Designing Efficient Processors Using Compiler-Directed Optimisations

Designing Efficient Processors Using Compiler-Directed Optimisations

Timothy M. Jones†, Michael F.P. O'Boyle†
Jaume Abella‡, Antonio González‡ and Oğuz Ergin§

†Member of HiPEAC,
School of Informatics
University of Edinburgh, UK
tjones1@inf.ed.ac.uk
mob@inf.ed.ac.uk

‡Intel Barcelona Research Center,
Intel Labs - UPC
Barcelona, Spain
jaumex.abella@intel.com
antonio.gonzalez@intel.com

§Dept. of Computer Engineering,
TOBB University of Economics
and Technology, Ankara, Turkey
oergin@etu.edu.tr

Abstract

In the quest for greater performance, superscalar processor designers implement large issue queues and register files to take advantage of the out-of-order execution of the architecture. However, there is a trade-off to be made as performance gains are achieved at the cost of increased energy consumption. There comes a point where increasing the size of these structures is too costly in terms of energy. Recently proposed compiler-directed optimisations can be used to reduce this overhead. Conversely, for the same energy consumption, larger issue queues and register files can be used to increase performance.

This paper considers the design space of issue queue and register file sizes in processors implementing a combination of issue queue throttling and early register releasing schemes under compiler control. Compared with the best baseline containing 64 issue queue entries and 96 integer registers, our scheme with a configuration of 80 entries and 80 registers can achieve an energy-delay-squared (EDD) product of 0.952 without any loss of performance and no increase in energy consumption. The same configuration without compiler optimisations has a EDD product of 1.078, losing 3% performance. Furthermore, this scheme can be applied to a range of baseline processors allowing designers to achieve EDD products as low as 0.880 whilst still maintaining at least the same performance and energy budget.

1. Introduction

Within a superscalar processor instructions and data are held in complex logic and structures as they pass through the pipeline. Increases in the number of in-flight instructions held in the out-of-order processor have contributed to greater performance, in part by increasing the size of the internal structures. However, larger structures consume greater amounts of energy and require sophisticated cooling systems, the cost of which will increase non-linearly compared with the amount of heat removed in the future [1].

This paper proposes the use of a combination of compiler-directed optimisations to aid the design of the processor. In essence, a different processor configuration using compiler optimisations can be selected that has a better energy-delay (ED) product, better energy-delay-squared (EDD) product, saves more energy or produces better performance, depending on the constraints designed for. The ED and EDD products are important metrics for processor designers as they capture the trade off between increased performance and increased energy consumption. This is explained in more detail in section 6.2.

The proposed compiler optimisation targets both the issue queue and integer register file using a combination of previously proposed schemes. The issue queue and register file are two of the most energy-consuming structures within the processor [2] yet are central to performance since the larger they are, the more in-flight instructions can be present in the processor and the greater the instruction throughput.

Recent work has proposed throttling the issue queue [3, 4], that is limiting the number of instructions it is allowed to contain, so as to reduce the energy consumed in waking operands and reading and writing data. However, these schemes result in non-negligible performance losses. Early register releasing has also been used by previous researchers for IPC gains [5] and energy savings [6]. These schemes release registers much earlier than they would usually be enabling empty registers to be turned off for energy savings. We build on these schemes to include the issue queue, which consumes a greater proportion of the total system energy, so that smaller EDD products are achieved.

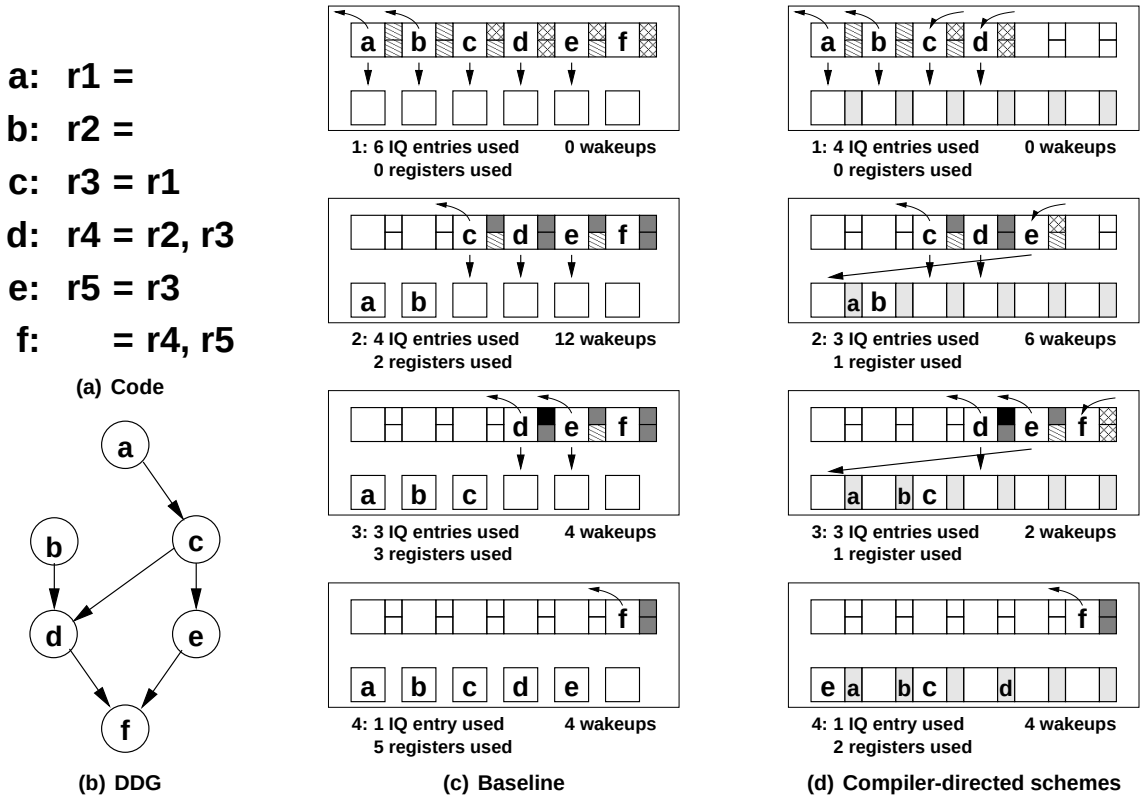

Figure 1. Pseudo-code, its data dependence graph and passage through the pipeline in the baseline processor and one that implements compiler-directed issue queue throttling and early register releasing. In this combined scheme, fewer wakeups occur (12 vs 20) occur and fewer issue queue entries (11 vs 14) and registers (4 vs 10) are used over all cycles.

The rest of this paper is structured as follows. Work related to this paper is described in section 2. We then motivate the use of issue queue limiting and early register releasing for performance improvements and energy savings in section 3. Details of the compiler analysis for the issue queue throttling and early register releasing techniques that we implement are given in section 4. Section 5 describes the microarchitecture of our scheme and the changes needed to allow issue queue limiting and early register releasing to work. Section 6 describes the use of our work to minimise the ED and EDD products in the design of issue queue and register file sizes and presents the results of our experiments. Finally, section 7 concludes this work.

2. Related Work

To the best of our knowledge there is no existing research that combines dynamic issue queue throttling with releasing registers early. However, separately, both techniques have been studied in detail.

Considering issue queue schemes, useless activity can be reduced by gating off the precharge signal for tag comparisons to empty or ready operands [2]. A dynamic throttling scheme is also proposed in [2] to increase the number of tag comparisons gated off in this way. A banked issue queue design is used by Buyuktosunoglu *et al.* with a similar resizing approach, where empty banks can be turned off for increased dynamic and static energy savings [7]. Other researchers have used this banked issue queue and proposed different heuristics for issue queue throttling based on queue theory [3] and compiler inferred knowledge [4]. We implement a similar technique to this latter scheme and it is described in section 4.1. A further technique dynamically adjusts the issue queue, reorder buffer and load-store queue sizes [8]. However, all of these throttling schemes experience a non-negligible performance loss which degrades their EDD product.

There have been several approaches to early register releasing under both hardware [9] and software [10] di-

rection. However, neither of these schemes implements precise interrupts or exceptions. Ergin *et al.* [5] proposed the checkpointed register file to store copies of registers that had been released early. They released once a register had been redefined, all consumers had started execution and the original defining instruction had committed. A compiler-directed early releasing scheme has also been designed which uses this checkpointed register file to provide precise interrupts and exceptions [6]. We use a similar early releasing technique, explained in more detail in section 4.2. However, in all cases only the register file energy is considered, whereas this paper considers total system energy and performance.

This paper proposes the combination of issue queue throttling with early register release through compiler-inferred information to aid the design of the issue queue and register file in a superscalar processor. Early releasing improves register utilisation and eliminates performance losses experienced by issue queue throttling. We show that by taking compiler optimisations into account, different configurations of these structures can be chosen that minimise the ED and EDD products without increasing total energy consumption or decreasing performance compared with the best configuration chosen with no compiler optimisations.

3. Motivation

This section motivates the use of issue queue throttling and early register releasing to save energy and increase the utilisation of these two important processor resources. Figure 1 shows an example run of some code in both the baseline and compiler-directed approaches. The pseudo-code is shown in figure 1(a) and its data dependence graph (DDG) in figure 1(b). As can be seen in the DDG, there are several registers only used once (r1, r2, r4 and r5) and the dependences between instructions mean that it will take several cycles for them all to be executed.

The issue queue and register file for the baseline machine are shown in figure 1(c), where the issue queue is above the register file and arrows between the two indicate which register will be written by each instruction. In the first cycle, a and b dispatch, there are no wakeups and no registers are in use, but 6 issue queue entries are filled. In the second cycle, a and b write back and their operands are forwarded to the issue queue where they cause 12 wakeups (each causes 1 wakeup for c, 2 for d, 1 for e and 2 for f). The registers get written with the correct value and, for simplicity, in figure 1(c) we have just shown the instruction name in the register file. Here, 4 issue queue entries contain instructions and 2 registers are filled. In cycle 3, 4 wakeups are caused by c writing back, 3 issue queue entries are used with 3 registers filled. Finally, in cycle 4, d and e writeback causing 4 wakeups and f issues. In this cycle only 1 issue queue entry is needed but 5 registers are active. In total, the baseline causes 20 wakeups using 5 registers.

Now consider the same code running on a processor implementing issue queue throttling and early register releasing schemes. In this example we have limited the issue queue so that it contains a maximum of 2 instructions after every cycle. We have also shown the checkpointed register file required by the early register releasing scheme, more details of which can be found in section 5.2. Figure 1(d) shows the passage of the instructions through the issue queue and register file again.

In cycle 1, a and b issue, making room for c and d to dispatch and there are no wakeups. The second cycle sees a and b writeback again but they only cause 6 wakeups compared with the baseline's 12, because there are fewer instructions in the issue queue. Instruction a defines a single-use register and the act of c issuing causes the register to be written but placed in the shadow bitcells of the register through checkpointing [1]. In this cycle only 3 issue queue entries are used and 1 main register is filled. Instruction e dispatches and can reuse the register written by a since the main part of it is free. In the third cycle, c causes 2 wakeups and d and e issue. As d issues, it starts a checkpoint of b since this is also single-use, leaving the main register free for another instruction to use. In the final cycle, d and e writeback and, since f issues, d is checkpointed. The result of instruction e cannot be checkpointed even though it is a single use value, because a is already held in the shadow bitcells of this register. In total, 12 wakeups occur using this scheme, saving energy. The maximum number of registers used is 2, allowing more energy savings or the possibility of using a smaller register file.

4. Compiler Analysis

This section describes the combined compiler analysis performed that calculates the issue queue size needed for a program region and identifies registers that can be released early based on previous research [4, 6]. The resulting binary can be run on a processor implementing issue queue throttling and early register releasing, leading to more efficient use of processor resources and enabling designers to minimise the ED and EDD products. The microarchitecture changes need to support such techniques are described in section 5.

1 In reality a would be checkpointed in the following cycle but we show it happening now for simplicity

1. Build the whole procedure's data dependence graph, $\mathbf{DDG}$, and identify single-use registers
2. $\forall$ instructions $\in \mathbf{DDG}$
   (a) If the destination register is single-use
      i. Rename this register and all consumers to one of the special single-use registers
3. Create the procedure's control flow graph, $\mathbf{CFG}$
4. Find natural loops and separate $\mathbf{CFG}$ into a set of DAGs, $\mathbf{DAGS}$, and loops, $\mathbf{LOOPS}$
5. Build each DAG and loop's critical path dependence graph
6. $\forall \mathbf{D} \in \mathbf{DAGS}$
   (a) Iterate over $\mathbf{D}$ recording youngest and oldest nodes reached
   (b) Use maximum distance between these sets of nodes to determine issue queue requirements
7. $\forall \mathbf{L} \in \mathbf{LOOPS}$
   (a) Create equations relating each node in $\mathbf{L}$ to its predecessors
   (b) Rearrange the equations and solve to determine issue queue requirements
8. Tag the entry point to each DAG and loop with the issue queue size required

Figure 2. Algorithm for performing the compiler analysis on a procedure.

4.1. Issue Queue Throttling

To analyse a procedure's issue queue requirements the compiler identifies two different kinds of program structure, namely directed acyclic graphs (DAGs) and loops. Analysis is specialised for each having first constructed a dependence graph which represents the critical path through a program region [11, 12]. Edges in the dependence graph are weighted with the number of cycles it takes to resolve the dependence between two nodes. Each instruction is represented by at least two nodes which correspond to the instruction's dispatch and issue (and issue from the load/store queue in the case of a memory instruction).

Analysis and computation of the issue queue requirements is specialised for directed acyclic graphs (DAGs) and loops. However, both involve iterating over the dependence graph to determine sets of instructions that will all be ready for issue on the same cycle. For DAGS, the distance between the youngest and oldest instruction on any cycle is important. For loops equations are set up, rearranged using a simple algorithm and then solved to calculate requirements. For more details about the analysis needed for this, please see [4].

The first instruction in each program region is tagged with the number of issue queue entries needed using redundant bits in the ISA. This value is then decoded by the processor at dispatch and used to throttle the queue, as described in [4]. In practice only a few bits are needed since it is only the size of the youngest part of the queue that is encoded and we believe 3 bits should be sufficient. Section 5.1 gives details of the issue queue and the microarchitecture changes needed to allow this limiting to occur.

4.2. Early Register Releasing

Early register releasing analysis is performed by constructing the control flow and data dependence graphs for each procedure within the compiler. These are then used to identify single-use registers: registers used only once along each path in the control flow graph from producer to each consumer. A fixed number of logical registers are designated single-use registers and those previously identified are renamed to use a register from this set, if one is available. When the processor sees an instruction using one of these then it knows that there will be no more uses of the register and it is safe to release it early. A backup copy is taken for use in the event of a branch mis-prediction, interrupt or exception. Section 5.2 contains details of the register file and the microarchitecture changes needed to implement this.

The processor also releases another set of registers upon each call or return instruction that is committed. These registers are the caller-saved registers that are not live across the procedure boundary and therefore are guaranteed not to be used again before first being redefined. Again, a copy of these registers is taken so that they can be restored should the precise processor state be needed.

An overview of the algorithm for determining issue queue requirements and performing early register releasing analysis is shown in figure 2.

5. Microarchitecture

Our processor is an out-of-order superscalar with a centralised architectural register file and issue queue. This section describes in more detail these two structures and the mechanisms provided to allow compiler-

directed issue queue throttling and early register releasing to produce processors that with more efficient resource utilisation.

5.1. Issue queue

The issue queue used in this paper is similar to that used in [7]. It is non-collapsible and instructions are placed in sequential order with no compaction, as this would result in a significant energy and complexity overhead. The issue queue is banked with eight entries per bank and the compiler-directed schemes can turn the banks off when they contain no valid instructions. The selection logic remains permanently on but consumes much less energy than the wakeup logic [13]. As in [2], the compiler-directed schemes allow the gating off of the precharge signal for tag checking when an operand is empty or ready.

The issue queue throttling is instigated by a tag on an instruction using redundant bits in the ISA. The calculation of the value in this tag is described in section 4.1. A new pointer into the queue is needed, called the *new_head* which sits between the *head* and *tail* pointers. The *tail* can be restricted such that there is a maximum number of entries it can be away from the *new_head*, i.e. the youngest part of the queue can only be a certain size. When the processor dispatches an instruction with a tag, it decodes the value it contains and uses it as this maximum distance. Instructions cannot dispatch if the youngest part of the queue would get too big.

5.2. Register File

We use a checkpointed register file [5] to allow the recovery of the precise processor state after early releasing in the case of an interrupt or exception. Here, cheap backup storage is provided next to each register, called shadow bitcells, to hold a copy of the main register value. The delay overhead is less than 0.5% for the maximum sized register file we evaluate [5]. There is an increase in the wordline and bitline energy consumption, but this is a small amount and is accounted for in all our experiments.

The register file is banked in the same way as in [3]. In the baseline the banks are always on, but the schemes with compiler-directed issue queue limiting, early register releasing or combined optimisations have the ability to automatically turn off register banks when they contain no valid data (in the main or checkpointed bitcells).

5.3. Reorder Buffer and Map Tables

Instructions are held in the reorder buffer as they pass through the pipeline between dispatch and commit. It holds information about the source and destination registers used and information about any early releasing that has occurred. The register dispatch and retirement map tables keep track of the mapping between logical and physical registers at the dispatch and commit stages of the pipeline. Extra bits are added to both tables to indicate whether early releasing can take place and whether the logical register resides in the main or shadow bitcells of the physical register pointed to.

Table 1. Processor configuration

Machine width	8 instructions
Branch predictor	16K gshare
BTB	2048 entries, 4-way
L1 Icache	64KB, 2-way, 32B line, 1 cycle hit
L1 Dcache	64KB, 4-way, 32B line, 2 cycles hit
Unified L2 cache	512KB, 8-way, 64B line, 10 cycles hit, 200 cycles miss
ROB size	128 entries
Int FUs	6 ALU (1 cycle), 3 Mul (3 cycles)
FP FUs	4 ALU (2 cycles), 2 MultDiv (4 cycles mult, 12 cycles div)

6. Results

This section describes the use of our combined compiler-directed optimisations to design more efficient processors with better ED and EDD products than the baseline.

6.1. Compiler, Simulator and Benchmarks

Our compiler analysis was written in MachineSUIF [14] and results were obtained from execution on the Wattch [15] simulator, based on SimpleScalar [16]. The main components are shown in table 1. The issue queue and register files were banked into groups of 8 and their full sizes are described with each result.

Our benchmarks are the Spec2000 suite with the exception of *eon* and the floating point benchmarks which could not be compiled. We ran each benchmark with *ref* inputs for 100 million instructions after skipping the initialisation part and warming the caches and branch predictor for 100 million instructions.

6.2. Evaluation

The aim of this paper is to show that a combined compiler-directed issue queue throttling and early register releasing scheme can be used to design more efficient processors. We decided to use the energy-delay

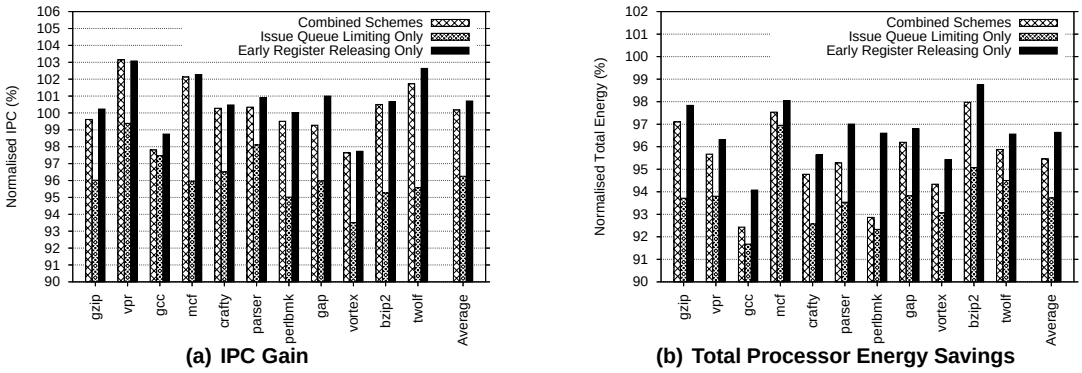

Figure 3. Performance and total processor energy normalised to the baseline for combined schemes producing the smallest ED and EDD products and the separate compiler-directed techniques alone. The baseline has an issue queue of 64 entries and a register file of 96 registers. The compiler-directed schemes have an issue queue of 80 entries and 80 registers. The combined scheme saves energy without losing performance.

(ED) and energy-delay-squared (EDD) products to define efficiency. These are important metrics in microarchitecture design because they indicate how efficient the processor is at converting energy into speed of operation, the lower the value the better [17]. The ED product implies that there is an equal trade off between energy and delay, whereas the EDD product places more emphasis on increasing performance.

To choose a baseline configuration we ran the benchmarks on processors with issue queue sizes ranging from 32 to 80 in steps of 8, and register file sizes from 40 to 112, also in steps of 8. We chose the configuration 64 issue queue entries and register file sizes of 96 registers which represents the best trade off between performance and energy consumed. All other configurations have a greater EDD product in comparison to this, meaning that this is the optimal configuration for this metric.

To determine the effects of our combined issue queue throttling and early register releasing schemes we ran our benchmarks on all configurations of the processor that we evaluated for the baseline, then calculated the ED and EDD products for each with respect to the baseline chosen. In all evaluations we used the total energy of the processor. To calculate this we assumed that the issue queue contributes 20% and the integer register file 10% of the total energy budget, although other researchers have found the issue queue's contribution can be as high as 27% [2]. We considered both dynamic and static energy, assuming that leakage accounts for 25% of the energy for each structure.

6.2.1. Minimising ED and EDD We first considered the case where a processor using the compiler-directed schemes can be implemented with minimal ED and EDD products yet no decrease in performance or increase in energy consumption. Of all configurations studied, we found that the combined scheme's smallest ED product of 0.953 is achieved when using an issue queue of 80 entries and a register file of 80 registers. This configuration also produces the smallest EDD product of 0.952. In comparison, a processor without the compiler optimisations and the same configuration has an ED product of 1.046 and an EDD product of 1.078, suffering a 3% performance loss.

Figure 3 shows the performance and total processor energy consumption of this combined scheme, and that of the same processor configuration using only issue queue limiting and only early register releasing. As can be seen in figure 3(a), the issue queue throttling scheme alone loses almost 4% performance, even though its energy consumption is low compared with the baseline (figure 3(b)). On the other hand, early register releasing alone achieves a slight IPC gain, but doesn't save as much energy. Combining the two schemes gives good energy savings and no performance loss, on average, resulting in low ED and EDD product values, producing a configuration that is better than the baseline.

6.2.2. Performance and Energy A processor designer may have other constraints to work with and so we next considered two further characteristics to optimise: best performance and lowest total processor energy consumption. Using these metrics, different configurations of the processor implementing compiler-directed optimisations could be chosen.

Figure 4 shows the performance and energy consumption of differing configurations of the combined scheme compared to the baseline when optimising for

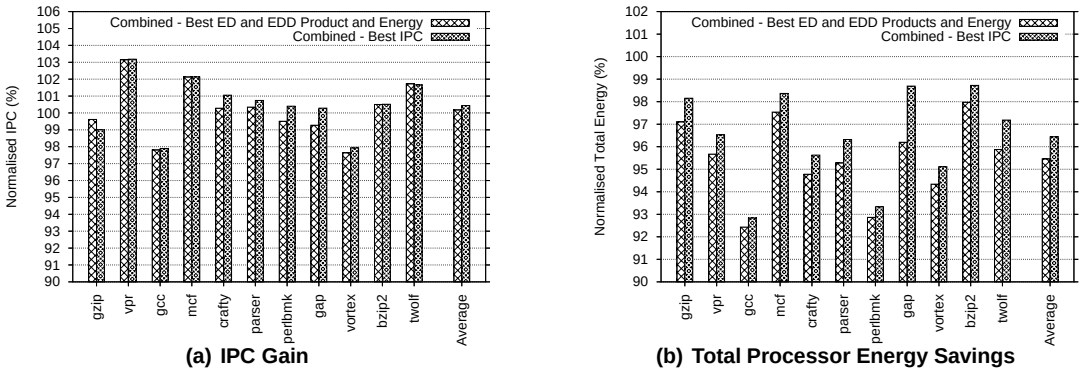

Figure 4. Performance and total processor energy normalised to the baseline for combined schemes producing the smallest EDD product, best performance gain and lowest energy consumption. The baseline has an issue queue of 64 entries and 96 registers. All combined schemes improve performance and save energy but differ in issue queue and register file sizes.

the best ED and EDD product, maximum performance (without increasing energy consumption) and minimum energy consumption (without incurring a performance loss). The configuration with the best ED and EDD product is that described in section 6.2.1 with 80 issue queue entries and 80 registers. That with the greatest performance also has 80 issue queue entries but 112 registers. Finally, the configuration with the smallest energy consumption has 80 issue queue entries and a register file of 80 registers again.

As figures 4(a) and 4(b) show, different design constraints lead to different configurations of the processor when compiler-directed optimisations are implemented. In all three cases, the performance and energy consumption are the same or better than in the baseline processor and the ED and EDD products are less than 1, showing the compiler optimisations can be used to implement more efficient processors over a range of design constraints and that the most efficient configuration changes when compiler optimisations are taken into account.

6.2.3. Baseline Reconfiguration To verify that our combined compiler-directed issue queue throttling and early register releasing schemes can be applied to various baseline configurations and still produce more efficient processors, we altered the issue queue and register file sizes and computed the ED product, EDD product, relative performance and energy consumption for all combined schemes. The baseline configurations chosen all have 32 more integer registers than there are issue queue entries, although this need not always be the case. For each baseline we chose the combined scheme achieving the best result for each design metric, shown in figure 5.

For each baseline configuration analysed, use of the compiler-directed schemes can improve the ED product, EDD product, IPC or energy consumption. In fact, with a baseline processor containing 40 issue queue entries and 72 integer registers, one configuration using compiler-directed optimisations achieves an ED product of 0.933 and another an EDD product of 0.880.

As can be seen in figure 5(a), none of the compiler-directed combined configurations decreases performance compared to the baseline that it is designed for. Likewise, all configurations produce energy savings, shown in figure 5(b). These figures show that our approach of using combined compiler-directed issue queue limiting and early register releasing can aid processor designers improve the ED product, EDD product, performance and total processor energy consumptions of their architectures, no matter what the original baseline configuration.

7. Conclusions

This paper has presented a novel approach to the design of the issue queue and register file within a processor. Our proposed scheme uses compiler-directed issue queue throttling and early register releasing to use these microarchitecture resources more efficiently than in the baseline configuration. Implementation of a processor that can take advantage of this compiler analysis can allow the designer to produce issue queues and register files of differing size configurations depending on the constraint being optimised. The energy-delay and energy-delay-squared products can be minimised to produce processors that efficiently convert energy into instruction throughput, and processor configurations can be designed that maximise performance without increas-

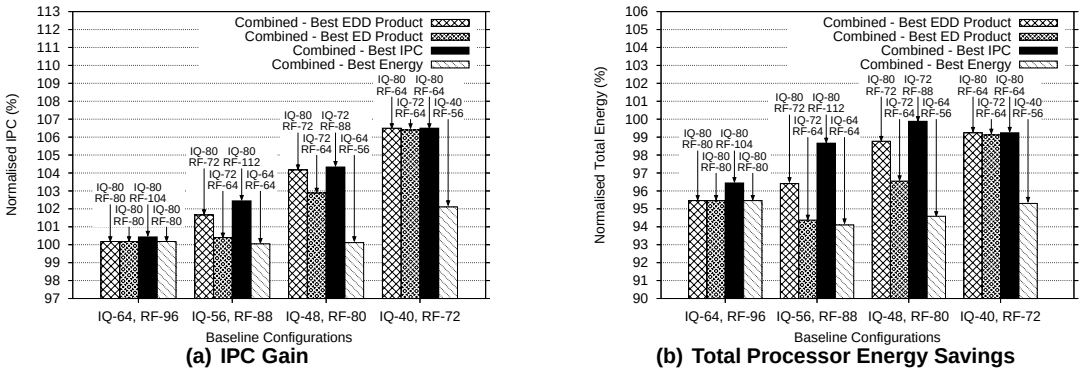

Figure 5. Performance and total processor energy normalised to differing baseline configurations for the combined schemes producing the smallest ED product, EDD product, best performance gain and lowest energy consumption with respect to that baseline.

ing energy consumption, or minimise energy consumption without sacrificing performance. Furthermore, an evaluation of this design strategy over various baseline configurations shows that our scheme can be applied to differing issue queue and register file sizes and still yield improvements in all metrics over the baseline.

In conclusion, were processors designed with these compiler-directed optimisations, different configurations of the issue queue and register file would be implemented, giving increased efficiency with no performance loss of increase in energy consumption.

Acknowledgements This work has been partially supported by The Spanish Ministry of Education and Science under grants TIC2001-0995-C02-01, TIN2004-03072, FEDER funds and Intel Corporation.

References

[1] S. H. Gunther, F. Binns, D. M. Carmean, and Jonathan C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, Q1, 2001.

[2] Daniele Folegnani and Antonio González. Energy-effective issue logic. In *Proceedings of ISCA-28*, 2001.

[3] Jaume Abella and Antonio González. Power-aware adaptive issue queue and rename buffers. In *Proceedings of HiPC-10*, volume 2913 of *LNCS*. Springer, 2003.

[4] Timothy M. Jones, Michael F.P. O'Boyle, Jaume Abella, and Antonio González. Software directed issue queue power reduction. In *Proceedings of HPCA-11*, 2005.

[5] Oguz Ergin, Deniz Balkan, Dmitry Ponomarev, and Kanad Ghose. Increasing processor performance through early register release. In *Proceedings of ICCD-22*, 2004.

[6] Timothy M. Jones, Michael F. P. O'Boyle, Jaume Abella, Antonio González, and Oğuz Ergin. Compiler directed early register release. In *Proceedings of PACT*, 2005.

[7] A. Buyuktosunoglu, S. Schuster, D. Brooks, P. Bose, P. Cook, and D. Albonesi. An adaptive issue queue for reduced power at high performance. In *Proceedings of PACS*, volume 2008 of *LNCS*. Springer, 2000.

[8] Dmitry Ponomarev, Gurhan Kucuk, and Kanad Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *Proceedings of MICRO-34*, 2001.

[9] Teresa Monreal, Víctor Viñals, Antonio González, and Mateo Valero. Hardware schemes for early register release. In *Proceedings of ICPP*, 2002.

[10] Jack L. Lo, Sujay S. Parekh, Susan J. Eggers, Henry M. Levy, and Dean M. Tullsen. Software-directed register deallocation for simultaneous multithreaded processors. *IEEE TPDS*, 10(9), 1999.

[11] Dean M. Tullsen and Brad Calder. Computing along the critical path. Technical report, UCSD, 1998.

[12] Brian Fields, Shai Rubin, and Rastislav Bodík. Focusing processor policies via critical-path prediction. In *Proceedings of ISCA-28*, 2001.

[13] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *Proceedings of ISCA-24*, 1997.

[14] Machine SUIF. http://www.eecs.harvard.edu/machsuif/-software/software.html.

[15] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of ISCA-27*, 2000.

[16] D. Burger and T. Austin. The simplescalar tool set, version 2.0. Technical Report TR1342, University of Wisconsin-Madison, 1997.

[17] Chung-Hsing Hsu, Wu chun Feng, and Jeremy S. Archuleta. Towards efficient supercomputing: A quest for the right metric. In *Proceedings of the High-Performance Power-Aware Computing Workshop*, 2005.