Adaptive Pipeline Depth Control for Processor Power-Management

Aristides Efthymiou Jim D. Garside
Department of Computer Science, The University of Manchester,
Oxford Road, Manchester M13 9PL, UK
{ae,jdg}@cs.man.ac.uk

Abstract

A method of managing the power consumption of an embedded, single-issue processor by controlling its pipeline depth is proposed. The execution time will be increased but, if the method is applied to applications with slack time, the user-perceived performance may not be degraded. Two techniques are shown using an existing asynchronous processor as a starting point. The first method controls the pipeline occupancy using a token mechanism, the second enables adjacent pipeline stages to be merged, by making the latches between them 'permanently' transparent. An energy reduction of up to 16% is measured, using a collection of five benchmarks.

1. Introduction

Power consumption has become a serious concern to both circuit designers and system architects, typically in the embedded systems area, where low power consumption may be more important than speed. Recently there is considerable interest in techniques that cause an increase in delay for an improvement in energy dissipation. These techniques may not be useful if the system under consideration is a busy, high-performance server which always has computationally demanding tasks to perform, because the extra delay imposed by those methods cannot be hidden. However they can be applied successfully in embedded and other systems that have idle time, either while waiting for user input or when running applications with soft real-time deadlines (e.g. some multimedia applications or communication protocols). In these systems the slack time may be better exploited for energy efficiency by slowing down the application rather than running the system at normal (high) speed and putting it to sleep for the remaining time.

Dynamic voltage scaling (DVS) is typically used for this purpose, but its slow transition times — usually a few μs — and extra hardware requirements put limitations on its use [3]. Thus a number of architecture adaptation techniques have recently been published which have the advantage of faster response times, albeit with lower power savings. These techniques save more energy compared to simply reducing the clock frequency, by reducing the effective switched capacitance of the processor.

The work presented here uses asynchronous design techniques [9] to build hardware that is capable of adapting to changing energy/performance requirements at run time. Asynchronous design makes the timing of the processor units much more flexible, being able to stretch or shorten "cycle" times at various places in a pipeline.

AMULET3, which is used as the base processor for the techniques presented, implements the ARM 4T architecture and delivers around 100 Dhrystone MIPS, for a power consumption of 137mW [5]. Its pipeline latches are controlled by latch controllers using handshake 'request-acknowledge' signals between neighbouring pipeline stages. In brief, when a latch controller receives an input request, it checks the status on its output to find if the previous output has been acknowledged. If it has, the controller loads its latches and acknowledges the incoming request. Otherwise, it delays the action until the previous transaction on the output side is complete.

2. Adaptivity in scalar processors

Although there are numerous reported architecture adaptation techniques for superscalar processors [7] [2] [3] [6], there are none for single issue processors. Admittedly there is far more field for experimentation in superscalar architectures, but the majority of embedded systems still use single-issue processors. The primary method of reducing the effective switched capacitance, even in a single-issue processor, is to control speculation, i.e. any operation which is performed before it is *known* to be required.

In this sense the most 'profound' type of speculation is pipelining; letting instructions into the processor pipeline before knowing for certain that the instruction flow is not about to branch is clearly speculative. In most instances this is justified, but discarded instructions that are fetched and

1063-6404/02 $17.00 © 2002 IEEE

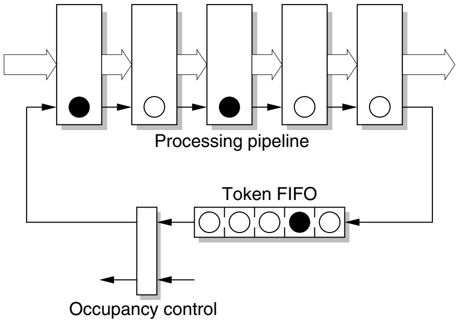

Figure 1. Pipeline occupancy control

partially processed waste energy. Furthermore to support or improve pipelining, extra operations like branch prediction are employed which operate and consume energy in every cycle. Although a branch predictor would save some energy by avoiding some wrongly fetched instructions, it is questionable whether the energy consumed by the predictor is less than that saved. Based on these observations, this work considers controlling the pipeline depth adaptively to trade speed for energy.

3. Pipeline occupancy control

A simple way to control the pipeline occupancy of a processor is to use a token FIFO with as many slots as pipeline stages between prefetch and execute. Prefetch would collect a token before trying to fetch an instruction and execute would return one for each completed instruction. Figure 1 shows an abstract view of this idea. The number of tokens available directly controls the maximum occupancy of the processor pipeline, which can range from zero (stalled) to fully occupied. In all but the fully occupied case some energy will be saved because fewer speculative operations will be wasted but, at the same time, the performance will be degraded. Tokens can be inserted or removed from the FIFO, using control circuits which can be under software or hardware supervision.

Incorporating this technique into AMULET3 is unfortunately not as straightforward as the above discussion implies. Instructions can be removed from the pipeline before the execute stage, for example when they are in the shadow of a taken branch. The instructions that have acquired tokens must return them to the token pool, otherwise the processor will eventually stall, deprived of tokens. A further complication is that long multiplications are executed in two cycles, so the execute stage is invoked twice for the same instruction. In this case only one token must be returned to the token pool. ARM's multiple load/store instructions cause similar, if more extreme, problems. Because of the above complications, some small modifications had to be carried out in the prefetch and execute units to be able to support token-based pipeline occupancy control. In addition a new handshake "channel" was inserted between the decode and execute stages to pass on the token when an instruction is removed.

4. Dynamic configuration of pipeline stages

Another way to control the pipeline depth is to join pipeline stages together. This must be done dynamically, so that the processor can return to its original, fully-pipelined configuration very quickly. The effect is similar to the token method outlined above, but more flexible because of the ability to control specific pipeline stages independently of the rest of the processor. Because of the asynchronous design style, the control of each stage is independent of — and cannot use any information from — the other stages. Thus joining pipeline stages does not require any global control changes, as might be expected in result-forwarding paths, for example.

Any combination of adjacent pipeline stages can be joined, simply by making the pipeline latches between them transparent. Thus the latch controllers must be able to be configured appropriately. The method used here is to add an extra input (*collapse*) to the latch controllers,which sets them in either 'collapsed' or in normal operating mode. The pipeline latches cannot be made transparent at any time while the system is working, but only when the downstream stage has finished processing its instruction and is ready for the next. Thus, *collapse* is 'bundled' with the the rest of the latched signals, and it takes effect when the request gets through to the next stage.

In the current implementation, *collapse* is latched like the other inputs of the pipeline latches and passed on to the next stage. Thus each latch controller mimics the one upstream and the prefetch unit controls them all. If individual controlling is required, because the *collapse* signal must be timed independently for each pipeline latch controller, a set of *collapse* signals can be generated at the prefetch unit and then passed on from each stage to the next until they reach their target latch controllers (figure 2).

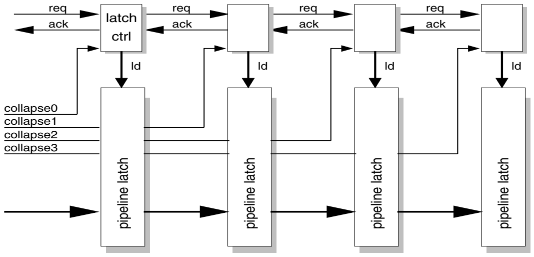

Figure 2. Individually controlled collapsible latch controllers

Dynamic configuration of pipeline stages has the potential to save some extra energy compared to the previous method because it can reduce the transitions on the enable lines of the pipeline latches. These lines are heavily capacitive, as the number of bits latched between pipeline stages can be very high. About 10% of the processor core's total energy consumption is spent switching them[4].

The drawback of this method is that glitches can propagate through more logic gates, now that they are not blocked by the pipeline latches. Unfortunately the effect of glitches cannot be quantified before the full processor is available in layout, so their contribution cannot be evaluated accurately at this stage.

5. Measurements

The pipeline depth control methods were evaluated by simulating a mixed behavioural/structural Verilog model of the processor, using NC-Verilog. A custom VCL-PLI module was linked with the simulator to count the toggles of a large number of nodes. The number of toggles was then multiplied by the capacitance of each node, including both parasitic and transistor gate/drain capacitances, which were extracted from the original processor. As the modifications made to the base processor for both pipeline control methods are very minor, their own effect on the energy consumption is ignored in the measurements.

The benchmarks used comprise of Dhrystone, a FIR filter taken out of an implementation of GSM encoding, a DES encryption and two SPECInt95 programs: *compress* and *ijpeg*. All are written in C and compiled with speed optimisations enabled, using the compiler provided with *ARM-tools 2.51*. The data input sizes for all benchmarks were small, so that the simulations could complete in reasonable time.

In addition to the usual speed, power, and energy metrics, the notion of "extra work" defined by Manne *et al.* [7], is used. This shows how many more instructions are processed in each pipeline stage than those that are finally executed.

Figure 3 shows the *normalised* — relative to the fully-occupied configuration — power, energy and execution time for each benchmark and for each pipeline occupancy, with branch prediction disabled. The energy results show how much extra power is saved compared to simply slowing the execution rate without any architecture adaptation. Figure 4 shows the extra work per pipeline stage and occupancy for each benchmark. It is clear that only when the pipeline occupancy is low there are reasonable energy savings. DES encryption has a few branches, so negligible energy is saved. GSM filter was optimised by the compiler so that many branches were replaced by conditional instructions, resulting in a lot of "extra work" at the execute stage. The other benchmarks have a significant amount of extra work, which was successfully removed by lowering the pipeline occupancy and this is reflected in the energy graphs.

	Power	Exec. time	Energy
Dhrystone	0.40	2.1	0.84
DES encrypt	0.30	3.1	0.96
GSM filter	0.35	2.7	0.92
compress	0.38	2.3	0.86
ijpeg	0.37	2.4	0.88

Table 1. Fully-collapsed pipeline results

Table 1 shows the normalised power, energy and execution time results for each benchmark with a pipeline occupancy of 1, using the collapsible latch controllers. The results show lower energy consumption but increased execution time, compared to the token based version with the same occupancy. The extra work metric for this case is the same as with the token based method, because the pipeline behaviour is the same for both cases.

5.1. Discussion

To our knowledge, there are no other reported microarchitecture adaptation techniques for single issue processors. The techniques closest to those presented here are I-cache throttling [8] and I-cache toggling [3]. They are both designed for superscalar processors and target thermal management; as they achieve this by managing power consumption, they are relevant to this work. In effect they control the transfer of instructions from the I-cache to the processor, either by restricting the width (throttling) or the frequency (toggling). Compared to the methods presented here, they do not turn off parts of the pipeline (such as the branch predictor) and they cannot merge all the pipeline stages into one to save energy on the latch enable signals.

Other related methods applied to superscalar processors are pipeline gating [7], pipeline balancing [1] and instruction flow-based throttling [2]. All these methods strive not to compromise performance because they target high-performance processors. The work presented here could be considered as an extension of those methods in low-performance, embedded processors. It would be interesting to see if the pipeline depth control techniques presented here can be effectively combined with speculation control in a superscalar processor, so that the processor can adapt from very low-power, low-performance to high-performance using only architectural techniques.

Thus far, the pipeline depth has been kept constant during the execution. We believe that some of the performance lost, can be recovered by applying the methods dynamically, keeping the pipeline relatively deep only for the tight loops.

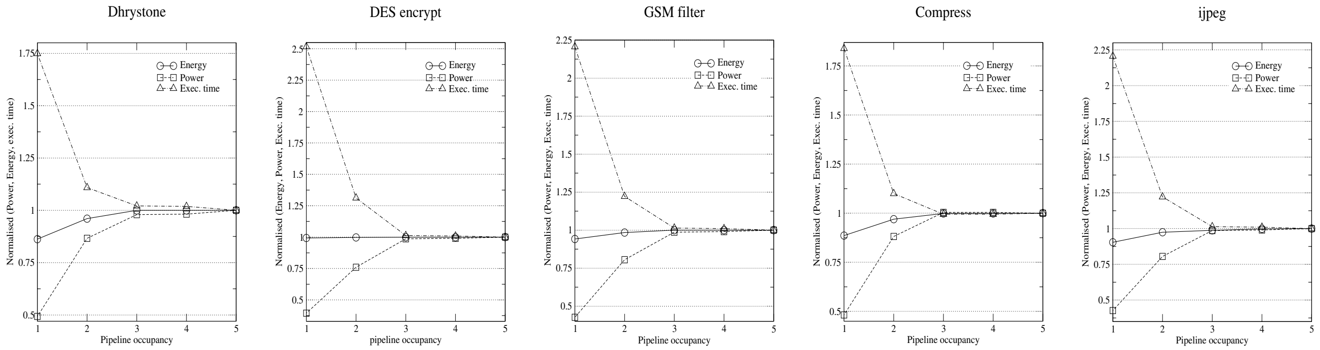

Figure 3. Results (power, energy, exec. time) of token based pipeline occupancy control

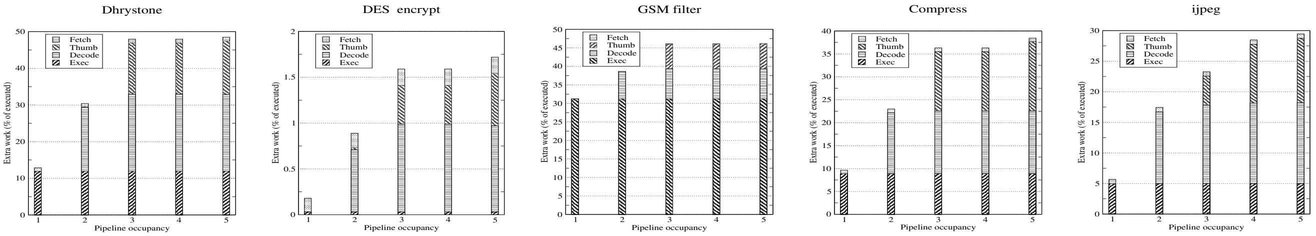

Figure 4. Results (extra work) of token based pipeline occupancy control

6. Conclusions

A novel method of managing the power consumption of a single-issue processor, by controlling its pipeline depth, is proposed. Decreasing the pipeline depth slows down the processor but also saves energy because fewer 'speculative' instructions are fetched and decoded. In addition, hardware that supports pipelining, such as branch prediction, can be turned off as it is not necessary in shallow pipelines. The power saved with this method is more than would be saved by merely slowing down the processor to achieve an equivalent performance, because of the lower switched capacitance per cycle.

Two techniques were presented to control the pipeline depth, using an asynchronous design style because of its inherent flexibility. One technique uses token passing as a method to control the pipeline occupancy, and thus indirectly, the pipeline depth. The other enables the selective and dynamic merging of adjacent pipeline stages by making the pipeline latches between them 'permanently' transparent. Simulating processor models employing these techniques shows an energy reduction of up to 16% compared to the base system, using a collection of five benchmarks.

Acknowledgements

Thanks to the reviewers for helping to improve the paper with their useful comments. A. Efthymiou is supported by the Department of Computer Science, University of Manchester. This support is gratefully appreciated.

References

[1] R. Bahar, S. Manne. Power and energy reduction via pipeline balancing. In *Proc. ISCA'01*, pages 218–229. June 2001.

[2] A. Baniasadi, A. Moshovos. Instruction flow-based front-end throttling for power-aware high-performance processors. In *Proc. ISLPED'01*, pages 16–21. Aug. 2001.

[3] D. Brooks, M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proc. HPCA-7*, pages 171–182. Jan. 2001.

[4] A. Efthymiou, J. Garside. A comparative power analysis of an asynchronous processor. In *Proc. PATMOS'01*, Sept. 2001.

[5] J. Garside, S. Furber, S. Chung. AMULET3 revealed. In *Proc. ASYNC'99*, pages 51–59, Apr. 1999.

[6] A. Iyer, D. Marculescu. Power aware microarchitecture resource scaling. In *Proc. DATE'01*, pages 190–196, Mar. 2001.

[7] S. Manne, A. Klauser, D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proc. ISCA'98*, pages 132–141. June 1998.

[8] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, J. Alvarez. Thermal management system for high performance PowerPC microprocessors. In *Proc. IEEE COMPCON'97*, pages 325–330. Feb. 1997.

[9] J. Sparsø, S. Furber, editors. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, 2001.