Dynamic Register File Resizing and Frequency Scaling to Improve Embedded Processor Performance and Energy-Delay Efficiency

Houman Homayoun[†], Sudeep Pasricha[†], Mohammad Makhzan[‡], Alex Veidenbaum[†]

[†]Center for Embedded Computer Systems
University of California, Irvine, CA
{hhomayou, sudeep, alexv}@ics.uci.edu

[‡]Department of Electrical and Computer Engineering
University of California, Irvine, CA
mmakhzan@uci.edu

ABSTRACT

With CMOS scaling leading to ever increasing levels of transistor integration on a chip, designers of high-performance embedded processors have ample area available to increase processor resources in order to improve performance. However, increasing resource sizes can increase power dissipation and also reduce access time, which can limit maximum achievable operating frequency. In this paper, we explore optimizations for the processor register file (*RF*), to improve performance and reduce the energy-delay product. We show that while increasing the size of the *RF* can potentially increase the IPC, overall it results in an increase in program execution time. In response we propose *L2MRFS* – a dynamic register file resizing scheme in tandem with frequency scaling, which exploits L2 cache misses to noticeably improve processor performance (11% on average) and also significantly reduce the energy-delay product (7%).

Categories and Subject Descriptors: C.1.1 [Processor Architectures]: Single Data Stream Architectures; C.4 Performance of Systems

General Terms: Performance, Design

Keywords: Performance, Embedded Processor, Register File, Dynamic Resizing

1. INTRODUCTION

With the advent of the digital convergence era, high performance embedded applications are increasingly using complex out-of-order superscalar embedded processors to meet performance goals. Examples of such embedded processors include the NEC's VR5500 and VR77100 Star Sapphire [6], and the IBM PowerPC 750FX [7] processors. Technology scaling into the ultra deep submicron (UDSM) region has allowed hundreds of millions of gates to be integrated onto a single chip. Designers thus have ample silicon budget to add more processor resources (e.g., increasing register file size, reorder buffer size, etc) in order to exploit application parallelism and improve performance. However, restrictions with the power budget and practically achievable operating clock frequencies act as limiting factors that prevent unbounded increases in processor resource sizes. Increasing register file (*RF*) size for instance, increases its access time, which reduces processor frequency, since access times to the multi-ported *RF* is one of the most critical timing factors that determines the achievable processor operating frequency.

In this paper, we present a novel technique for dynamically resizing the *RF* (*L2MRFS*) which in tandem with dynamic frequency scaling (*DFS*) significantly improves the performance and reduces energy-delay product for embedded processors. Our approach exploits L2 cache misses, adaptively reducing *RF* size during the period when there is no pending L2 cache miss, and using a larger *RF* during the L2 cache miss period. To enable single cycle access during *RF* resizing, the processor frequency is dynamically scaled. Such a dynamic frequency scaling (*DFS*) should be done fast, otherwise it can negatively impact performance during dynamic *RF* resizing. In our studied processor, IBM PowerPC 750FX, *DFS* is done in effectively zero cycle using a dual PLL architecture [11]. The *RF* size adaptation is realized using a circuit modification scheme that comes with minimal hardware modification, unlike costly banking or clustering techniques. Experimental results show that *L2MRFS* noticeably improves performance and also significantly reduces energy-delay product for out-of-order embedded processers.

DAC 2008, June 8–13, 2008, Anaheim, California, USA

2. RELATED WORK

There has been a lot of research that has proposed altering the structure of the register file (RF) for improving performance. One set of techniques uses localities of communication to split the microarchitecture into distributed clusters, each containing a subset of the RF. [1]-[2]. A critical issue in the design of such systems is the heuristics used to map instructions to clusters. These schemes have the potential to scale to larger issue widths but require complex inter-cluster control logic to map instructions to clusters and to handle inter-cluster dependencies.

Alternatively, other approaches retain a centralized microarchitecture, but partition the processor units such as the *RF* [3]-[4] to reduce access time and energy dissipation. Partitioning the *RF* into multiple banks for instance reduces the ports on the partitions, which reduces the pre-charging and sensing times and the related energy dissipation. But these reductions come at the cost of value multiplexing and port conflict problems. The major drawback of all these banking techniques in general is in the complexity that speculation adds and more specifically the complexity they introduce on handling the coherency in register caches and banking conflicts. This added complexity becomes even more critical for embedded processors that work in resource-constrained environments.

None of the above schemes exploit the L2 cache misses for dynamic register file resizing. The technique that comes closest to our work is [5], which performs early register de-allocation based on L2 misses to improve performance. However, such a scheme increases complexity since it requires additional resources such as bit vectors to identify the sources and the destinations of the load-dependent instructions, additional bits in the ROB and possibly a backup register file to store de-allocated values. In contrast, we propose a much simpler architecture and circuit level approach that dynamically adapts the RF size on L2 cache misses to achieve significant performance improvements.

3. MOTIVATION FOR INCREASING RF SIZE

Consider the case where during processor execution, a long latency L2 cache miss occurs. The processor executes some independent instructions but eventually ends up becoming stalled [12]. When a cache miss occurs, the load instruction that caused the L2 miss stays on top of the reorder buffer (*ROB*) and doesn't allow the subsequent instructions to be committed. As a result, the subsequent instructions occupying the *ROB* and *RF* cannot be released until the miss returns. This gradually increases *ROB* and *RF* occupancy, and reduces the processor issue rate. Due to the long service time, either the *ROB* or *RF* can completely fills up with subsequent instructions and the processor ends up being stalled until the miss is serviced. The same scenario occurs for the load queue (*LQ*), store queue (*SQ*) and instruction queue (*IQ*) – the subsequent dependent instructions to the load with a miss cannot be issued due to data dependency. Such instructions reside in the *IQ* until the miss returns.

Given the long cache miss service time, the above scenario can happen quite frequently. To estimate the frequency of stalls due to L2 cache misses, we explored the PowerPC 750FX architecture [7] with a separate IL1 (level 1 instruction cache) and DL1 (level 1 data cache) of 32KB with access time of 2 cycles and a unified L2 (level 2 cache) of size 256KB with an access time of 12 cycles. The size of *ROB*, *IQ* and *RF* was kept as 16, 8 and 24 respectively, and the miss penalty of accessing the main memory was 60 cycles.

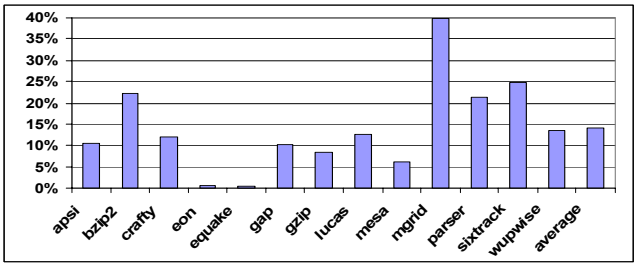

Figure 1. Relative processor stall period due to L2 cache miss

In Figure 1 we report how often the entire processor is stalled due to L2 cache misses. As the results indicate, the entire processor pipelines stalls by up to 40% of the program execution time. The average is almost 15%. The large idle period reported in Figure 1 impacts the processor performance significantly. One simple solution to reduce the occurrence of such an idle period is to increase the size of processor resources such as the *RF*, *IQ*, *ROB* and etc. With larger resources it is less likely that these resources will fill up completely during the L2 cache miss service time and potentially improve performance. It should be noted that the sizes of these resources have to be scaled up together; otherwise the non-scaled ones would become a performance bottleneck.

4. IMPACT OF INCREASING RF SIZE

While increasing the size of *RF*, (as well as *ROB*, *LQ* and *IQ*) can potentially increase processor performance by reducing the occurrences of idle periods, it has a critical impact on the achievable processor operating frequency. This is specifically the case for the *RF*, the size of which is a limiting timing factor that determines the maximum operating frequency of a processor as discussed in several other works [1] [3] [5]. Increasing the size of the *RF* increases its access time. This is mostly due to an increase in the length of the bitline. Figure 2 shows the delay breakdown among the various components of the *RF*, for 24, 32 and 48 entry *RF* configurations. Results are shown for a single read operation, with delays calculated using a modified version of CACTI4 [8]. A clear trend seen in this figure is the significant increase in bitline delay when the size of the *RF* increases. This can be explained as follows. The signal propagation delay of the bitline is relative to its equivalent capacitance. The equivalent capacitance on the bitline is C_{eq} = *N* * *diffusion capacitance of pass transistors + wire capacitance* (usually 10% of total diffusion capacitance) where *N* is the total number of rows. As the number of rows increases the equivalent bitline capacitance also increases. Since the propagation delay on the bitline is relative to RC_{eq}, the propagation delay approximately increases with the number of rows. The propagation delay for the remaining RF components increases only slightly with an increase in RF size, as shown in the figure.

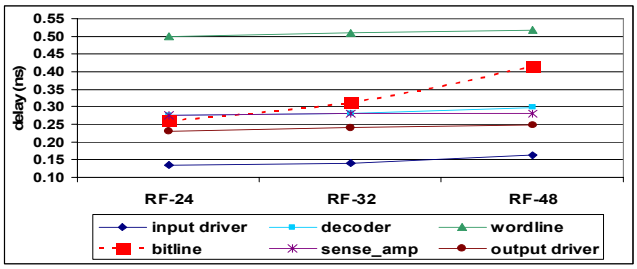

Figure 2. Breakdown of RF component delay with increasing size

It is thus clear that increasing *RF* size increases its access time mostly due to increase in the bitline delay. A similar increase in access time occurs for the *ROB* and *IQ*. As a result, the achievable operating frequency of the processor is reduced when resource sizes are increased. Note that it is possible to apply banking or clustering techniques to improve RF access time, as has been proposed for high performance processors [3]-[4]. However banking or clustering the RF is a costly solution, due to the significant complexity that is introduce in handling banking conflicts and coherency in RF banks. Such complexity can be prohibitive, especially in the resource constrained environments in which embedded processors operate.

Table 1. Reduction in clock freq with increasing resource size

Processor Configuration	Baseline	Conf_1	Conf_2
RF size	**24**	**32**	**48**
ROB size	16	24	32
IQ size	8	12	24
RF access time (ns)	1.67	1.76	1.92
Operating Freq (MHz)	595	568	520

4.1 Static Register File Sizing

In order to study the impact of statically increasing the size of the RF (and also ROB and IQ), we consider three different processor configurations, as shown in Table 1. The baseline configuration shown in the table has a 595 MHz operating frequency. *Conf_1* represents an intermediate configuration, with the *RF*, ROB and IQ upsized to 32, 24 and 12 entries, respectively. Using a modified version of CACTI4 [8], the access time for RF is found to increase to 1.76 ns. As a result, the operating clock frequency for the processor cannot exceed 570 MHz for this configuration. The *Conf_2* represents a configuration in which resources are upsized aggressively and the achievable operating clock frequency is reduced further to 520 MHz.

Figure 3 shows the performance in terms of IPC for the different configurations described above (normalized to the baseline configuration), while operating at their maximum achievable operating frequency. Figure 4 provides more insight on the relative idle period processor stalls due to L2 cache misses for the different configurations.

From Figure 3 it can be seen that in all cases, increasing the size of resources increases the IPC. Also the results in Figure 4 verify that increasing resources can potentially reduce the occurrences of idle time in the processor due to L2 cache misses. However, increasing the resources size, while effective in increasing IPC, causes the processor

to run slower (from 595MHz down to 568 and 520 MHz, as reported in Table 1).

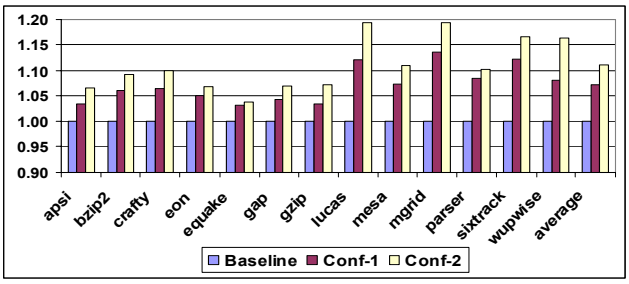

Figure 3. Performance for different processor configuration in terms of IPC normalized to the baseline configuration

Since the three configurations discussed above operate at different clock frequencies, we must take into consideration their maximum achievable operating clock frequency in addition to their IPC for performance comparison. Accordingly, in Figure 5 we report the execution time for the three designs. Interestingly in most benchmarks we observe the execution time increases with larger resource sizes. In other words, for the trade-off between having larger resources (and hence reducing the occurrences of idle period due to L2 cache misses as reported in Figure 4) and lowering the clock frequency, the latter becomes more important and plays a major role in deciding the performance. On average there is almost 0.5% performance degradation for intermediate configuration (*Conf_1*) compared to the baseline. The aggressive configuration (*Conf_2*) results in performance degradation by up to 10% and on average 4%, compared to the baseline.

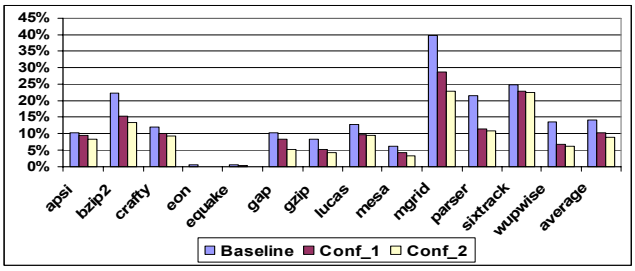

Figure 4. Relative idle period processor stalls due to L2 cache misses for different configurations.

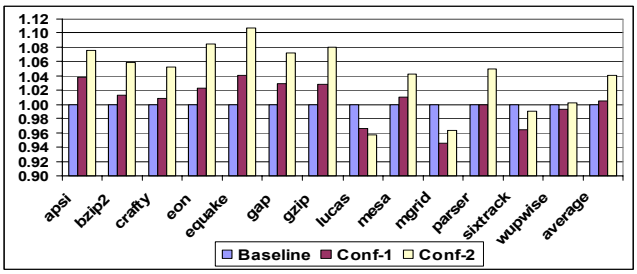

Figure 5. Normalized execution time for different configs with reduced operating frequency compared to baseline architecture

5. DYNAMIC REGISTER FILE RESIZING

Based on the observations made in the previous section, we realize the need for a *dynamic RF scaling* based on L2 cache misses, that allows the processor to use smaller RF (having a lower access time) during the period when there is no pending L2 cache miss (*normal period*) and a larger RF (at the cost of having a higher access time) during the *L2 cache miss period*. Note that the assumption is that the RF size decides the maximum achievable operating clock frequency both in its typical size or when it is being resized. During the *normal period*, RF is kept at its typical size (24 entries) and as such it can operate at the baseline operating frequency (almost at 595 MHz from Table 1). During the *cache miss period* we scale up the RF size, which results in an increase in its access time. Finally, the RF size is returned to its normal size once the L2 cache miss has been serviced and the scaled up part has no more data. We refer to this technique as *L2 miss driven register file scaling* (*L2MRFS*).

Note that to be able to satisfy accessing the RF in one cycle we need to reduce the operating clock frequency when we scale up its size. Such dynamic frequency scaling (*DFS* in brief) needs to be done fast, otherwise it could have a negative impact on the performance benefit of dynamic RF resizing. For this reason we need to use a PLL architecture capable of applying *DFS* with the least transition delay. Our studied architecture, the IBM PowerPC 750FX, has such a PLL architecture. PowerPC 750 [7] [11] uses a dual PLL architecture which allows fast DFS with effectively zero latency (one processor clock cycle is being skipped). With the benefit of such PLL architecture, there is almost no transition overhead associated with changing the operating clock frequency [11] when resizing the *RF*.

It should also noted that this technique has a fairly simple implementation and does not add significant complexity to the embedded processor pipeline, since the scheduler in embedded processors already keeps track of miss load instructions in L2 caches.

5.1 Circuit Modification

In this section we propose circuit modifications to assist the architectural *L2MRFS* technique. As in the previous section, we assume that *RF* size decides the critical path before and after resource resizing. From Table 1, increasing *RF* size results in an access time greater than the baseline *RF* access time. The challenge here is to design the *RF* in such a way that its access time is dynamically being controlled such that at typical size (24 entries) it has the baseline 1.67 ns access delay and at its scaled up size, 32 and 48, it has 1.76 ns and 1.92 ns access delay respectively.

Figure 6 shows our proposed solution which requires a minimal modification to a unified *RF* (unlike more complex banking schemes). Among all *RF* components, the bitline delay increase is responsible for the majority of *RF* access time increase (for instance this is 80% for a 48 entry *RF* compare to 24 entry *RF* as reported in Figure 2). This is due to the fact that bitline delay is decided by its equivalent capacitance which in turn is proportional to the number of *RF* entries (rows).

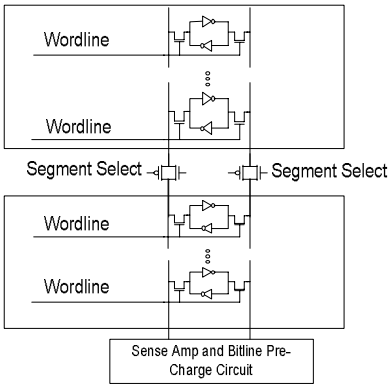

Figure 6. Proposed circuit modification for RF

Accordingly, for the *Conf_2* architecture to be able to achieve an access delay close to the baseline *RF* access delay for a 48 entry *RF* when only 24 of its entries are being used, we need to reduce the equivalent capacitance on the bitline by eliminating the diffusion capacitance of the 24 unused entries. For this purpose, the *RF* is divided into two segments of 24 entries each which are connected through pass transmission gates as shown in Figure 6. This allows the

upper segment bitline to become isolated from the lower segment bitline if the pass gate is off. It should be noted that all other components such as the bitline, sense-amp, etc. are shared for both structures. During the *normal period* the upper segment is power gated and the transmission gate is turned off to isolate the lower bitline segment from the upper bitline segment. Only the lower segment bitline is pre-charged during this period. Since the upper bitline segment is floating, the bitline capacitance during normal period is decided by the lower segment bitline. As such, the bitline delay in this case remain close to the bitline delay of a 24-row register file (the only difference is the delay added by the source capacitance of the pass gate, which is negligible).

In addition, we need to be able to detect when the upper segment is empty (for downsizing at the end of cache miss period when the added segment is empty). To do this, we have augmented the upper segment with one extra bit per entry. This bit is set when an entry (register) is taken and is being reset when the entry is released. By ORing these bits we can detect when the segment is empty. Since the remaining components of *RF* delay change very slightly, the *RF* access delay remains close to that for a 24-row *RF* when the upper segment is isolated. The delay of accessing the *RF* while the upper segment is isolated is 1.70 ns, which is only slightly larger than for a baseline 24-row register file (1.67 ns). When the upper segment is active and the *RF* has its full size, its access requires 1.77 ns for *Conf_1* and 1.93 ns for *Conf_2*.

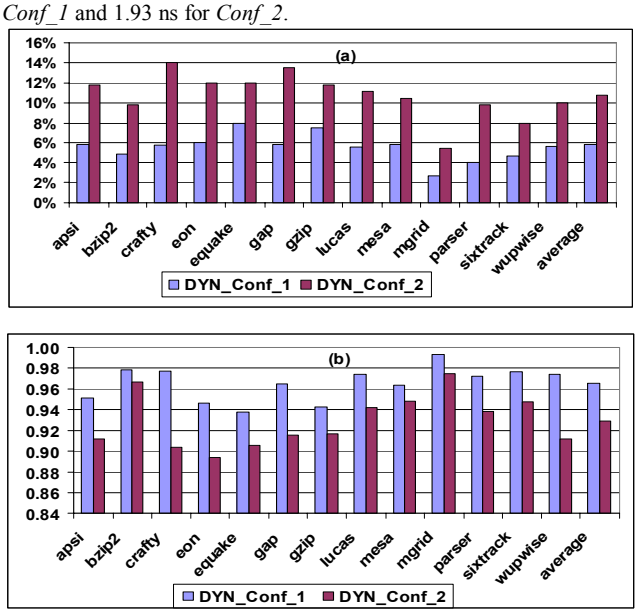

Figure 7. Experimental results: (a) normalized performance improvement for L2MRFS (b) normalized energy-delay product compare to conf_1 and conf_2

6. RESULTS

In this section we present experimental results to show how the dynamic RF resizing approach impacts processor performance. We use SPEC2K benchmarks executed with reference data sets, and compiled with the O4 option using the Compaq compiler. The architecture was simulated using an extensively modified version of MASE (SimpleScalar 4.0) [9]. The benchmarks were fast–forwarded for 500 million instructions, then fully simulated for 1 billion instructions. We used a modified version of CACTI4 [8] for estimating access time for the *RF*. For estimating energy consumption of our adaptive technique we integrate Wattch [10] into our simulator infrastructure. We used process parameters for a 70nm process at 595MHz with 1V supply voltage.

In Figure 7(a) and (b) we report the result of using the proposed *L2MRFS* technique. In Figure 7(a) we report the performance improvement in terms of normalized execution time when we apply *L2MRFS* to *Conf_1* (referred to as *DYN_Conf_1*) compare with *Conf_1* and for *DYN_Conf_2* compare to *Conf_2*. From Figure 7 (a), a performance improvement can be seen across all benchmarks for *DYN_Conf_1* when normalized to *Conf_1*; the performance improvement is almost 6% on average. The performance improvement for *DYN_Conf_2* is even more, up to 14% for *crafty* and 11% on average.

In Figure 7 (b) we report the energy-delay products of *DYN_Conf_1* and *DYN_Conf_2* compared to *Conf_1* and *Conf_2* respectively. As can be seen for all the benchmarks our dynamic RF resizing technique results in overall reduction in energy-delay product. The average energy-delay is reduced by 3.5% and 7% respectively for *DYN_Conf_1* and *DYN_Conf_2* compare to *Conf_1* and *Conf_2*. Finally, it should be noted that while *DYN_conf_2* delivers lower energy-delay and higher performance in terms of execution time, it requires more silicon budget and dissipates more power compared to *DYN_Conf_1* (results are not reported due to space limitation). This in fact is due to having larger resources such *IQ*, *RF* and *ROB*.

7. CONCLUSION

In this paper, we presented a novel technique for dynamically resizing the *RF* (*L2MRFS*) which in tandem with dynamic frequency scaling (*DFS*) significantly improves the performance and reduces energy-delay product for embedded processors. Our approach exploits L2 cache misses, adaptively reducing RF size during the period when there is no pending L2 cache miss, and using a larger RF during the L2 cache miss period. The RF size adaptation is realized using a circuit modification scheme that comes with minimal hardware modification, unlike costly banking or clustering techniques. Our extensive experimental results have shown that *L2MRFS* noticeably improves performance (11% on average) and also significantly reduces energy-delay product (7%) for out-of-order embedded processers.

8. REFERENCES

[1] A. Terechko, M. Garg, H. Corporaal, "Evaluation of speed and area of clustered VLIW processors", VLSI Design, 2005.

[2] O. Ergin, et al., "Increasing Processor Performance through Early Register Release", in ICCD 2004.

[3] J.H. Tseng et al., "Banked Multiported Register Files for High-Frequency Superscalar Microprocessors", ISCA 2003.

[4] R. Balasubramonian, et al. "Reducing the complexity of the register file in dynamic superscalar processors." in MICRO-34, 2001.

[5] J. Sharkey and D. Ponomarev, "An L2-Miss-Driven Early Register Deallocation for SMT Processors", in ICS 2007.

[6] Stijn Eyerman, et al. "Efficient Design Space Exploration of High Performance Embedded Out-of-Order Processors", in DATE 2006.

[7] IBM Corporation. PowerPC 750 RISC Microprocessor Technical Summary. www.ibm.com.

[8] "Cacti4," http://quid.hpl.hp.com:9081/cacti/.

[9] SimpleScalar4 tutorial, SimpleScalar LLC. http://www.simplescalar.com /tutorial.html

[10] D. Brooks, V. Tiwari and M. Martonosi. "Wattch: A framework for architectural-level power analysis and optimizations." in ISCA 2000.

[11] S. Geissler et al., "A low-power RISC microprocessor using dual PLLs in a 0.13/spl mu/m SOI technology with copper interconnect and low-k BEOL dielectric", in ISSCC 2002.

[12] H. Li et al., "VSV: L2-miss-driven variable supply-voltage scaling for low power." in MICRO, 2003.