Asymmetric-Frequency Clustering: A Power-Aware Back-End for High-Performance Processors

Amirali Baniasadi
Electrical and Computer Engineering
Northwestern University
amirali@ece.northwestern.edu

Andreas Moshovos
Electrical and Computer Engineering
University of Toronto
moshovos@eecg.toronto.edu

ABSTRACT

We introduce asymmetric frequency clustering (AFC), a micro-architectural technique that reduces the dynamic power dissipated by a processor's back-end while maintaining high performance. We present a dual-cluster, dual-frequency machine comprising a performance oriented cluster and a power-aware one. The power-aware cluster operates at half the frequency of the performance oriented cluster and uses a lower voltage supply. We show that this organization significantly reduces back-end power dissipation by executing non-performance-critical instructions in the power-aware cluster. AFC localizes the two frequency/voltage domains. Consequently, it mitigates many of the complexities associated with maintaining multiple supply voltage and frequency domains on the same chip. Key to the success of this technique are methods that assign as many instructions as possible to the slower/ lower power cluster without impacting overall performance. We evaluate our techniques using a subset of SPEC2000 and SPEC95. AFC provides a 16% back-end power reduction with 1.5% performance loss compared to a conventional, dual-clustered processor where each cluster has schedulers of the same width and length.

Categories and Subject Descriptors

C.1.1 [Single Data Stream Architectures] Pipeline processors.

General Terms

Design

Keywords

Power-Aware Architectures, Processor Back-End, Instruction Criticality, Assymetric Frequency Clustering, High-Performance Processors.

1. INTRODUCTION

Frequency and *voltage scaling* are two commonly used circuit-level techniques that offer a trade off between latency and switching (dynamic) power. These techniques have been used extensively to reduce power in the non-critical circuit paths of modern high-performance processors. Further power benefits may be possible when these circuit-level techniques are combined with architectural-level methods. The key idea is to use lower power and consequently performance units for executing *non-critical instructions*. These are instructions whose execution time can be prolonged without impacting performance. There are two main challenges: (1) Developing power-aware heuristics that can identify non-critical instructions with sufficient accuracy and little power overhead, and (2) building pipelines that successfully mix different supply voltage and frequency domains.

In this work, we focus on the processor's *back-end (i.e*., instruction issue, complete and commit). There are two reasons: First, a large fraction of power in high-performance processors is dissipated by their back-end. Second, most of the front-end power is dissipated by SRAM-like structures where a reduction in supply voltage may compromise cell stability and correct operation.

We introduce *asymmetric-frequency clustering* (AFC) as a power-aware back-end that leverages frequency and supply voltage scaling for reducing power dissipation. We study an AFC processor comprising two clusters: one that is a *performance* oriented conventional cluster and another *low-power* oriented cluster. The fast cluster operates at twice the frequency that the slow cluster operates at. This facilitates the use of a lower supply voltage at the slower cluster resulting in further power benefits. We argue that AFC simplifies the design of dual-frequency, dual-voltage processors as it *localizes* the two frequency/voltage domains into two *coarse grain* blocks defining a clear architectural and physical interface between them. Moreover, the AFC maintains the frequency advantage of clustering [8]. Key to the success of AFC is the ability to predict those non-critical instructions that can be executed twice as slow without reducing performance. We present three heuristics for doing so that are based on local information. One of the heuristics is a modification of a previously proposed criticality predictors [7].

Our contributions are: (1) We introduce new, simple to implement and effective non-criticality detection heuristics, and (2) we demonstrate that when these heuristics are used with AFC they can significantly reduce power while maintaining high performance.

The rest of the paper is organized as follows: In section 2, we present the rationale of our approach and discuss various heuristics for determining non-critical instructions. In section 3, we commend on related work. In section 4, we present our methodology. In section 5, we report performance and power results. Finally in section 6, we summarize our findings and offer concluding remarks.

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

2. ASYMMETRIC-FREQUENCY CLUSTERING

The goal of AFC is to use units operating at a lower voltage supply and frequency *intelligently* only for those instructions that are non-critical. Previous work has shown that there are many instructions that could be delayed without impacting performance [5]. Processors that use dynamic voltage/frequency scaling exist. However, adjustments to voltage/frequency are done at a coarse grain (e.g., thousands of cycles) since the circuits that facilitate this scaling incur considerable delay and power overheads that have to be amortized over long periods of time. AFC aims at providing the flexibility to use lower supply voltage and frequency units on a *per instruction basis*. In AFC, the processor's resources are *statically* divided into frequency/voltage domains at a coarse level. Switching power can be reduced by *intelligently* distributing instructions to the different voltage/frequency domains without requiring dynamic voltage/frequency scaling.

In general, mixing multiple voltages and frequencies on the same circuit is a challenging task.Different power supply lines are required, the design of the clock distribution network becomes more complex and noise considerations increase. Of particular concern is efficient interfacing between resources operating under different voltages and frequencies. For example, driving a higher supply voltage circuit using the output of a lower supply voltage circuit requires conversion of the voltage levels. Such circuits impose area overheads, can be slow and may dissipate relatively high power since the lower voltage levels may not be sufficient to completely turn-off transistors operating at a higher supply voltage. Finally, using a lower voltage supply is highly questionable in some cases. This includes the various SRAM-based structures (e.g., caches and branch predictors).

In this work, we argue that clustering [8] can be used to *minimize* the complexities associated with designing and interfacing circuits operating at different frequencies and supply voltages. To illustrate the potential of this method, we *statically* divide the processor's back-end into two different clusters. Each cluster operates under a different frequency/supply voltage domain. We devote one cluster to power efficient resources (low voltage and frequency) while the other cluster uses performance oriented resources (high voltage and frequency). We refer to such clustering as *frequency-clustering*. With frequency-clustering, we restrict interfacing to cluster boundaries. In AFC, voltage shifting is limited to a set of clearly identified buses that connect the two clusters. These buses are used mostly for inter-cluster communication. In addition, by localizing both frequency and voltage domain we reduce the associated voltage/frequency distribution network.

2.1. AFC Architecture

Figure 1 shows our AFC processor. The processor's back-end comprises a 4-way performance-oriented cluster (FASTC) and a 2-way power-aware cluster (SLOWC). FASTC runs at a frequency that is twice faster than that of SLOWC. FASTC has a 64-entry window and can issue up to four instructions per cycle. SLOWC has a window size of 32 instructions that can issue up to two instructions per cycle. All memory references are performed in the fast cluster. This is done, so that we do not worry about caches operating at different voltages Inter-cluster communication latency is at least two cycles.

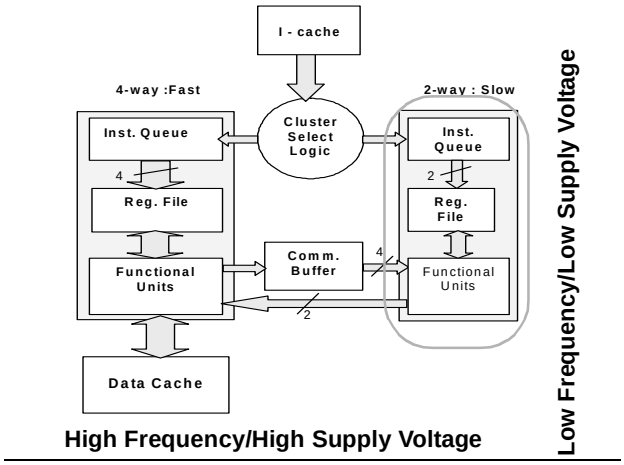

Figure 1: An asymmetric-frequency, dual-cluster processor.

FASTC has six writeback ports since it may receive up to four results from itself and two from SLOWC. Since SLOWC is twice as slow, inter-cluster communication is limited to even cycles. FASTC can receive up to four results during odd cycles and up to six during even cycles. SLOWC receives results only on even cycles. Since FASTC may be producing results during odd cycles too, a FIFO buffer is provided between the two clusters. This *inter-cluster communication buffer* also serves to minimize the number of writeback ports in SLOWC as follows: In the worst case, SLOWC may need to receive up to 10 (8 from FASTC, 2 from SLOWC) results in a single cycle. That would require a total of 10 writeback ports at SLOWC. We found that doing so greatly increases power at SLOWC. In order to reduce the number of write ports and hence the power dissipation, we limit the number of results that can be simultaneously received by SLOWC to six. The inter-cluster communication buffer serves to smooth out communication. In our studies, we used a 16-entry buffer. If this buffer is full we stall FASTC until the SLOWC frees up four entries.

2.2. Cluster Assignment Heuristics

The goal of the cluster assignment heuristics is to identify those non-critical instructions than can execute twice as slow. We present a number of heuristics that utilize local information only. Since our goal is to reduce power consumption, in the evaluation we take into account the power dissipated by the auxiliary structures required by the heuristics. With all heuristics, we initially assign all instructions to FASTC. We assign instructions to SLOWC when instructed by the heuristic. Our goal is not to develop the best heuristic possible. Rather, we aim at demonstrating that even simple heuristics can result in significant power improvements. We have experimented with various heuristics and present the best ones here.

Generation-Time Gap (GTG): This heuristic is a combination of two separate heuristics. The GTG heuristic deems an instruction as non-critical if *any* of the two underlying heuristics does so. The first, looks at the gap in cycles between the time an instruction writes its result and the time its children, if any, issue. The intuition here is that if an instruction writebacks a

result and its children do not issue immediately, then they are probably waiting for some other instruction to finish. Since it takes 2 cycle to communicate results across clusters and since SLOWC is twice as slow we use four cycles as our threshold (two cycles for executing most instructions, plus two for inter-cluster communication). The second heuristic, uses an approximation of dynamic instruction distance between an instruction and its children. If an instruction is dispatched much earlier than its children, this is an indication that it appears much earlier in the instruction stream. Empirically, we found this to be a good indicator that the parent can be delayed. In this study we used two cycles as the threshold. That is, if no children are dispatched in the next two cycles after an instruction is dispatched the latter is deemed as non-critical.

In both cases, the determination that an instruction is non-critical is available after the instruction was dispatched and assigned to a cluster. Accordingly, we use a table to predict if future instances of the same instruction are non-critical. In our experiments we used a 4k table. Every entry contains six bits. These are used as a six-bit saturating counter. Entries are allocated only for critical instructions as they commit. If an entry exists, it is also updated at commit time. The entry is initially set to 32. When we detect critical instructions we increment the counter by eight, otherwise we decrement by one. An instruction is deemed non-critical by accessing the table at dispatch and if the entry found is *below or equal* a threshold (16 in our case).

Young in Queue (YIQ): This method is based on the QOLD criterion proposed by Tune *et al.*,[7]. The QOLD method marks the oldest instruction in the instruction queue as critical during each cycle. Once the oldest instruction is marked as critical, we also mark its parents. Upon commit, we update a prediction table as we did with the GTG method. This method also uses a prediction table similarly to the GTG method. Upon dispatch an instruction is deemed critical if: (1) the prediction table contains an entry that exceeds the pre-specified threshold, (2) the prediction table contains no entry, and (3) if the instruction's parents are marked as critical.

Complete Time Estimation (CTE): If we had an oracle and we knew in advance when an instruction will complete, then a heuristic for identifying non-critical instructions during dispatch would be as follows: Compare your complete time with the *maximum* complete time of all preceding instructions that are currently in the window. If this instruction will be completing earlier, it will be forced to wait for that other instruction to commit. Hence, it may be safe to delay this instruction. Of course, we cannot have such an oracle. Instead, we estimate the completion time of instructions as they are dispatched. This is straightforward: We associate a completion time with every register. As instructions are dispatched we predict how long they will take to execute once issued (that is, we "predict" the latency of the functional unit they will be executing). We also obtain the maximum completion time for its source operands. Adding the two we obtain an estimate for when this instruction will complete. We then compare this estimate with the maximum completion time seen thus far (only one entry is required). Modern processors already predict instruction latencies for dynamic scheduling purposes.

3. PRIOR WORK

Seng *et al*. [2], suggested exploiting slower functional units for processing non-critical instructions. Our measurements show that in addition to functional units, other back-end sub-sections consume a considerable part of processor back-end power. Therefore, we extend prior work to cover the processor back-end (*e.g.*, instruction selection and wake-up). G. Moshnyaga suggested a complexity adaptive issue logic where voltage supply was changed dynamically [3]. We extend the voltage duality to the entire back-end and remove supply voltage switching and its associated costs by using clusters. Pyreddy and Tyson [4] studied how exploiting dual speed pipelines affects processor performance. They used heuristics to mark and send instructions through execution paths with varying latencies. Their heuristics used profiling methods for marking instructions, while we focus on dynamic mechanisms. In addition we take into account voltage reduction effects. Casmira and Grunwald [5] defined and measured *slack* as the number of cycles than an instruction can wait before being issued and becoming critical. Semeraro also suggested using multiple frequency and voltage domains throughout the processor and showed that fine grain voltage/frequency scaling this can lead to significant power benefits [9]. In our work, we argue that clustering can alleviate the complexities and potential power overheads associated with fine-grain mixing of frequency/voltage domains.

4. METHODOLOGY

We used benchmarks from the SPEC'2K and SPEC95 suite. The benchmarks were compiled for the MIPS-like architecture used by the Simplescalar v3.0 simulation tool set. We used GNU's gcc compiler (flags: -O2 –funroll-loops –finline-functions). We simulated 2 Billion of the instructions after skipping the initialization. The main architectural parameters of our processor model are shown in table 2. We used WATTCH [1] for power estimation. We modeled an aggressive 2GHz (the slow cluster operates at 1GHz) superscalar microarchitecture manufactured under a 0.1micron technology. To estimate the relevant process parameters, we used the process scaling methodology developed for CACTI [8] and that is incorporated in WATTCH.

Table 1: Base configuration details.

Branch Predictor	32K GShare, 32K bi-modal, 32K selector
Schedulers	fast cluster: 64 entries slow cluster: 32 entries, RUU-like
Fetch Unit	Up to 6 instr. per cycle. Max 2 branches per cycle 64-entry Fetch Buffer
Load/Store Queue	64 entries, 3 loads or stores per cycle Perfect disambiguation
Decode width	any 6 instructions / cycle
Issue, Commit width	fast cluster: any 4 insts / cycle slow cluster: any 2 insts / cycle
Func. Unit Latencies	same as MIPS R10000
L1 - Instr. /Data Caches	64K 4-way SA, 32B blocks, 3 cycle hits
Unified L2	256K 4-way SA, 64B blocks, 16-cycle hits
Main Memory	Infinite, 100 cycles

5. RESULTS

In section 5.1, we study the performance of our power-aware dual-cluster organization. We will refer to our organization as AFC. In section 5.2 we study power

5.1. Performance

Overall, our dual cluster architecture has a 96-entry window and can issue 6 instructions per cycle. Accordingly, we compare with an identically organized, dual-cluster configuration where both clusters operate at the fast frequency. We refer to this configuration as DUAL-UF.

In figure 2(a) we report performance compared to the DUAL-UF architecture. To distribute instructions to clusters, we start with the *dependence* method [8]. In AFC, the distribution decisions are selectively overwritten by the criticality prediction mechanism. On average, AFC's performance is comparable to DUAL-UF. Average performance slowdown is 1.5%. 1.9% and 1.6% for CTE, GTG and YIQ respectively. To better understand the relative importance of clustering and the use of a slow cluster, in figure 2(b) we report the percentage of instructions assigned to the *fast-cluster*. On average, GTG assigns the minimum number of instructions to the fast cluster (81%). YIQ assigns the maximum number of instructions to the fast cluster (93%). Finally, CTE assigns 86% of the instructions to the fast cluster.

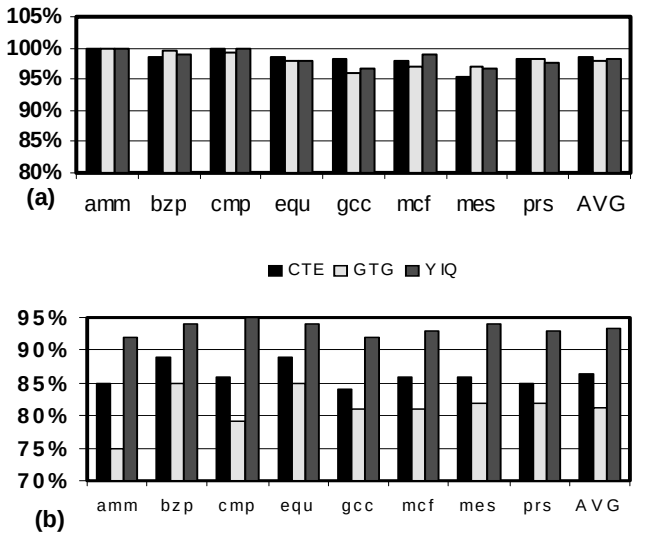

Figure 2: (a) Relative performance compared to DUAL-UF (b) The fraction of instructions assigned to the fast cluster.

5.2. Power

Chandraskan *et al.*, have shown that even though the exact determination of voltage reduction under performance constraints is complex and technology specific, it is possible to closely predict it [6]. Their methods predicts that when the frequency is twice as slow, the supply voltage can be reduced even to less than 50%. Accordingly, in figure 3 we report power savings when the slow cluster operates with a voltage supply that is 70% of that used in the fast cluster. We use a 0.7x factor as opposed to 0.5x to pessimistically account for voltage conversion latency overheads. Figure 3 reports power savings when compared to a DUAL-UF machine. Average power reduction is 16%, 15% and 10% for CTE, GTG and YIQ.

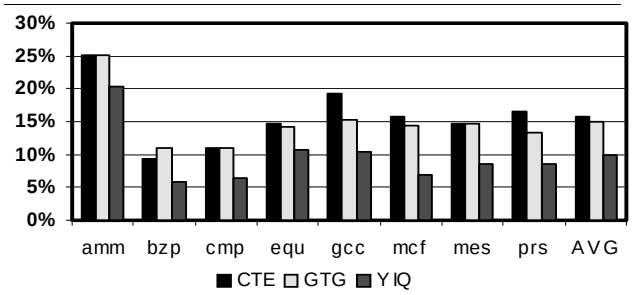

Figure 3: Back-end power reduction relative to a DUAL-UF

6. CONCLUSION

We introduced and evaluated AFC as an asymmetric dual-cluster, dual-frequency microarchitecture comprising a performance oriented cluster and a power-aware one. AFC aims: (1) at reducing switching power by executing non-critical instructions slower, and (2) at maintaining performance by executing performance critical instructions as fast as possible. In AFC non-critical instructions are meant to execute in the power-aware cluster that is narrow and uses a lower frequency and power supply. Performance critical instructions are meant to execute in the performance oriented cluster that is wide and uses a higher frequency and voltage supply. By localizing the two frequency/voltage domains, we mitigate many of the complexities associated with maintaining multiple supply voltage and frequency domains on the same chip. Essential to the success of our technique are methods for distributing instructions across the two clusters. By exploiting our suggested methods, we save up to 16% of back-end power with only a 1.5% performance loss compared to a conventional, dual-clustered processor.

REFERENCES

[1] D. Brooks, V. Tiwari M. Martonosi "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations". In *Proc of the 27th Int'l Symp. on Computer Architecture*, 2000

[2] John S. Seng, Eric S. Tune, Dean M. Tullsen, "Reducing Power with Dynamic Critical Path Information", In *Proc. 34th Annual International Symposium on Microarchitecture*, December, 2001.

[3] Vasily G. Moshnyaga, "Reducing Energy Dissipation of Complexity Adaptive Issue Queue by Dual Voltage Supply", *2001 Workshop on Complexity-Effective Desig*n, June 2001

[4] R. Pyreddy and G. Tyson. Evaluating design tradeoffs in dual speed pipelines. *Workshop on Complexity-Effective Desig*n, June 2001

[5] J. Casmira and D. Grunwald. Dynamic instruction scheduling slack. In *2000 KoolChips worksho*p, Dec. 2000.

[6] Anantha P. Chandraskan, Samuel Sheng, and Robert W. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits, Vol. 27, No. 4. April, 1998.*

[7] E. Tune, D. Liang, D. Tullsen, and B. Calder. Dynamic prediction of critical path instructions. In *Proceedings of* the Seventh International Symposium on High-Performance Computer Architecture, Feb. 2001

[8] S. Palacharla, N. P. Jouppi, and J. E. Smith. *Complexity-effective superscalar processors.* In *Proc. International Symposium on Computer Architecture-24*, June 1997.

[9] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M. L. Scott, *Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling*, In *Proceedings of* the 8th. Intl. Symposium on High-Performance Architecture, Feb. 2002