A First-Order Analysis of Power Overheads of Redundant Multi-Threading

Niti Madan and Rajeev Balasubramonian
School of Computing, University of Utah

***Abstract*— Redundant multi-threading (RMT) has been proposed as an architectural approach that efficiently detects and recovers from soft errors. RMT can impose non-trivial overheads in terms of power consumption. In this paper, we characterize some of the major factors that influence the power consumed by RMT. We outline mechanisms that can reduce this power and derive simple analytical estimates to identify the most promising approach.**

I. Introduction

Soft error rates in microprocessor logic have been projected to increase at an alarming rate [10]. Soft errors can be handled at the process level, circuit level, or at the architecture level. Redundant multi-threading (RMT) has emerged as an efficient mechanism to detect and recover from faults at the architecture level. In RMT, two copies of a thread are executed and results are periodically checked. A number of implementations have been proposed over the last decade [1], [3], [5], [6], [7], [8], [9], [11], [12], [13]. Many of these implementations rely on the processor's ability to support multiple thread contexts (such as in an SMT or CMP processor). A few implementations [1], [11] augment a conventional pipeline with an in-order-like pipeline that redundantly executes every instruction.

Most studies have not focused on the power overheads of RMT. Given that power consumption is already a major design constraint, we expect that power-efficient implementations of RMT will receive much attention from the research community. A first-order estimate of RMT power overheads serves as a useful guideline for such research. In a recent technical report [4], we have carried out a detailed analysis of a number of strategies to reduce RMT power. This paper is an attempt to distill the insight gathered from that study. While [4] relies on detailed simulations, this paper deals exclusively with analytical estimates. This allows us to present data for a wide design space, while occasionally compromising on accuracy.

II. RMT Techniques

We restrict ourselves to RMT techniques that leverage the processor's ability to maintain multiple thread contexts. We assume that the processor is a heterogeneous chip multi-processor (CMP), where each core itself can execute one or more threads in simultaneous multi-threaded (SMT) fashion. For every *primary* thread, a redundant *checker* thread is executed in a different thread context. We assume that the checker thread trails the primary thread by a certain amount of slack. The primary thread is also referred to as the *leading* thread and the checker thread is also referred to as the *trailing* thread. For most of this discussion we assume that a leading thread and its trailing thread execute on different cores within the CMP.

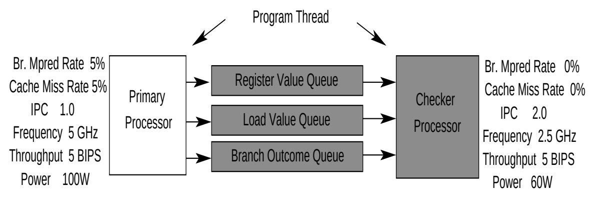

Fig. 1. An example of the effect of scaling the frequency of the checker core.

A. Baseline RMT

In our baseline implementation (similar to that in [3],[6]), the leading thread commits instructions just as a conventional processor would, except that stores are written to a Store Buffer (StB) instead of to the L1 data cache. As shown in Figure 1, committed results are communicated to the trailing thread through a Register Value Queue (RVQ). The results of loads are also sent to the trailing thread (through a Load Value Queue (LVQ)) so that the trailing thread never has to access its data cache (this is necessary for correctness). The trailing thread executes just as a conventional thread, except that (i) instructions are committed only after confirming that the result matches that produced by the leading thread, (ii) store results are forwarded to the leading thread so that the result in the StB can be checked and written to the leading thread's L1 data cache, (iii) load values are acquired from the LVQ. If there is any mis-match in results, the state of the trailing thread is used to initiate recovery. Branch outcomes are also communicated by the leading thread to the trailing thread (via the branch outcome queue (BOQ)), allowing the latter to have perfect branch prediction. If multiple threads are co-scheduled on an SMT core, fetch priorities are adjusted to allow every set of leading and trailing threads to maintain a roughly constant slack.

The fault model is assumed to be the same as in [3], [6]. Storage structures such as caches are ECC-protected. The LVQ and the entire datapath from the cache to the LVQ to the trailing core are also ECC-protected as the trailing thread directly uses these values. The register file need not be ECC protected as a copy exists in the other core. Based on this model, the above RMT implementation is guaranteed to detect and recover from a single event upset in either leading or trailing cores.

B. Power-Efficient RMT

When executing leading and trailing threads, it is desireable that a constant gap (slack) be maintained between both threads. The trailing thread experiences no branch mispredicts or cache misses because of the LVQ and BOQ. The results in the RVQ can also be exploited to implement a perfect value predictor. The trailing thread is therefore capable of high IPC. It may be able to match the throughput of the leading thread even if it throttled back its clock speed or IPC. This provides an opportunity to reduce the power consumed by the trailing thread. We consider three major techniques to exploit the power-performance trade-off.

Dynamic Frequency Scaling (DFS)

Frequency scaling can reduce the dynamic power consumed by the core, but has no effect on leakage power. We will assume that a mechanism exists to select the effective frequency for the trailing thread such that leading and trailing throughputs are matched. Dynamic frequency scaling can also be accompanied by dynamic voltage scaling (DVS). DVS has much higher overheads for every voltage change, but can significantly reduce dynamic and leakage power. Dynamic power is a quadratic function of the voltage and leakage power is roughly linearly proportional to supply voltage.

In-Order Execution

An inherently simple microarchitecture can also exploit the power-performance trade-off. For example, the in-order Alpha EV5 consumes half the power of the out-of-order Alpha EV6 (when operating at the same frequency), while trading off a significant degree of IPC. If the trailing thread leverages the RVQ for perfect value prediction, in-order execution does not degrade IPC as instructions are never stalled at dispatch. The trailing thread continues to achieve high IPCs and we can further apply DFS and DVS to the trailing core.

Workload Parallelization

The trailing thread can be decomposed into a number of parallel threads [7]. Such a workload is embarrassingly parallel as the contents of the RVQ can be used to eliminate all dependences between trailing threads. Such parallelization boosts IPC and therefore affords further opportunities for DFS and DVS. In the next section, we will provide analytical estimates of power overheads when the above three techniques are incrementally employed.

III. Analytical Power Estimates

After a close analysis of detailed architectural power simulators (based on Wattch [2]), we have identified the factors that have the greatest influence on RMT power consumption. It is possible to improve the accuracy of our analytical model by taking more factors into account. For example, dynamic power is considered to be linearly proportional to instruction count. The model can be improved by considering the mix of instruction types.

Firstly, the power consumed by the core executing a single leading thread is the sum of its leakage and dynamic power.

$$Leading\_power = leakage_{leading} + dynamic_{leading}$$

The power consumed by the baseline RMT mechanism that executes the trailing thread on a neighboring identical core at the same frequency is given by the following equation.

$$Trailing\_power = leakage_{leading} + \\ dynamic_{leading}/wrongpath\_factor + queue\_power$$

In the above equation, $wrongpath\_factor$ takes into account the fact that perfect branch prediction allows the trailing thread to never execute instructions that are eventually squashed. The term $queue\_power$ includes additional power consumed within the LVQ, RVQ, BOQ, and StB. The dynamic power saved by not accessing the data caches and branch predictor of the trailing core is factored into the $queue\_power$ term.

Since the IPC of the trailing thread is typically much higher than that of the leading thread, we will assume that its frequency can be scaled by a factor eff_freq. The power of the trailing core is now given by the following equation.

$$Trailing\_power = leakage_{leading} + \\ dynamic_{leading} \times eff\_freq/wrongpath\_factor \\ + queue\_power$$

If scaling the frequency by a factor eff_freq allows us to scale voltage by a factor $eff\_freq \times v\_factor$ (in practice, v_factor is greater than 1), the trailing core's power is as shown below.

$$Trailing\_power = leakage_{leading} \times eff\_freq \times v\_factor \\ + dynamic_{leading} \times eff\_freq^3 \times v\_factor^2/wrongpath\_factor \\ + queue\_power$$

Next, we will consider the effect of employing an in-order core. Note that eff_freq will be a function of the in-order core's IPC, which in turn, depends on whether we use value prediction or not. If value prediction is employed, $queue\_power$ will change as there will be more accesses to the RVQ. The trailing core's power is given by the following equation (when only employing DFS). The terms lkg_ratio and dyn_ratio (typically greater than 1) account for the power difference between an out-of-order and in-order core.

$$Trailing\_power = leakage_{leading}/lkg\_ratio + \\ dynamic_{leading} \times eff\_freq/(wrongpath\_factor \times dyn\_ratio) \\ + queue\_power \quad (1)$$

Finally, we will consider the effect of parallelizing the verification workload across N in-order trailing cores. Assuming that we only employ DFS for each in-order core, trailing thread power is given by:

$$Trailing\_power = N \times leakage_{leading}/lkg\_ratio + \\ N \times dynamic_{leading} \times eff\_freq/(wrongpath\_factor \times dyn\_ratio) \\ + queue\_power$$

Note that the dynamic power remains the same as in Equation (1) – eff_freq goes down by a factor N, but that amount is now expended at N different cores. In other words, the same amount of work is being done in either case. Leakage power increases because leakage is a function of the number of transistors being employed. Parallelization has a benefit only if we are also scaling voltage. Power is then expressed as follows:

$$Trailing\_power = v\_factor \times leakage_{leading}/lkg\_ratio +$$
$$N \times v\_factor^2 \times dynamic_{leading} \times eff\_freq$$
$$/(wrongpath\_factor \times dyn\_ratio \times N^2) + queue\_power$$

Voltage cannot be arbitrarily scaled down as N is increased. Hence, v_factor in such a scenario is likely to be much higher than in the scenarios examined earlier. While the above estimates only consider a single-thread workload, they can be easily extended to deal with multi-threaded workloads as well.

We validated our analytical models by comparing their results with detailed simulation results obtained from Wattch [2]. A baseline out-of-order processor simulation was used to compute parameters such as the contribution of leakage, wrong-path instructions, etc. An implementation of trailing cores with frequency scaling was modeled in detail on Wattch (for out-of-order and in-order trailing cores) and the outputs of the analytical model closely matched those of Wattch. Results were also verified after modifying major parameters such as $wrongpath\_factor$, eff_freq, and contribution of leakage for out-of-order and in-order trailers. In all cases, the accuracy of the analytical model was within 10%.

IV. Observations

The previous section attempts to capture the first-order effects of a number of factors on power consumption. These factors include the following:

- Contribution of leakage in a given baseline technology
- Number of wrong-path instructions executed by a thread
- Power overhead of inter-core buffers
- Effective frequency, which quantifies the IPC benefit of a perfect cache, branch predictor, and value predictor
- Ratio of voltage and frequency scaling (v_factor)
- Ratio of power consumed by in-order and out-of-order microarchitectures
- Degree of parallelization for verification workload

A designer can plug in various assumptions for the above factors to determine early estimates of the power efficiency of each approach. As examples, we plot the effects of the major parameters in Figures 2-7.

We make the following assumptions for each of these figures, unless otherwise specified. In-order cores consume half the leakage and dynamic power of the out-of-order leading core. $Wrongpath\_factor$ is assumed to be 1.25 and $queue\_power$ is always assumed to be 10% of the leading core's power. For a single trailing thread, the eff_freq is assumed to be 0.5. When the trailing thread is parallelized across two in-order cores, eff_freq drops to 0.25 and voltage is scaled such that v_factor is 1.5. Leakage power is assumed to be 10% of the baseline core's total power.

Figure 2 shows the power overhead of the trailing thread executing on in-order and OoO cores with DFS for different effective frequency assumptions. While lower frequencies do

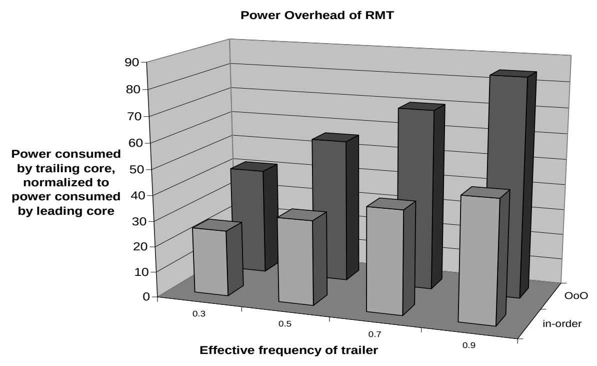

Fig. 2. Power overhead of the trailing core, relative to the leading core, as a function of effective frequency, for in-order and OoO trailing cores.

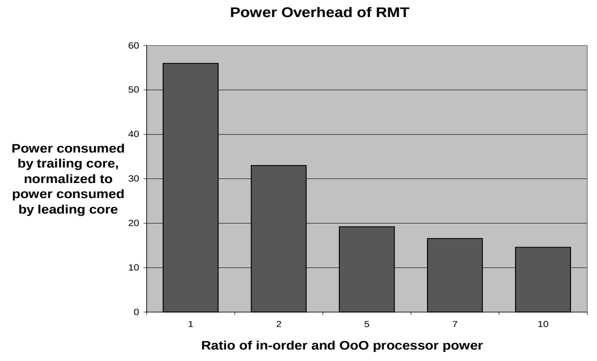

Fig. 3. Power overhead of the in-order trailing core, relative to the leading core, for different relative OoO and in-order power ratios.

reduce the power overheads of an out-of-order trailer, the power overhead of redundancy continues to be very high. This is partially because DFS does not reduce leakage power. An in-order core, even at high frequencies, can match the power overhead of a highly frequency-scaled out-of-order core. In our simulations with a 4-wide in-order trailer, we observed that perfect value prediction can help reduce effective frequency by a factor of two. Based on Figure 2, we can calculate that this approach is worthwhile provided value prediction does not increase the power of the inter-core buffers by an amount that is approximately 15% of leading core power. In Figure 3, we show the power overhead of the trailing in-order core for different relative power ratios of in-order and out-of-order cores. Because of the power overhead of the inter-core buffers, there is little power benefit to employing an in-order core that consumes less than 20% of the leading core's power.

Figure 4 shows the power overhead of the trailing cores for various $wrongpath\_factor$ assumptions. If the leading core suffers from a high branch misprediction rate, then the corresponding trailing core would have lower power overhead due to perfect branch prediction. This effect is more pronounced for the out-of-order model as it wastes more dynamic power executing wrong-path instructions. In Figure 5, we show the power overhead of the trailing cores for different contributions of leakage power to the baseline power. As the contribution of leakage power increases, DFS has a marginal effect on reducing the trailing core's power overheads, especially for an aggressive out-of-order processor. DVS is required to reduce

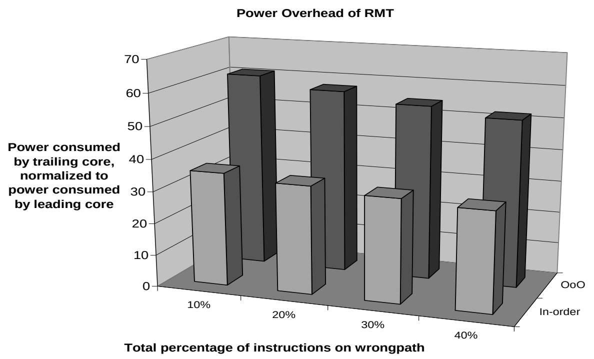

Fig. 4. Power overhead of the trailing cores relative to the leading core for different $wrongpath\_factor$ values.

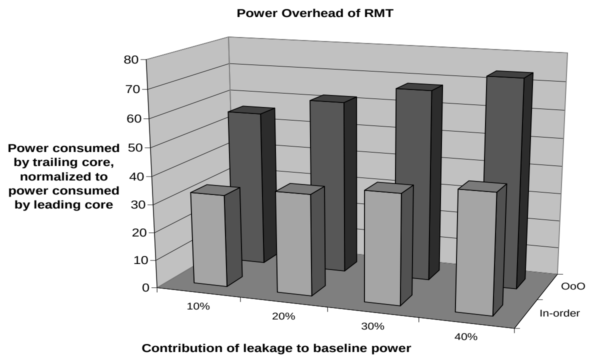

Fig. 5. Power overhead of the trailing cores relative to the leading core for different contributions of leakage power.

leakage power overheads and this effect is shown in Figure 6. Based on these results, we conclude that (for the assumed parameters) a trailer core can impose a power overhead of as little as 20% (half of it attributed to the inter-core queues), but this will require employing a power-efficient in-order core that can be voltage or frequency scaled. We also note that value prediction may be required in the in-order core to allow it to operate at low frequencies, while still matching leading thread throughputs.

Finally, we examine the effect of parallelizing the verification workload. Figure 7 shows the power overhead of the trailing thread with and without parallelization for different assumptions on the contribution of leakage power. Recall that parallelization helps reduce dynamic power, but increases leakage power (even if DVS is assumed). If the contribution of leakage is high, parallelization actually causes an increase in total power. If the contribution of leakage is low, parallelization reduces overall chip power by less than 5%. Voltage scaling also entails a non-trivial cost and future technologies will afford smaller voltage margins. Such an early rough estimate leads us to the conclusion that workload parallelization yields little benefit for many design points.

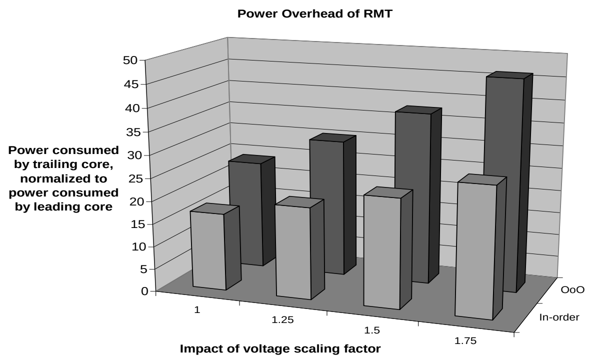

Fig. 6. Power overhead of the trailing cores relative to the leading core for different v_factor values.

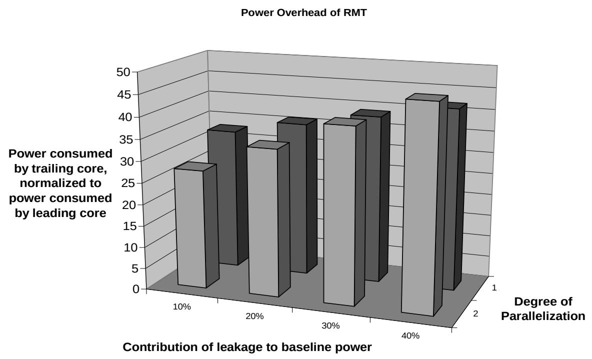

Fig. 7. Power overhead of the trailing core, relative to the leading core, with and without parallelizing the verification workload.

References

[1] T. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings of MICRO-32*, November 1999.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of ISCA-27*, pages 83–94, June 2000.

[3] M. Gomaa, C. Scarbrough, and T. Vijaykumar. Transient-Fault Recovery for Chip Multiprocessors. In *Proceedings of ISCA-30*, June 2003.

[4] N. Madan and R. Balasubramonian. Power-Efficient Approaches to Reliability. Technical Report UUCS-05-010, University of Utah, December 2005.

[5] A. Mendelson and N. Suri. Designing High-Performance and Reliable Superscalar Architectures: The Out-of-Order Reliable Superscalar O3RS Approach. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2000.

[6] S. Mukherjee, M. Kontz, and S. Reinhardt. Detailed Design and Implementation of Redundant Multithreading Alternatives. In *Proceedings of ISCA-29*, May 2002.

[7] M. Rashid, E. Tan, M. Huang, and D. Albonesi. Exploiting Coarse-Grain Verification Parallelism for Power-Efficient Fault Tolerance. In *Proceedings of PACT-14*, 2005.

[8] J. Ray, J. Hoe, and B. Falsafi. Dual Use of Superscalar Datapath for Transient-Fault Detection and Recovery. In *Proceedings of MICRO-34*, December 2001.

[9] E. Rotenberg. AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors. In *Proceedings of 29th International Symposium on Fault-Tolerant Computing*, June 1999.

[10] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinatorial Logic. In *Proceedings of DSN*, June 2002.

[11] J. Smolens, J. Kim, J. Hoe, and B. Falsafi. Efficient Resource Sharing in Concurrent Error Detecting Superscalar Microarchitectures. In *Proceedings of MICRO-37*, December 2004.

[12] T. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-Fault Recovery via Simultaneous Multithreading. In *Proceedings of ISCA-29*, May 2002.

[13] N. Wang, J. Quek, T. Rafacz, and S. Patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *Proceedings of DSN*, June 2004.