proposal, the *Chained In-Order/Out-of-Order DoubleCore*

Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05)
1550-6533/05 $20.00 © 2005 IEEE

Architecture (CIO2) has the following major benefits: it reduces the register file power, it reduces the instruction queue and it does this without unnecessarily complicating the design.

This paper is organized as follows. Section 2 will first describe the basic decoupled state-execute machine which aims at reducing the register file power. The optimizations of register file banking and writeback filtering are also described. Section 3 describes the technique of pre–execution and presents the CIO2 architecture. The simulation framework is described in Section 4 and the architecture is evaluated in Section 5. The paper is completed with a description of related research in Section 6. Finally, Section 7 summarizes the main conclusions.

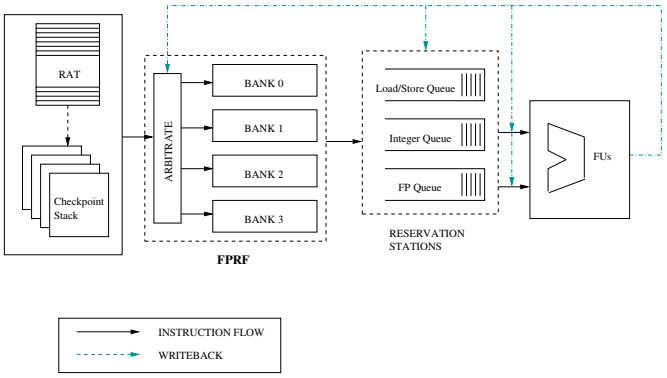

Figure 1. The Decoupled State/Execute Architecture

2. Decoupled State-Execute Architecture

All current speculative superscalar microprocessors use either some variation of the Future File [18] or some form of centralized operand storage indexed via register mappings [11].

Each of these alternatives has its strengths and weaknesses. A register-mapped architecture tends to suffer due to the centralized storage. Keeping all in-flight values in a single register file is inefficient in terms of energy and poses some complexity problems in determining whether a register is currently mapped or if it will be read again. Fast branch misprediction recovery and the scalability of instruction queues are often considered the strong point of such an architecture.

Instruction queues seem to be the bigger problem in a Future File Architecture. In a Future File, registers that have a computed value can be read from a logically indexed register file (the *future file*) at the time of instruction insertion. To make this work, the remaining in–flight values must be somehow obtained. The instruction queues in a Future File Architecture are augmented to contain operand values. This is inefficient because in addition to the register tag, operands need to be broadcast to all the instruction queue slots. Such an organization of the Instruction Queue is normally referred to as *Reservation Stations*.

2.1. DSE Microarchitecture

The *Decoupled State/Execute Architecture* (DSE) is an approach that combines the benefits of the two traditional architectures introduced in the previous section. The basic architecture vaguely resembles a Future File.

The microarchitecture of the DSE is shown in Figure 1. The primary modifications are made to the front–end. The DSE architecture proposes replacing the Future File with a physically indexed register file, the *Front-End Physical Register File* (FPRF). If all physical registers are maintained together, recovery times can be drastically improved as only a single register mapping and free list restoration will be necessary to rollback the processor to a previous execution point.

The use of physical register indices to access register values available from the FPRF forces us to extend the pipeline by one cycle. This cycle is used to implement register renaming using a register alias table (RAT), similar to the MIPS R10000 [21]. In a second stage the FPRF is accessed in case the register descriptor is associated with a computed value. This is checked beforehand during the rename stage.

The remaining parts of the microarchitecture follow the scheme of the Future File. After reading the available registers, the instructions are inserted into the reservation stations along with the operands that they have just read. This is potentially more wasteful because, as indicated before, operand-wide buses need to be laid out along the instruction queue. The pre–execution technique described later will be used to mitigate the problems of the reservation stations.

Note that the FPRF is drawn as a banked register file in Figure 1. The FPRF is large and potentially expensive. Banking is a well known technique that fits well in the DSE architecture. The interaction is explained in the following section.

2.2. Register File Banking

In general, the DSE architecture benefits from making a clean separation between the state (FPRF in Front-End) and the data-flow program execution (implemented by the Instruction Queues and Execution Units in the Back-End). This conceptually clean separation allows implementing some optimizations with little complexity. The following paragraphs explain some optimizations that can be implemented simply yet effectively in the context of the DSE ar-

chitecture. The first of these optimizations is register file banking.

Banking memory structures is a popular technique that can be used to reduce energy consumption and delays of memory structures. In the case of register files [20] the technique is not trivial. Register files are small structures, and improvements in access time have small margins. Thus, area and energy reductions will provide the main advantage.

This is a consequence of having a smaller number of local ports implemented in each bank. Externally, the register file will have as many global ports as an unconstrained single–banked file. When a register is read it is first read out from its bank using one of the local ports and then the value is forwarded to a global port. Each bank must have at least two local ports. Otherwise an instruction that sources two registers mapped to the same bank could never progress beyond the FPRF read stage.

Conflicts are handled in the access to the banked structure by adding an arbitration stage in front of the banked structure. This arbitration logic analyzes the banks that will be accessed in the next cycle and sets up the necessary signals. In addition it checks if all requests can be satisfied. If this is not the case, the first instruction that cannot progress is stalled along with all later instructions in that cycle.

A simple optimization known as *read sharing* [2] can be used to reduce the number of conflicts. Read sharing allows a single local port to be connected to multiple global ports thus giving the impression that more local ports are available in a bank.

Figure 2 shows the 9–stage pipeline of the DSE architecture once banking has been implemented.

2.3. Elimination of transient values via Writeback Filtering

In a basic writeback scheme, all physical registers are written back into the register file once their value has been computed. However, many of these writebacks are not necessary. In an architecture that accesses operands after issue (*issue→read*), if a value has a single consumer, the value was forwarded using the bypass network, and it does not belong to a checkpoint, then there is no real reason to write it back to the register file. This raises the possibility of blocking certain writebacks to reduce power consumption in the register file.

The conceptual distinction of program state (front–end) and execution (back–end) in the DSE architecture allows us to propose such a technique employing simple logic. The DSE architecture has the special characteristic that the FPRF is only used to get values for the current program state or for recovery paths. Taking into account this definition a simple transient value elimination technique, consisting of filtering those values that are not part of the processor state, is proposed. In this research this technique is called *Writeback Filtering*.

The concept of state must be clearly defined:

1. All physical registers that are currently mapped in the RAT (front–end)

2. All physical registers that are mapped in any of the RATs saved during the decode of branches and instructions likely to cause exceptions. These are saved in a *Checkpoint Stack* structure.

The writeback filtering scheme is tightly coupled with the rename logic.

For its implementation it is simpler to use a renamer like the the Alpha 21264 [8] bit vector renamer. The bit–vectors are arrays of bits where each bit identifies a physical register. If the bit is active it means that the register belongs to the current mapping of that checkpoint. Thus, checking whether a physical register belongs to the processor state is equivalent to checking whether the register bit is active in any of the saved mappings or in the current mapping. This can be done by `OR`'ing the different checkpoints with the current mapping to obtain a *global mapping* which can then be used to implement the filter.

3. Chained In-Order/Out-of-Order DoubleCore Architecture

The DSE microarchitecture presented so far, with its early operand read, is well suited to support an implementation of *instruction pre–execution*.

3.1. Motivation

In dataflow execution any instruction can start executing as soon as it has all of its operands available. In the DSE architecture this may happen even before queue insertion. The percentage of integer instructions that have all operands available after FPRF access has been measured over SPEC2000 using the Alpha ISA and the same architecture as in Section 5. On average, around 40% of all integer instructions belong to this group. The largest part correspond to load address calculations (20%) and integer arithmetic (15%). The remaining 5% corresponds to control flow operations and Store Address calculation. This means that about 40% of all integer instructions can potentially be executed right after the FPRF read stage.

3.2. Adding Pre–execution of ready Instructions

Instruction pre–execution refers to this possibility of executing instructions while they are still in the in–order

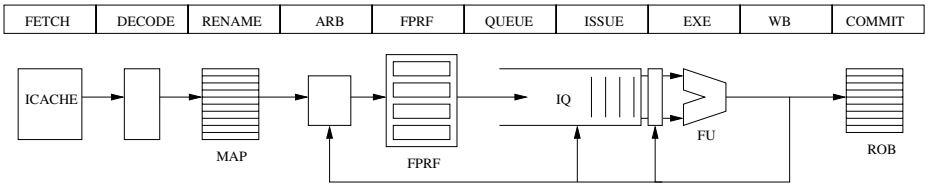

Figure 2. Pipeline of the DSE Architecture

front–end of the processor. The technique extends the in–order front–end so that ready instructions may be executed before they enter the instruction queues. Due to early execution, insertion into the queues is now unnecessary. This saves energy. Instructions that obtain all operands from the FPRF form a category that will be referred to as *early ready*.

The result is a processor that has an in–order execution core for *early ready* instructions followed by an out-of-order execution core. Due to the in–order/out–of–order chaining this new architecture is called *Chained In-Order/Out-of-Order DoubleCore Architecture* (CIO2).

Pre-execution has the interesting side-effect that many loads will have their address calculated some cycles earlier and that some branches are resolved earlier. This enables some improvements in IPC.

Pre–execution is implemented by adding a stage after the FPRF read. During this stage a subset of *early ready* instructions are selected to execute in a collection of dedicated functional units that are located next to the FPRF. Non–pre–executable instructions cross the functional units as if they were `NOP` operations and are then inserted in the reservation stations in the next cycle. A general implementation of this scheme can be very complex due to the handling of multi-latency instructions like multiplications, divisions or floating point operations. Two limitations are imposed to greatly simplify the processing scheme:

1. Only integer instructions may be pre–executed.
2. Only instructions with single-cycle latency will be candidates

FP pre–execution is disabled because little FP instructions obtain all their operands from the FPRF. In addition, pre-executing multi-cycle operations is complex because all operands generated during writeback may need to be bypassed to instructions traversing the pre–execution stage. This number should be minimized.

Pre–Execution is implemented as follows: once the instruction reads the operands, it tries to pre–execute during the next stage. This depends on the availability of functional units and registers. The pre–execution units have only local bypasses and it is only possible to obtain one value using the local bypass. All other values need to be obtained from the FPRF. This strategy also limits the bypass to instructions that belong to adjacent decode groups. In no case will the pre–execution stage stall instructions. If an *early read* instruction cannot be executed due to shortage of resources, then it progresses to the back–end like other non–pre–executable instructions. To better understand the technique, Figure 3 shows the pipeline of the basic CIO2 architecture.

The pipeline shows how the CIO2 architecture adds an additional pipeline stage to handle pre–execution. Adding stages to the front-end means that there will be higher penalties to pay for each branch misprediction. The effectiveness of this scheme depends on how well the load pre-execution capabilities can counter the additional latencies caused by longer branch misprediction recovery.

4. Simulation Infrastructure

The evaluations have been performed using an execution driven simulator that makes use of the simplescalar 3.0 front–end [1]. The cycle–accurate back–end of the simulator has been rewritten from scratch and models all of the techniques presented in this paper: banking, read sharing, writeback filtering and pre–execution.

To evaluate the CIO2 architecture all SPEC2000 benchmarks have been run for 100 million of committed instructions. The benchmark regions have been selected using a criteria based on SimPoint [13]. The simulated benchmark suite consists of binaries compiled using cc DEC C V5.9-008 on Digital UNIX V4.0 and using the `-O2` optimization level.

To study the behavior of the CIO2 microprocessor several microarchitectures are simulated. They represent three steps from a conventional baseline architecture to a fully optimized CIO2 architecture. Table 1 contains the parameters of the baseline microarchitecture. These parameters also apply to the other evaluated configurations unless explicitly stated otherwise.

5. Evaluation

In this section the DSE and CIO2 architectures are evaluated. The evaluation is performed using the following three

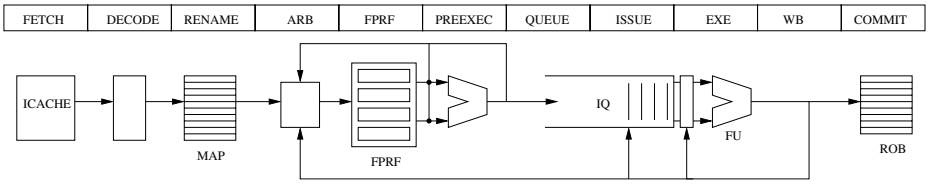

Figure 3. Basic CIO2 pipeline

Table 1. Parameters of the baseline configuration

Fetch/Issue/Commit Width	4 instructions/cycle
D-L1 size	32 KB, 4-way, 2 rd/wr ports, 2 cycle latency
D-L2 size	1 MB, 4-way, 2 rd/wr ports, 10 cycle latency
Memory Width / Latency	32 bytes / 150 cycles
Global Ports to the Register File	8 Read & 4 Write
Reorder Buffer Size	128
Integer/FP Physical Registers	160 / 160
Load/Store Queue	32 entries
Integer/FP Queue	32 entries / 32 entries

different configurations:

1. **Baseline**: The baseline architecture is a centralized DSE architecture whose parameters are shown in Table 1. It implements a single-bank FPRF with 8 read and 4 write ports and it does not implement any of the optimizations discussed in Section 2. The pipeline length is 9 stages.

2. **DSE-OPT**: A banked DSE model with all optimizations active is used to better evaluate the potential of the DSE architecture. It implements a FPRF with 8 banks. Each bank has two read ports and two write ports. This model uses the read sharing and writeback filtering optimizations. The pipeline length is 10 stages (see Figure 2).

3. **CIO2**: The CIO2 architecture, the target of this evaluation, is a DSE architecture extended with pre–execution. It uses a FPRF with 8 banks, each of which has two read ports and three write ports. Of the three write ports, two are statically assigned to the out–of–order back-end and the remaining port is statically assigned to the pre–execution units. All optimizations are used and writeback filtering is applied to both the pre–execution units and to the back–end core. The length of the pipeline is 11 stages (see Figure 3)

5.1. IPCs

Figure 4 shows the IPCs that are obtained by the three evaluated configurations. As can be seen, the differences are very small in this case. The analysis of these results is complex, but here are the main points: First, the FPRF in the front–end has fewer register file accesses than a centralized register file after the issue queues. This results in less bank conflicts and thus less IPC reduction compared to a model such as [19]. Second, in the case of the CIO2 architecture, the pre–execution of instructions helps improve IPC because many loads have their address calculated one cycle earlier. The performance of the architectures must be further analyzed taking into account the fact that the pipeline lengths are different. The DSE-OPT and, more notably, the CIO2 configurations have deeper pipelines than the baseline and will suffer from higher branch misprediction recovery times. The effectiveness of the pre–execution technique is enough to recover and even increase the IPC values for some benchmarks. On average the variations are minimal. The DSE-OPT architecture loses 0.83% IPC while the CIO2 loses only 0.28%.

5.2. Register File Analysis

The FPRF is optimized using three different techniques: read sharing, banking, and writeback filtering. There are several performance factors to check. Here the main concern is energy. Energy is a function of two parameters: the energy consumed by each register file access and the number of accesses to each register file. Table 2 lists the number of reads to the FPRF in each of the three configurations. The percentages of reduction compared to the baseline are also shown. These numbers have been obtained for runs of 100

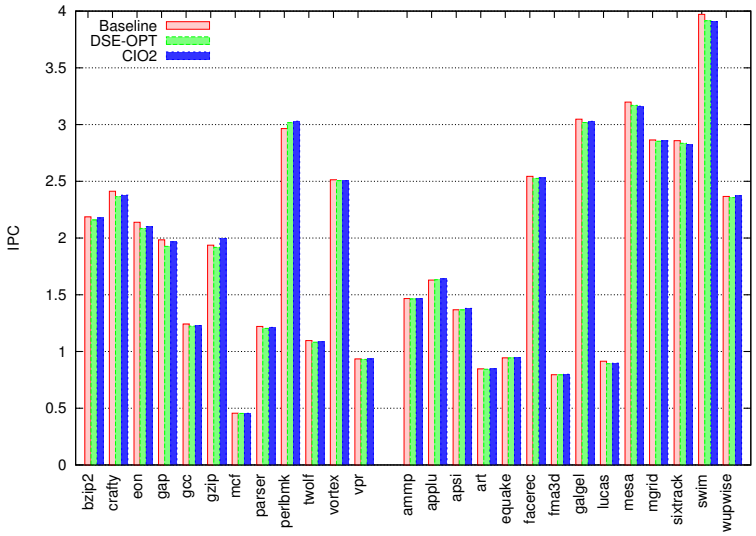

Figure 4. IPCs achieved by the three configurations

million of committed instructions:[1] The number of writes is analyzed in the next section.

Both the DSE-OPT and the CIO2 architecture perform considerably less register file reads. The main force behind this reduction is the technique of read sharing. The CIO2 architecture performs about 5–8% more register reads than DSE-OPT. Pre–execution generates many values early. These values are written back into the FPRF only two cycles after the read. The probability that instructions will read this register is still very high (it decreases with instruction distance).

Table 2 shows also the number of conflicts during reads to the FPRF. The number of conflicts is well below 1% of all accesses. Conflicts happen rather infrequently in the DSE/CIO2 architectures. In any case, a conflict is not very problematic, as there is no recovery procedure involved. This keeps complexity low.

The FP benchmarks show a larger number of conflicts than integer benchmarks. This may come as a surprise, as FP and integer instructions access different register files, which should reduce the conflict rate. However, the fact that FP instructions normally have two sources while most integer instructions normally have a single source is what increases the conflict rate for SpecFP benchmarks.

5.2.1 Effectiveness of Writeback Filtering

The way writeback is handled differs from configuration to configuration. The baseline architecture does not use a writeback filter. All values generated are written back into the FPRF. On the other hand, the DSE-OPT and the CIO2 architectures implement a writeback filter and are thus able to reduce the number of writebacks.

Table 3 shows the total number of writes to the register files for the three configurations in executions of 100 million committed instructions. The baseline has the highest number of writes as it does not implement any filtering. In comparison, the DSE-OPT configuration performs up to 28% less writes for the integer benchmarks and up to 40% less for FP benchmarks. For the CIO2 architecture these numbers are 26% for the integer benchmarks and 36% for the FP benchmarks. The percentage of filtered writebacks is also shown in Table 3.

The CIO2 architecture generates some more writebacks than DSE-OPT. This is because pre–executed values are less likely to be filtered as they are in most cases still part of the current RAT.

The availability of register file access numbers allows to give some energy reduction values. A model for the energy consumption of the register file, similar to [15], has been used. Under such circumstances the DSE-OPT architecture achieves a reduction of 75% in the register file energy, while the CIO2 architecture reduces the energy by 67%. Note that the CIO2 architecture not only performs more accesses, but it also has one additional write port in the front–end for pre–execution writebacks.

5.3. Reductions in Instruction Queue Usage

Pre–executed instructions have the important property that they do not need to be entered in the instruction queues. This can lead to large energy reductions in the CIO2 back–

[1]The number of speculative instructions that actually cross the decode stage is much higher. Hence the large numbers of accesses

Table 2. Number of FPRF Reads and Conflicts (%)

Configuration	SpecINT FPRF Reads	SpecINT FPRF Read Conflicts	SpecFP FPRF Reads	SpecFP FPRF Read Conflicts
Baseline	$666 * 10^6$	0%	$1010 * 10^6$	0%
DSE-OPT	$512 * 10^6$ (-23%)	0.19%	$889 * 10^6$ (-11%)	0.49%
CIO2	$562 * 10^6$ (-15%)	0.26%	$945 * 10^6$ (-6%)	0.54%

Table 3. Number of FPRF Writes and Percentage of Filtered Writes

Configuration	*SpecINT*			*SpecFP*		
	Int Writes	**FP Writes**	**Filtered Int/FP**	**Int Writes**	**FP Writes**	**Filtered Int/FP**
Baseline	$856 * 10^6$	$49 * 10^6$	0%/0%	$409 * 10^6$	$720 * 10^6$	0%/0%
DSE-OPT	$622 * 10^6$	$28 * 10^6$	28%/43%	$307 * 10^6$	$374 * 10^6$	25%/48%
CIO2	$638 * 10^6$	$29 * 10^6$	25%/40%	$330 * 10^6$	$389 * 10^6$	19%/45%

Table 4. Number of Integer Queue Insertions

Configuration	**DSE-OPT**	**CIO2**
Integer Queue Insertions	$1308 * 10^6$	$872 * 10^6$
Address Issues From LSQ	$1095 * 10^6$	$587 * 10^6$

end. The number of queue insertions has been measured to estimate the reduction in the activity of the instruction queues.

The pre–execution technique mainly affects integer operations. Thus only the integer queues (IQ) will see a reduction in their usage. However, the LSQ is also optimized by this technique as it reduces the number of address calculations that need to be performed. The baseline has identical behavior to DSE-OPT in this case and is not shown. Table 4 contains the statistics generated for DSE-OPT and CIO2 averaged over all benchmarks of SPEC2000.

The results show that it is possible to reduce the number of integer queue insertions by 33%. For the address calculation the reduction is around 47%. Although all loads/stores are inserted in the LSQ, there are still improvements in the energy because only a 53% of loads/stores need to issue the address calculations to the ALUs. If the architecture uses a separate queue for the address calculation, that queue would see a decrease in the energy of 47%.

6. Related Work

There is a large amount of research related to the topics discussed in this paper. Unfortunately, due to space constraints, only the most relevant will be covered.

The body of related work on register file energy optimization is large. Many recent papers have proposed mechanisms to reduce the number of the ports by means of modifying the register file architecture, such as [22] [17] [12] [10] [19] [20]. A reduced number of ports may be more efficient both in terms of energy and access time, which can improve performance. The mechanism proposed in this paper is quite orthogonal to these, and also benefits from a reduced number of ports.

The *Writeback Filtering* technique that has been evaluated is related to other work performed in the context of centralized out–of–order architectures such as the work by Ponomarev *et al.* [14] and also in the context of VLIW architectures [16]. However, as can be seen there are significant differences in the complexity between our proposal and these two papers.

The *Pre–execution* technique that leads to the CIO2 architecture is a direct derivation of the DSE architecture. The authors are unaware if there exists any direct prior research, but there are some similarities with previous research. For example, [6] extends the rename stage to implement several optimizations, notably *constant propagation*. These optimizations allow pre–execution of some operations in the front-end. Compared to this proposal, we achieve higher degrees of pre-execution (40% in our proposal vs 25% in [6]). In [3], a technique to pre–execute load addresses is presented. It works by tracking the stack pointer and pre–computing addresses when the value is available early. Again, pre–execution in the context of CIO2 is much more general as all integer single–cycle instructions are candidates for pre–execution, not only load address computations.

The goals of *pre–execution* are to eliminate instructions before they enter the instruction queues. This leads to less power consumption and, to a limited extent, to the possibility to implement smaller queues. The amount of research that has been done in relation to instruction queues is also

very large. But in the general case the approach is to eliminate inherent inefficiencies in the IQ design as is done in [7], [4], [5] and [9]. Instead, our approach does not modify the instruction queue architecture but reduces the number of instructions that have to be processed.

7. Conclusions

This paper has presented the Chained In–Order/Out–of–Order DoubleCore Architecture (CIO2). The CIO2 architecture integrates several techniques to reduce the energy of the register file and of the instruction queues. The paper proposes a subset of the architecture, the decoupled state–execute architecture (DSE), which tries to capture the best from the Future File and from the centralized physical register file architecture. This architecture is then extended with banking, writeback filtering and instruction pre–execution, three techniques that provide excellent results with little complexity. While the optimizations to the register file reduce its energy by around 70%, the addition of pre–execution reduces the number of integer queue insertions by 33%. It also reduces the number of address calculations that have to issue from the LSQ by about 50%. The new architecture, despite its lower energy consumption and longer pipeline loses only 0.28% IPC on average when running the Spec2000 benchmarks.

Acknowledgements

This work has been supported by the Ministry of Science and Technology of Spain under contract TIN–2004–07739–C02–01, the HiPEAC European Network of Excellence under contract IST-004408, and by the Barcelona Supercomputing Center (BSC-CNS).

References

[1] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *IEEE Computer*, 2002.

[2] R. Balasubramonian, S. Dwarkas, and D. Albonesi. Reducing the complexity of the register file in dynamic superscalar processors. In *Proc of the 34th Intl. Symp. on Microarchitecture*, 2001.

[3] M. Bekerman, A. Yoaz, F. Gabbay, S. Jourdan, M. Kalaev, and R. Ronen. Early load address resolution via register tracking. In *Proc. of the Intl. Symp. on Computer Architecture*, 2000.

[4] A. Buyuktosunoglu, D. Albonesi, S. S. ands David Brooks, P. Bose, and P. Cook. A circuit level implementation of and adaptive issue queue for power-aware microprocessors. In *Proc. of the 11th Great Lakes Symposium on VLSI*, pages 73–78, 2001.

[5] D. Ernst and T. Austin. Efficient dynamic scheduling through tag elimination. In *Proc. of the 29th Annual Intl. Symp. on Computer Architecture*, pages 37–46, 2002.

[6] B. Fahs, T. Rafacz, S. J. Patel, and S. S. Lumetta. Continuous optimization. In *Proc. of the 32th Annual Intl. Symp. on Computer Architecture*, 2005. (to appear).

[7] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *Proc. of the 28th Annual Intl. Symp. on Computer Architecture*, pages 230–239, 2001.

[8] R. Kessler. The Alpha 21264 microprocessor. *IEEE MICRO*, 19, Mar. 1999.

[9] I. Kim and M. Lipasti. Half-price architecture. In *Proc. of the 30th Annual Intl. Symp. on Computer Architecture*, pages 28–38, 2003.

[10] N. S. Kim and T. Mudge. Reducing register ports using delayed write-back queues and operand pre-fetch. In *Proc. of the 17th ACM Intl. Conf. on Supercomputing*, June 2003.

[11] J. Liptay. Design of the IBM Enterprise System/9000 high-end processor. *IBM Journal of Research and Development*, 36(4), July 1992.

[12] I. Park, M. D. Powell, and T. Vijaykumar. Reducing register ports for higher speed and lower energy. In *Proc. of the 35th Annual Intl. Symposium on Microarchitecture*, Dec. 2002.

[13] E. Perelman, G. Hamerly, M. V. Biesbrouck, T. Sherwood, and B. Calder. Using SimPoint for accurate and efficient simulation. In *Proc. of the Intl. Conf. on Measurement and Modeling of Computer Systems*, June 2003.

[14] D. Ponomarev, G. Kucuk, O. Ergin, and K. Ghose. Reducing datapath energy through the isolation of short-leved operands. In *Proc. of the 12th Intl. Conf on Parallel Architectures and Compiler Techniques*, 2003.

[15] S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens. Register organization for media processing. In *Proc of the 6th Intl. Symp. on High Performance Computer Architecture*, pages 375–386, 2000.

[16] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon. Exploiting data forwarding to reduce the power budget of vliw embedded processors. In *Proc. of the Conference on Design, automation and test in Europe*, pages 252–257, 2001.

[17] A. Seznec, E. Toullec, and O. Rochecouste. Register write specialization register read specialization: a path to complexity-effective wide-issue superscalar processors. In *Proc. of the 35th Intl. Symp. on Microarchitecture*, pages 383–394, 2002.

[18] J. E. Smith and A. R. Pleszkun. Implementation of precise interrupts in pipelined proccessors. *Proc. of the 12th Intl. Symp. on Computer Architecture*, pages 34–44, 1985.

[19] J. Tseng and K. Asanovic. Banked multiported register files for high-frequency superscalar microprocessors. In *Proc. of the 30th Annual Intl. Symp. on Computer Architecture*, 2003.

[20] S. Wallace and N. Bagherzadeh. A scalable register file architecture for dynamically scheduled processors. In *Proc. of the 5th Intl. Conf. on Parallel Architectures and Compilation Techniques*, pages 179–184, 1996.

[21] K. C. Yeager. The MIPS R10000 superscalar microprocessor. *IEEE Micro*, 16:28–41, Apr. 1996.

[22] V. Zyuban and P. Kogge. The energy complexity of register files. In *Intl. Symp. on Low Energy Electronics and Design*, pages 305–310, 1998.