Mitigating the Impact of Process Variations on Processor Register Files and Execution Units

Xiaoyao Liang and David Brooks
Division of Engineering and Applied Sciences
Harvard University
33 Oxford St., Cambridge, MA 02138
{xliang, dbrooks}@eecs.harvard.edu

Abstract

Design variability due to die-to-die and within-die process variations has the potential to significantly reduce the maximum operating frequency and the effective yield of high-performance microprocessors in future process technology generations. One serious manifestation of this increased variability is a reduction in the mean frequency of fabricated chips due to fluctuations in device characteristics causing reduced circuit performance. In this paper, we propose to mitigate the impact of variations through variable-latency register files and execution units which are key architectural components that may encounter variability problems. We also illustrate the importance of closing the gap in expected delay of these distinct structures. A post fabrication test and configuration strategy is proposed. We find that 23% mean frequency improvement with an average IPC loss of 3% (and never exceeding 5% for worst case chips) is possible for the 65nm technology node by properly adopting the proposed schemes.

1. Introduction

Future advanced process technologies will continue to provide transistor density and speed improvements through aggressive feature scaling and novel device topologies. Unfortunately, chip designers will soon be forced to design with the expectation of significant variations in transistor feature sizes and threshold voltages due to sub-wavelength lithography and random dopant fluctuations. Process variations (PV) will manifest in several different ways – through random or systematic (correlated) variations that may occur within a single die (WID: within-die variations) or across multiple dies (D2D: die-to-die variations) in a production run. Recent estimates suggest that process variability could impact performance by a full process generation and traditional frequency binning alone cannot solve the strong variation problem [7].

While the last few years have seen an increased interest in developing statistical timing models and circuit-level techniques to address variability, there has been comparably little work at higher levels of design. However, fundamental microarchitectural decisions that impact performance (e.g, selection of pipeline depth [16] and sizing of architectural resources [18]) have a substantial impact on the distribution of chip frequency as well as IPC (instructions-per-cycle). Under strong PV, each fabricated chip has different characteristics, and therefore we argue that machines should no longer be designed with rigid, fixed configurations. We propose to use variable-latency structures to mitigate the impact of PV.

In this paper, we provide a detailed study of the microarchitectural ramifications and circuit implementation of the variable-latency approach in the integer and floating-point register files (RF) and execution units, which are similar to EBOX and FBOX in the Alpha 21264 microprocessor [15]. The register files and execution units provide a compelling starting point for our investigation into variable-latency microarchitectures because these units are frequency as well as IPC-critical datapath components. Both structures are typically thermal hotspots [24], so other techniques like forward body bias (FBB) cannot be blindly applied due to increased leakage power [20]. Simply upsizing the transistors for speed can incur large power and area overhead. Moreover, these structures provide an example of a direct connection between timing-critical SRAMs and logic structures which we show provides a special opportunity for optimization under PV.

In this initial study, we assume that variability in other microarchitectural blocks can be handled through worst-case design or with other variability-mitigation techniques such as adaptive-body bias (ABB) [28] or cache resizing [3]. We demonstrate that our approach moves the register files and execution units off the list of PV-critical components, and we note that the approach can also apply to other microarchitectural units (e.g. front-end and memory hierarchy pipelines).

This paper takes several steps in the direction of PV-tolerant architecture design and makes three major contributions:

- We motivate and describe a variable-latency register file (VL-RF) and a variable-latency floating-point unit (VL-FPU). The VL-RF helps to reduce the number of entries with long access times in the register file. The VL-FPU helps to mitigate both random and systematic variations. These techniques provide a large frequency benefit with a small IPC penalty.
- We find under strong variations, there is a large gap in the critical delay between SRAM-dominated register files and logic-dominated execution units. We illustrate the importance of averaging the delay between these two distinct structures and show that circuit techniques like time borrowing (or slack passing) can be applied to close the gap. We also propose to use *selective* FBB for additional frequency improvement with minimal leakage power overhead.
- We propose a test strategy to combine all of these techniques and generate the best processor configuration under PV. We present detailed circuit-level Monte-Carlo simulations for the combined approach that demonstrate a 23% mean frequency improvement with a 3% average IPC loss for all chips.

The next section describes the key motivation and implementation necessary to apply VL-RF and VL-FPU. Section 3 illustrates the delay gap between SRAM and logic under PV and how our

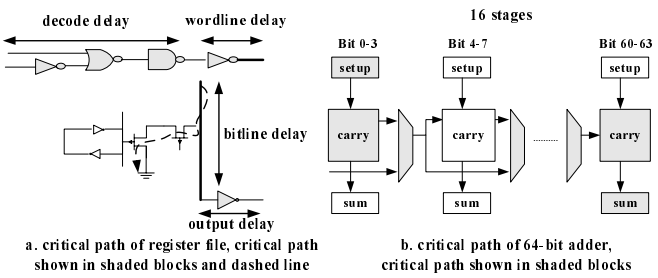

Figure 1. Critical paths of register file and integer ALU.

approach can leverage this gap. Section 4 discusses testing requirements for the system and Section 5 describes our modeling and simulation approach. Section 6 presents simulation results. Section 7 presents related work, and Section 8 summarizes our work.

2. Variable-Latency Microarchitecture

We propose variable-latency approaches to compensate for the frequency loss due to process variation and we discuss techniques that are appropriate for different microarchitectural structures with distinct characteristics under PV: the register file and floating-point execution unit (FPU). In this section and in Section 3, we use the intuitive FMAX model [7] to understand the impact of PV on each component's frequency distribution and the frequency distribution is approximated as normal distribution. However, all the simulation results presented in this paper are derived from detailed circuit-level Monte-Carlo simulations discussed in Section 5.

The FMAX model provides a prediction for the maximum frequency distribution of a microprocessor as a function of the amount of PV and circuit parameters which are key to the delay distribution of functional units: ncp, the number of gates on a critical path, and Ncp, the number of critical paths in a unit. The FMAX model provides insight into the fact that the frequency impact due to PV is not solely determined by the severity of process variations. In fact, the performance impact is strongly related to the logical organization of the functional blocks. A larger logic depth or higher gate count on a critical path (ncp) helps to cancel out random effects on the critical path and makes the path less sensitive to variations. On the other hand, a large number of critical paths (Ncp) makes a unit very sensitive to PV, because more critical paths in a unit implies a higher probability that one critical path has delay that cannot meet timing specifications. PV-tolerant designs prefer a large ncp but a small Ncp [16]. This has significant ramifications to our approach and will be further discussed in Section 3.

2.1 Variable-Latency Register File

2.1.1 Observation

Multi-ported register file delay is dominated by SRAM access time. We focus on SRAM read access time failures which have been shown to be the most likely source of failures in SRAMs [3]. Our RF implementation will not cause read stability failure because the read bitlines are decoupled from the internal cell, as shown in Figure 1a, thus preventing the unintentional flip of the stored value. We defer write failure modeling to future work. The impact of variability on SRAM structures is severe because of the relatively small number of gates that impact delay (ncp) and the relatively large number of parallel critical paths (Ncp). Figure 1a illustrates that only a few gates impact the SRAM read access time and variation within any of these gates will have a large impact on delay.

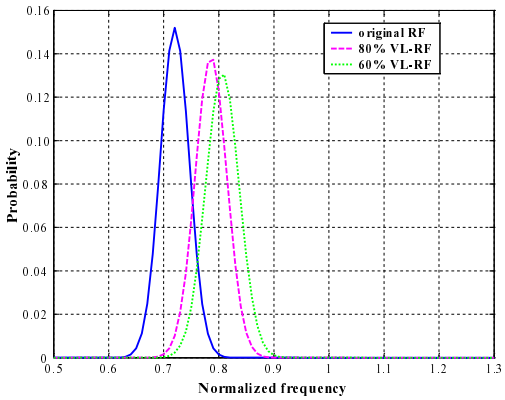

Figure 2. VL-RF frequency distribution improvement.

The number of critical paths for the register file is large, because it is approximately equal to the product of the number of entries in the register file, the number of bits per entry, and the number of read ports. SRAM frequency is limited by the slowest bit of every entry-port combination. Furthermore, compared to larger SRAM structures like L1/L2 caches, register file SRAMs tend to have a smaller amount of wire delay due to fewer global wires [4]. In this work, we extract wire parasitics from the SRAM layout, and wire delay is treated as a fixed portion of the nominal delay. We do not model variations in wires because metal lines are significantly more PV-tolerant than gates [5]. The behavior of the RF under PV is in sharp contrast to a logic-dominated integer adder which has many gates on relatively few critical paths (Figure 1b).

2.1.2 Variable-Latency Technique

To reduce the impact of extremely slow entry-port combinations, we propose to use variable-latency register files. We will divide the RF entries into fast entries that perform read access in a single clock cycle, and slow entries that take two clock cycles. We define a $n\%$ VL-RF to mean that for each read port in the register file, the slowest $(100 - n)\%$ of the entries will be marked slow and accessed in two clock cycles. The final frequency of the register file is determined by the slowest access time in the SRAM for the remaining $n\%$ entries for all the ports. The VL-RF effectively avoids the long delay paths in the SRAM. Figure 2 shows the frequency distribution with $n = 100\%$ (unmodified machine), $n = 80\%$, and $n = 60\%$. Our detailed simulations show a 27% mean frequency loss for the register files compared to a machine without PV. We find that with the 80% VL-RF, the mean frequency improves about 12% as the slowest 20% entries for each read port are identified and accessed in two cycles, effectively removing them from the list of critical paths. Further extension to 60% VL-RF does not help much (less then 4% improvement). Based on these results, we will use 80% VL-RF throughout this paper.

Whenever a slow entry for a certain port is read, the pipe that port belongs to must stall for one clock cycle for RF reading. This adds an additional bubble into the pipeline and reduces IPC. We find that VL-RF causes about a 9% IPC loss for SPEC2000 benchmarks, discussed later in Section 6. Since the frequency improvement is about 12%, most of the benefit is compromised by IPC loss and the naïve variable-latency RF cannot help to improve performance. Our previous work also considered marking slow entries as completely unusable, but we found that this results in a huge IPC loss [17], and thus we do not consider it further.

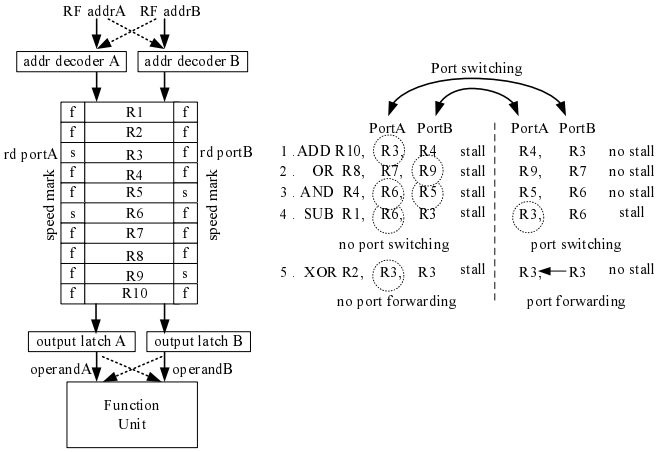

Figure 3. Port switching to reduce stalls. Circled registers means the register causes a stall for reading.

Due to the random nature of PV, different ports in a single register file can have different characteristics. Thus, it is common that the same entry may be slow for one port and fast for another port. We leverage this characteristic with an approach called *port switching*. With this technique, we propose to opportunistically switch from slow ports to fast ports for RF entry access in order to mitigate IPC loss.

Figure 3 provides a simple example to demonstrate the approach. In this example, there are 10 entries in the register file with 2 read ports, each of which provides one operand to the execution unit. For each port, the two slowest entries are marked. The labels 'f' and 's' identify the fast and slow entries for each port. Without port switching, every instruction in the code sequence in Figure 3 incurs a stall, because each instruction has operands that require reading from the slow entries from either or both of the two ports. However, if we can identify the problem and steer the register access from the slow port to the fast port before entering the register file read stage, we can avoid stalls. For the first ADD instruction, we can use portA to read R4 and portB to read R3. Since both ports are fast, we can eliminate the stall. By using port switching, we can avoid most of the stalls in the sample code sequence. Port switching cannot help the fourth instruction because there is always one port reading a slow entry whether switching or not. For the fifth instruction, switching also cannot help. But because the two operands are reading the same register, and since portB can read faster, it is possible to forward the data to portA. In order to simplify the hardware implementation, we limit port switching to the two read ports connected to the same execution unit. By using port switching, the IPC loss due to VL-RF can be reduced to 2% and detailed simulation is presented in Section 6.

2.1.3 Implementation

Microarchitectural modifications are necessary to implement variable-latency register files with port switching. First, we assume that the SRAM port entry speed information can be collected using BIST [27] and that the chip operating frequency is pre-selected (more details in Section 4). This information is configured into a ROM at test time. When a free physical register in the free register pool is called up for renaming, the per-port speed information will be loaded from the ROM and accompany the standard register entry address into the rename table. For example, for a 128-entry register file with 4 read ports, we use an extended address of 11 bits, with the lowest 7 bits storing the standard RF entry address and the highest 4 bits storing the speed information (fast or slow) for each of the four read ports for that entry. When a source operand is renamed in the renaming stage, it grabs the physical RF address as well as the four port speed bits and creates an address bundle. If the address is not a true register file address (e.g. immediate operands or other special registers that do not require a register file access), all the ports speed bits will be set to fast. The 11-bit address bundle is propagated through the pipeline.

Figure 4 shows the hardware implementation details. When an instruction is waiting in the issue queue for wake-up, the port speed bits of both RF addresses are checked. If a switch can help to eliminate the stall, the "switch_addr" signal is generated. This switch signal will be passed (latched), if this instruction is selected and issued to the register file, to control the multiplexors to properly steer the address to the address decoder in the RF, and the switched address will be used to access the register file. At the same time, the logic compares the two addresses and decides if a port forwarding operation is necessary. The port forwarding and switch signal will be combined to generate "switch_out0" and "switch_out1" signals and these signals will be latched twice to control the output multiplexors and switch the operands back or forward the operands before entering the execution stage.

Port switching cannot avoid all stall conditions due to slow register file entries so stall detection logic is implemented. Whenever necessary, a stall signal is generated and latched when this instruction is issued and passes back to the issue stage so that all the following dependent pending instructions must wait for an extra cycle and no instruction can be issued to this pipe for the next clock cycle. The stall latch is sticky and it will reset to "0" the next cycle after it is asserted. In order to allow the register file to take two clock cycles for access, the stall signal must also be latched twice to prevent precharge for the second cycle of the access. Forwarding also needs to be changed. When a back to back instruction needs a stall, the forwarding data is propagated through the forwarding path2 rather than forwarding path1.

All the additional control signals are generated in parallel with wake-up and selection in the issue queue, so they will not add to the issue queue critical path delay. However, the new multiplexor before the decoder in the RF stage adds to RF critical path. Also, the multiplexors before the ALU is expanded for operand switching. The delay of this additional hardware can also vary under PV. For all the results reported in this paper, all of these hardware overheads and variations have been included into our circuit simulation.

VL-RF also requires some small modifications to the scheduling logic. The RF stall signal is latched, if necessary, during the beginning of the cycle after the instruction is issued. So all dependent instructions, including back-to-back instructions, will see the stall signal at the beginning of their schedule cycle at the same time they are awakened and deciding whether to issue. Thus, there will not be any speculative issue problems requiring replays. Figure 5 shows the hardware modification of the scheduling logic. In a conventional design [26], as shown in Figure 5a, each instruction operand in the issue queue includes an issue cycle register that records the issue cycle after the operand wakes up. For example, "111111" in the issue cycle register refers to a back-to-back instruction that can be issued the same cycle when it is waken up. A string of "111110" means that the instruction can be issued one cycle after wake-up. When the instruction sees the wake-up signal, the information in the issue cycle register is loaded to a shift register, and the shift register will shift right to decide when the instruction is ready for selection/issue. For back-to-back instructions, the wake-up signal

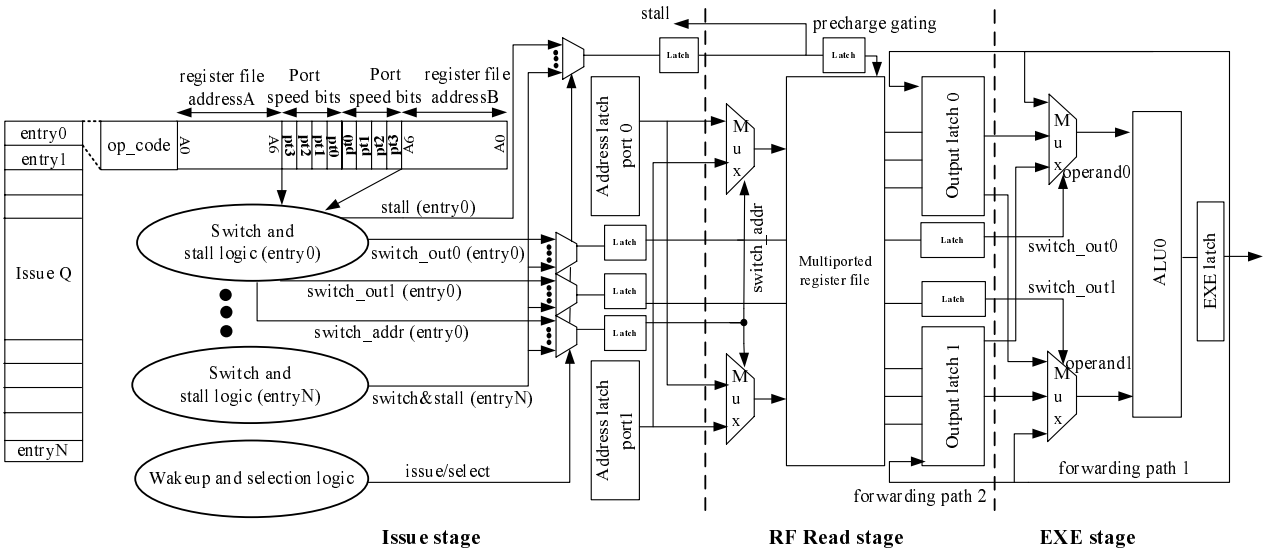

Figure 4. Hardware implementation for VL-RF, a '1' in bit pt0 means port0 reading this address is fast, while '0' means slow.

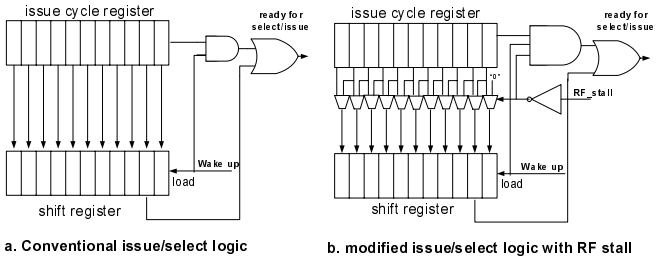

Figure 5. Scheduling logic modifications for RF stall.

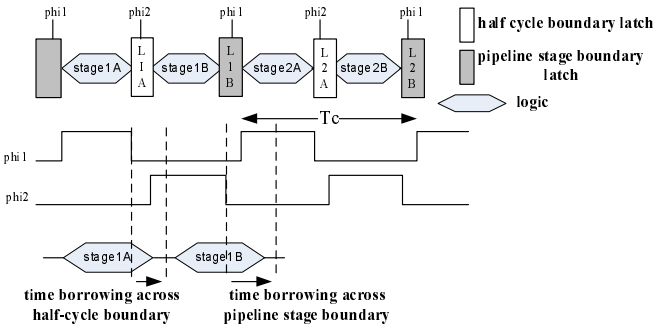

Figure 6. Time borrowing in two-phase clocking design.

requires a logical AND with the issue cycle register to ensure consecutive issue. The modified version of this logic that includes the register stall signal is shown in Figure 5b. If there is no RF stall, everything behaves exactly the same. If there is a stall, the back-to-back instruction is gated from issuing during the wake-up cycle. Also, the shift register is loaded with a biased version (left shift 1 bit) of the issue cycle register to include the extra one cycle delay for the register file access.

2.2 Variable-Latency Floating-Point Unit

2.2.1 Observation

It is well known that circuit techniques like time borrowing (slack passing) can be applied to pipelined units to mitigate delay imbalance between pipeline stages [29]. We find that time borrowing has a critical role to play in variable-latency techniques for the FPU. In contrast to the integer ALU, which is usually designed for single-cycle latency, the FPU typically requires multi-cycle latencies. While time borrowing is possible with nearly all clocking disciplines, the approach becomes trivial with pipelined structures using two-phase clocking. The basic time borrowing approach is shown in Figure 6. The diagram plots a two stage (4 half logic stages) pipeline design. The grey boxes indicate pipeline stage latches while the white boxes indicate half-cycle boundary latches; these latches are physically identical and are drawn in this fashion to illustrate the approach. Ideally, all the logic stages are balanced and should finish in a half clock cycle. However, since data can pass through latches when latches are transparent, it is possible for long delay logic to borrow time into the next clock phase. As in the figure, stage1A borrows time across the half-cycle boundary when L1A is transparent and stage1B borrows time across the pipeline stage boundary when L1B is transparent. The maximum borrowing time from one stage to another is limited by Eq. (1), where T_c is the cycle time, t_{setup} is the latch setup time, $t_{nonoverlap}$ is the non-overlapping period of the two clocks.

$$t_{borrow} \leq \frac{T_c}{2} - (t_{setup} + t_{nonoverlap}) \quad (1)$$

Time borrowing between logic stages has the potential to average out random variations. This is illustrated in Figure 7. The top of the figure shows a pipelined FPU. The dark boxes designate hard time boundaries and no time borrowing is allowed beyond these two blocks to ensure proper operation of the preceding and following stages. White and grey boxes imply soft time boundaries and time borrowing is allowed across them. For the machine without any variations, each logic stage can fit in its own delay slot, as shown in Figure 7a. With random variations, it is possible to cancel out the random delay variation through borrowing. In this example, stage1A and stage1B borrow time from stage2A, and stage2B borrows time from stage3A. For systematic variations, it is possible that every stage has slower than nominal delay paths, and each

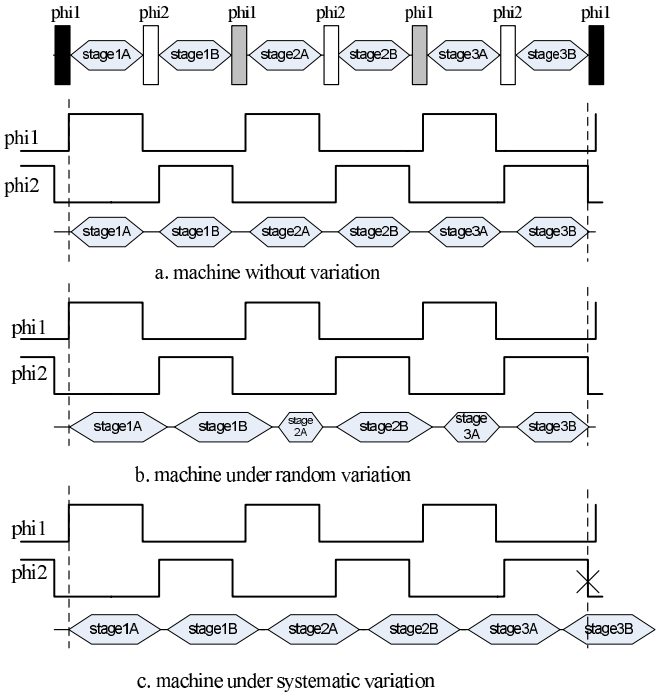

Figure 7. Time borrowing under process variations.

stage ends up borrowing time from the next stage. This results in a situation where every stage pushes a little delay overhead to the next stage from the head of the pipeline, eventually causing timing errors in the last stage, shown in Figure 7c.

2.2.2 Technique and Implementation

In a traditional design, the FPU frequency must be reduced to meet timing path delays under systematic variation. To avoid this, we propose to use a variable-latency FPU design. Instead of slowing down the frequency, we will insert an extra pipeline stage into the datapath and extend the latency of the FPU datapath by one cycle when systematic PV is large. This is shown in the top of Figure 8. If there is no significant variation and the delay can fit into three pipeline stages, the extra latches are clock gated and the pipeline looks like Figure 8a. But if large delay variations are present, and the delay can no longer fit into three cycles, the extra latches are enabled and the pipeline latency is extended to four cycles as shown in Figure 8b. The dummy stage created has no real function and is purely for the purpose of time borrowing. Except for the extra latch and multiplexor delay (delay2), almost a half clock cycle is added into the middle and the end of the pipeline for time borrowing. The clock signals for latches in between the two additional latches should be inverted if an extra stage is inserted to maintain proper timing. The extra stage makes the FPU tolerant to very large systematic variations. The frequency improvement is shown in Figure 9.

In this paper, we do not dynamically change the latency of pipelines. Once the chip is fabricated and tested, the FPU pipeline latency is determined and fixed. The latency of the pipeline (e.g. whether it requires an extra stage) is hardwired into the machine and cannot change during CPU runtime. Thus, although we call it a variable-latency pipeline, the pipeline depth is actually fixed for each individual chip. The scheduler modifications for this approach are simple, as the scheduler knows the exact latency of each FPU

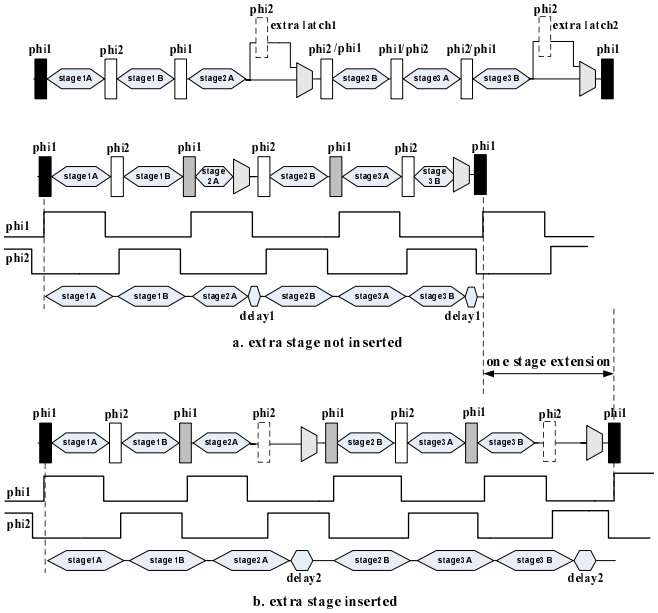

Figure 8. Variable-latency FPU.

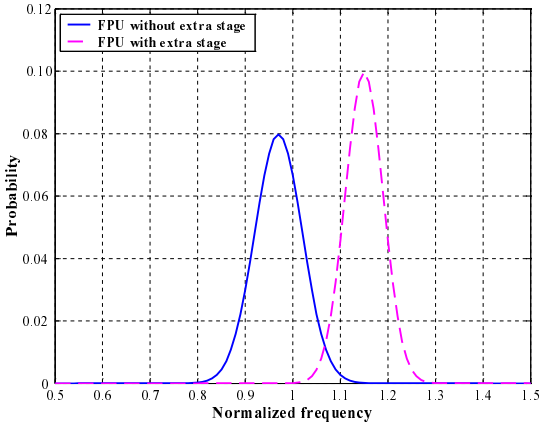

Figure 9. Frequency benefit for VL-FPU.

pipeline and can issue instructions accordingly. Obviously, different chips will suffer different amounts of PV and some machines will require more pipeline stages than others.

The scheme incurs two kinds of overhead. First, the added multiplexors add delay to the FPU critical path (delay1 in Figure 8a). We can overdesign stage2A and stage3B to accommodate the added delay but this incurs a power overhead. Time borrowing can also absorb and average the added delay through the whole pipeline. We include the delay and power overhead in our simulation. Second, there will be an IPC penalty due to deeper pipelining for chips that engage the extra stage. Because this extra stage is only added to the floating-point unit (and only for some chips), we find that the IPC penalty is small compared with the large frequency benefit. Detailed IPC and simulation results are shown in Section 6.

3. SRAM and Logic Delay Gap

SRAM and logic structures have significantly different delay distribution under PV. If two units with large delay mismatch happen to

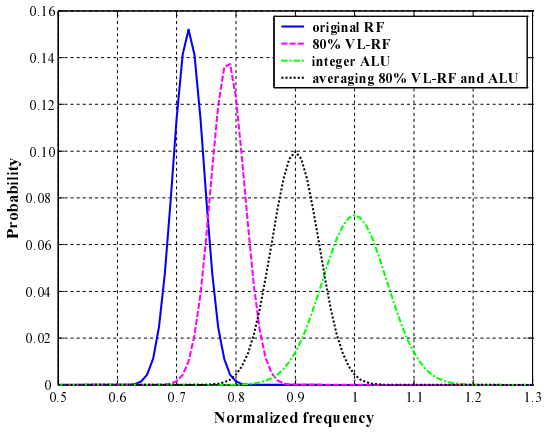

Figure 10. Frequency distribution gap between RF and ALU. Averaging delays can improve the final EBOX frequency distribution.

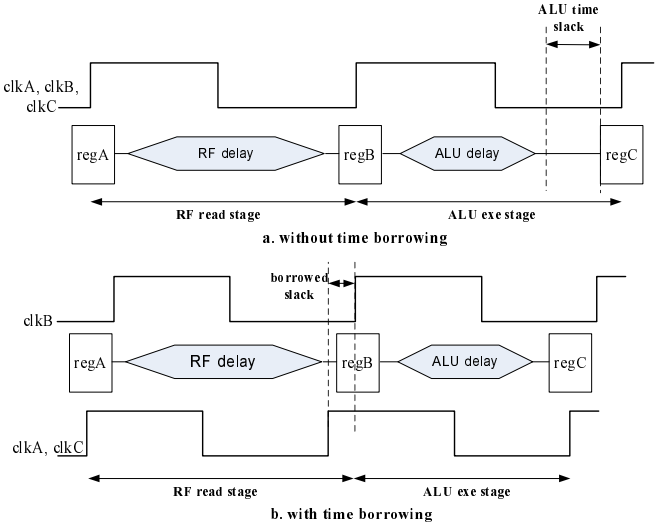

Figure 11. Time borrowing between RF and ALU.

be connected together, time borrowing can be applied to balance the delay so the frequency is not limited by the unit with worst delay. This type of borrowing only happens under PV. The unit connections are specified at the architectural-level, and it is important to guide circuit designers on where to apply borrowing.

3.1 Delay Gap Between SRAM and Logic Under PV

Previous papers [16, 19] related ncp to logic depth and Ncp to the gate count of a unit. This is not always true, especially for our EBOX and FBOX. For example, in the integer ALU, the carry chain of a 64-bit adder is always the most critical delay path while other logic paths like AND and XOR are much less critical. Recall Figure 1b which shows the critical path of a carry-bypass adder. The carry chain needs to propagate through 2 carry propagation blocks, 15 multiplexors, 1 setup and final sum block which makes ncp very large for the carry path. Furthermore, the Ncp for the ALU is just "1" because the carry chain is always the dominant critical path. Although the gates on the other paths can be downsized to trade speed and power, we assume they should be conservatively downsized to ensure nominal frequency even under variations [22].

The large ncp and small Ncp make the integer ALU fairly tolerant to PV. On the other hand, we have shown that the small ncp and large Ncp make the register file very vulnerable to PV. This argument is supported by both the FMAX model and our detailed simulation. We find that the register file is the most delay critical part for almost all chips, and there is a large gap in the delay between the register file and ALU. The gap is shown in Figure 10 which plots the frequency distribution of the two components. We find that even the 80% VL-RF is more delay critical than the ALU. Because the slower component determines the EBOX frequency, in many cases there will be some time slack in the ALU execution cycle which is purely wasted.

3.2 Time Borrowing To Close the Gap

Proper application of time borrowing can average the delay and close the gap between SRAM and logic datapaths. The EBOX we studied provides a unique opportunity for time borrowing because the register file is connected directly to the ALU, and these two structures happen to be two extremes (in terms of ncp and Ncp) under PV. The key idea of time borrowing in this situation is illustrated in Figure 11. Without time borrowing, there is some wasted time slack in the ALU execution stage. By properly delaying the clock of registerB, the RF access time can extend into the ALU stage and the ALU time slack is "borrowed" by the RF so that the frequency is increased. The EBOX frequency distribution after time borrowing is shown in Figure 10.

Although the approach is straightforward, the circuit implementation is more complex than time borrowing purely within the FPU. Figure 12 shows our SRAM design and different clocks required for proper functionality: clk_prec is used to control the precharging PMOS for the bitline precharge, clk_eval is combined with the decoder signal to properly fire the array wordline for array access, and clk_dff is used to latch the output data from the SRAM and serves as the data source to the next full cycle of ALU execution. The basic timing scheme for these clocks is shown in Figure 13a. The SRAM access starts from the decoder. The precharge logic raises the bitlines in parallel with address decode, effectively hiding the precharge time. After that, the array starts to evaluate (wordline delay, bitline delay, output buffer delay). Finally, the data is latched.

The modified clock scheme is shown in Figure 13b. The falling edge of clk_eval is delayed to extend the array evaluation time. The delayed clock extends into the ALU execution stage and borrows ALU time slack. Clk_dff must be delayed accordingly to properly latch the SRAM output data. The precharge time is squeezed because the array needs more time for evaluation. Also the precharge clock can be gated by the precharge gating signal generated from the variable-latency register file.

Three things decide the amount of time that can be borrowed by the RF. First, since the precharge time is squeezed, we must guarantee that the bitlines can be precharged back to the high value. The PMOS precharge transistors can be oversized slightly to account for the relatively short precharge period. Second, since the array evaluation extends to the next cycle, the amount of time borrowed should be less than the fastest decoder path to avoid race between the current access and the following access (the current array access is overlapped with next read decode). Especially under variation, some decoder paths can be faster than nominal delay. Third, the time borrowed from the ALU plus the ALU delay should still be within one clock cycle. Due to these limitations, we use very conservative time borrowing. In our design, we set the maximum delay chain to be 20% of the nominal clock cycle time. The delay chain

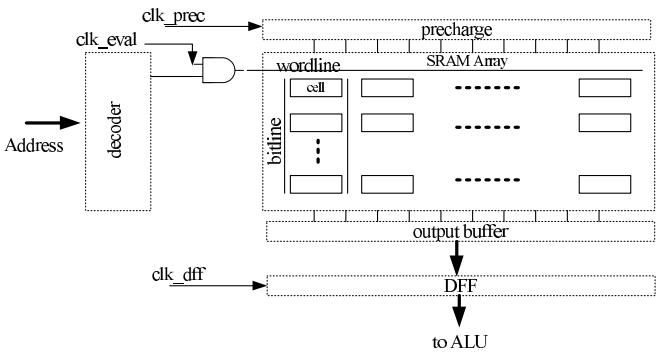

Figure 12. SRAM control clocks.

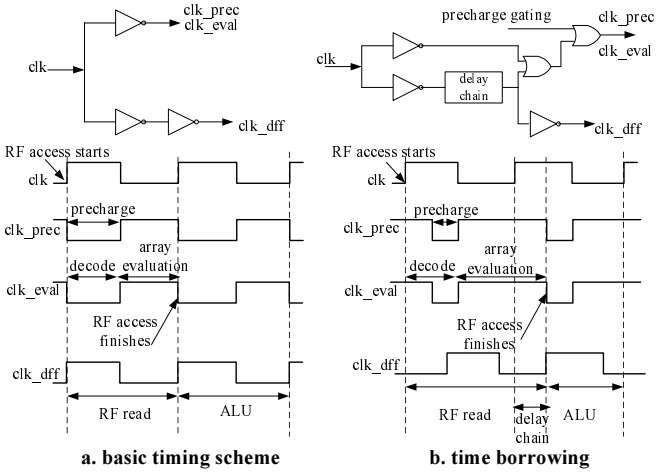

Figure 13. Clock scheme of RF for time borrowing.

also varies, so we use a worst case of $\pm 5\%$ of clock uncertainty in our delayed clock network in the simulation.

After applying time borrowing to allow slack exchange between the PV-sensitive RF and less PV-sensitive ALU, there is one more optimization that we can employ if fine grained forward body bias (FBB) is permitted in the design flow. Previous work has suggested applying FBB to blocks if those units are impacted by PV, e.g. if the register file appears to be slow, FBB is applied so that transistors can have higher speed [28] and thus the whole RF can run faster. But FBB also causes significant leakage power, and because the register file is a well-known thermal hotspot [24] composed of many transistors, applying FBB to register files is something that designers would like to avoid. Time borrowing in our coupled RF-ALU design provides an alternative solution. Instead of applying FBB to the RF, we selectively apply FBB to the carry chain in the integer ALU. This allows the speed of the ALU to increase providing more time slack within the integer ALU for borrowing by the register file. This allows designers to avoid applying FBB to the register file, but still achieve some of the benefits. Obviously, the amount of time that can be borrowed is still limited by the RF precharge time and decoder delay as described above.

We apply time borrowing between the integer RF and ALU (with and without selective FBB). We also allow the floating-point RF to borrow slack from the FPU. FBB is not required in this case, because the longer pipeline and extra stage in the FPU provide sufficient time slack opportunities to balance the overall delay.

4. Testing Strategy

This paper proposes several variable-latency schemes that seek to provide the best microarchitectural configuration for each chip. In this section, we describe an overview of the testing procedures that will be required by this approach. It is likely that testing methodologies will need to adapt to future designs which incur variability problems and the development of more built-in test strategies will help mitigate the testing overheads. With our approach, each fabricated chip should go through a standard test flow that must be augmented to choose the best frequency/microarchitectural configuration given the PV-constraints. We assume the processor has implemented BIST for SRAM structures [27] and there is a suitable range of tunable frequencies.

The test strategy is divided into three major steps. For the first step, register file BIST is carried out. For each port in the RF, speed information is collected for each entry, the slowest 20% entries are found and marked as "slow," and the remaining entries are marked as "fast". This information is configured into a ROM which will be loaded at CPU runtime for the variable-latency RF. In the second step, the remaining 80% entries from each port in the integer RF are connected to the integer ALU and test vectors are inserted to find the highest stable operating frequency. For each frequency, we sweep different delay chain configuration to find the proper amount of time-borrowing. For the third step, the FPU is tested under the frequency determined by the integer side. We sweep different delay chain configurations for the floating-point RF to find the necessary amount of time-borrowing. The need for the extra FPU stage is also tested. Finally, if the FPU test fails even with the extra latch (infrequently), we need to loop back to the second step and try the second highest frequency for the integer side and so on until we find the best acceptable frequency for all components.

5. Experimental Methodology

This section described the circuit implementation of our design, the Monte-Carlo based PV-simulation methodology, and our architectural performance simulator.

5.1 Monte-Carlo Circuit Simulation

Since there is no widely accepted analytical model for determining delay distribution under PV, we apply Monte-Carlo based circuit simulation, which can be used to closely mimic real chip fabrication. We do not use any simplified analytical delay equations. All the delay values used are extracted from Cadence and Synopsys CAD tools. For all the units considered in this paper, we build corresponding circuits and layout. The register file and integer ALU were designed using a custom flow, while the FPU was designed in Verilog and synthesized, placed, and routed using Synopsys tools. The FPU is properly pipelined, path delay balanced, and retimed to match the delay of the integer ALU and register file. We build four copies of a 64-bit carry-bypass adder and integer unit, three copies of an 80-entry, 4-rd/2-wr port register file, a 6-stage FPU adder and a 6-stage FPU multiplier, both IEEE 754-compliant. All gate parameters and wire parasitics are extracted from layout and back annotated. The design and layout is based on UMC 130nm process library. The gate delay, transistor sizes, and layout area are linearly scaled down to 65nm. Berkeley Predictive Technology Models (BPTM) [10] are used whenever we need to perform HSPICE simulation at the 65nm node.

Gate length (L) and threshold voltage (V_{th}) have been identified as the two main sources of transistor variations for the next several generations of process technology [8, 11]. In order to extract the

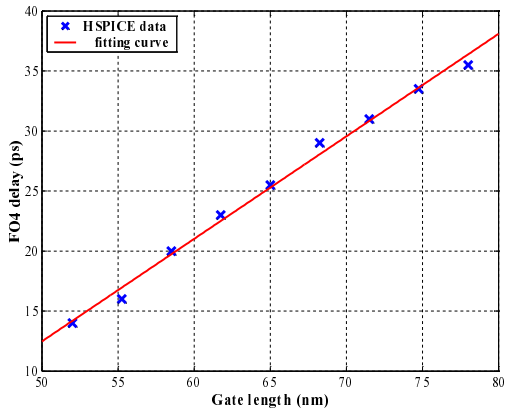

Figure 14. HSPICE curve fitting for an FO4 inverter.

delay variations due to the device parameter variations, we use a first order approximation which is widely used in statistical timing analysis [1, 21]. Eqs. (2, 3, 4) show the relation. The delay for a critical path is the summation of the path nominal delay (D_0) and the additional delay caused by parameter fluctuation of each gate on the path, assuming n total gates. For small scale fluctuations, a first order linear approximation is sufficient and the coefficients (a_k and b_k) can be curve-fitted with circuit simulation. Figure 14 shows the delay and gate length fitting curve for an FO4 inverter using BPTM, and the linear fit matches the HSPICE simulation for the range under consideration.

$$D_{path} = D_0 + \Delta D_{L_1} + ... + \Delta D_{L_n} + \Delta D_{V_{th_1}} + ... + \Delta D_{V_{th_n}} \quad (2)$$

$$\Delta D_{L_k} = \frac{\partial D}{\partial L_k} \times \Delta L_k = a_k \times \Delta L_k \quad (3)$$

$$\Delta D_{V_{th_k}} = \frac{\partial D}{\partial V_{th_k}} \times \Delta V_{th_k} = b_k \times \Delta V_{th_k} \quad (4)$$

$$L = L_0 + \Delta L = L_0 + \Delta L_{D2D} + \Delta L_{WID} \quad (5)$$

$$V_{th} = V_{th_0} + \Delta V_{th} \quad (6)$$

For each gate, we model the gate length as in Eq. (5), where L_0 is the nominal gate length. We consider both die-to-die and within-die components of the variations. For die-to-die variations, we generate a random variable for each chip according to a normal distribution and use this variable for ΔL_{D2D} for every gate on that chip. For within-die variations, we consider systematic correlations. To capture this effect, we use the method introduced in [1]. The component area is divided into a multi-level quad-tree partitioning as shown in Figure 15. We show the layout of the FPU adder in the figure to illustrate how the method works. The FPU adder area is covered with several layers of quadrants. For each quadrant we generate a random variable according to a normal distribution. The ΔL_{WID} of a gate can be obtained by adding up all the random variables of the quadrants it belongs to. For example, the gate (nand1) shown in the figure has within-die gate length variations of $\Delta L_{WID} = rand_{0,1} + rand_{1,1} + rand_{2,1}$. Therefore, nand1 and nand2 have strong correlation because they share the variable $rand_{0,1}$ and $rand_{1,1}$, while nand1 has less correlation with nand3, because they only share the variable $rand_{0,1}$. The exact location of the three gates is marked on the FPU layout. This method captures the proximity related correlations. According to the data in

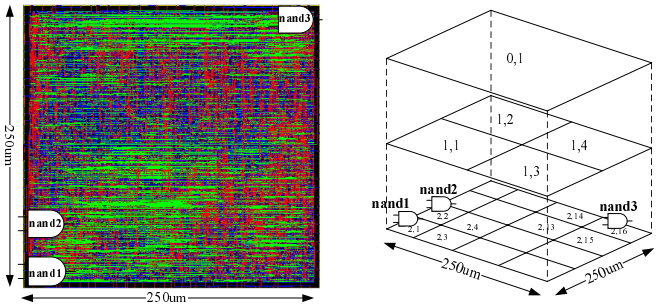

Figure 15. Multi-level partitioning for within-die gate length variations.

Issue Width	4 instructions	Issue Queues	20-entry INT, 15-entry FP
Load Queue	32-entries	Store Queue	32-entries
Reorder Buffer	80-entry	Physical Register File	80-entry INT, 72-entry FP
Instruction TLB	128-entry Fully-Associative	Data TLB	128-entry Fully-Associative
Integer Functional Units	4 FUs	Floating Point Functional Units	2 FUs
Instruction Cache	32KB, 2-way Set Associative	Data Cache	32KB, 2-way Set Associative
L2 Cache	2MB 4-way	Branch Predictor	21264 Tournament Predictor

Table 1. Baseline processor configuration.

[1, 13], 4 levels of quadrants are enough to model the correlation of the $250\mu m \times 250\mu m$ area, resulting in the finest quadrant having an area of $32\mu m \times 32\mu m$. All the gates located in the same finest quadrant have perfect gate length correlations.

The threshold voltage is modeled in Eq. (6), where V_{th_0} is the nominal value. Threshold voltage variations are mainly incurred due to the random dopant effect [8] as well as gate length variations. Since the impact of gate length on threshold has already been encapsulated in the delay-gate length fitting shown in Figure 14, we model ΔV_{th} as a random variable which obeys a normal distribution [3] only due to the random dopant effect.

We use Monte-Carlo simulation to generate all the random variables necessary in the model and generate the gate length and threshold voltage for each gate in all our modeled units. We then use the fitting curve to calculate the delay of all the gates in the chip. We generate 400 test chips and each chip has a unique delay profile for simulation. We then perform chip-by-chip delay simulation and obtain the frequency of each individual chip according to the testing flow introduced in Section 4. For our simulations, we assume $\sigma L_{D2D}/L_0 = 5\%$, $\sigma L_{WID}/L_0 = 5\%$. We assume $\sigma V_{th}/V_{th_0} = 10\%$. These assumptions are based on estimates for future variability forecast in [3, 7], and later in the paper we study the sensitivity of these assumptions.

5.2 Architecture Simulation

For our baseline machine, we use a 19FO4 design which is comparable to the reported pipeline logic for out-of-order microprocessors such as the Alpha 21264 and POWER4 [14, 25]. For our IPC simulations, we utilize the validated *sim-alpha* simulator which provides ample parameters for size and latency scaling [9]. The baseline machine uses parameters comparable to the Alpha 21264 and POWER4 listed in Table 1. We use 25 of the 26 SPEC2000 benchmarks (we had difficulty simulating "galgel") using SimPoint for sampling [23]. For each benchmark, 500 million instructions are simulated after fast forwarding to the specific checkpoint.

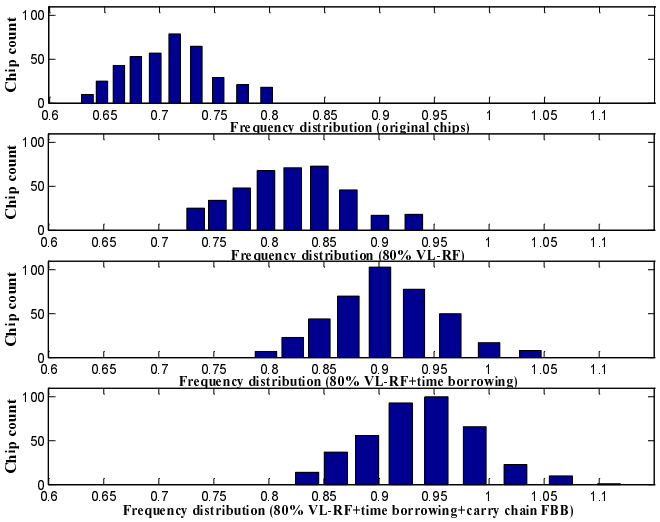

Figure 16. Chip frequency distribution with applied techniques.

When we report single IPC numbers in this paper, this is the harmonic mean of all benchmarks.

6. Simulation Results

We present our detailed simulation results in this section based on the experimental methodology described in Section 5. We first introduce the performance results in terms of frequency gain, IPC reduction, and total performance. We then discuss power and area overheads of the scheme and discuss the sensitivity of our results to the amount of process variability.

6.1 Frequency and Performance Improvement

Figure 16 shows the frequency improvement for the 400 simulated chips after application of the proposed techniques. The chip mean frequency of the unmodified machines degrades about 27% under process variations leading to unacceptable performance loss; this performance loss is within the range of other estimates [7]. We find that for most chips, the frequency is initially limited by slow register files. When the proposed variable-latency RF is used, the slowest entries in the RF no longer impact the critical delay, and the frequency is improved by about 12%.

We find that another 8% mean frequency improvement can be obtained after application of time borrowing in conjunction with the variable-latency FPU. As discussed in Section 2, time borrowing can efficiently close the gap between the register file and execution units, so the machine frequency is determined by the average delay across units rather than solely that of PV-sensitive SRAM structures.

The final plot in Figure 16 shows the impact of applying forward body bias to the integer ALU carry chain. Forward body bias can potentially improve the ALU critical path by about 15% [20]. But since we use conservative time borrowing, most of the extra time slack from FBB cannot be borrowed by the register file so we only apply a small amount of FBB (less than 300mV). After applying FBB to the carry chain in the integer ALU and leveraging time borrowing, an extra 3% frequency improvement is possible.

Applying all of these optimization techniques together helps to improve the chip mean frequency by about 23%. Since the total mean frequency loss is about 27%, most of the chips are returned to a small region around the nominal design frequency, thus effectively mitigating the impact of process variation.

The variable-latency approaches that we propose will increase the amount of stalls in the pipeline, and we must understand the IPC impact of these stalls across a range of applications. Figure 17 shows the relative IPC reduction compared to the baseline machine for each of the techniques. The first two bars in the plot show the VL-RF approach with and without port switching. The remaining three bars show the IPC impact of increasing FPU latency, when combined with VL-RF with port switching. We find that the variable-latency register file mainly impact the IPC of integer benchmarks. On average, an 80% variable-latency RF without port switching causes about 9% IPC loss, while port switching help to reduce the IPC loss to just 2%. As expected, the VL-FPU has almost no impact on integer benchmarks. For floating-point benchmarks, if the FPU multiplier has one extra stage inserted, 3% IPC loss is observed. The FPU adder is more sensitive to an extra stage and we observe a 4% IPC loss with the longer pipeline depth. If both of these units need an extra stage, there is a 5% IPC loss.

We find that all 400 simulated chips require the 80% variable-latency RF. However, the required FPU pipeline depth depends on each chip's PV condition and final operating frequency. About 15% of the chips need to open an extra stage in the FPU multiplier and 13% of the chips need an extra stage in the FPU adder. We find that 14% of the chips need an extra stage in both units. Given that many chips do not require the extra stage to be enabled, the average IPC loss is about 3% for all the simulated chips, with a worst-case IPC loss of 5%. We plot the performance (i.e. $frequency \times IPC$) of each chip before and after optimization in Figure 18. We observe a 20% mean performance improvement which is the total performance improvement achieved by applying all the proposed schemes.

6.2 Area and Power Overhead

In this section, we will quantify the area and power overhead for the approach. The area overhead mainly results from the added latches in the FPU, added logic for RF port switching and latency adaptation, and some extra multiplexor and wires. Our scheme adds about 2% area overhead to the execution units and issue queue.

The power overhead is mainly due to the extra latch power in the FPU, added logic and multiplexors, and delay chain power. Our scheme increases the power of the execution units by 5% in the worst case when all extra stages are needed. As discussed earlier, many of the chips do not need to open the extra stage so these latches can be clock gated. Compared with the total chip area and power consumption, both of these overheads are negligible.

Another advantage of our scheme is that we only need to apply a very moderate amount of FBB to the ALU carry chain. In addition, if FBB is not available or is difficult to implement, the scheme only loses 3% of the additional frequency benefit as most of the frequency loss is recaptured using architecture and circuit solutions. Compared to solutions that purely apply FBB to slow chips or units, the proposed scheme will have significantly less leakage power overhead. On the contrary, power saving techniques like reverse body bias (RBB) can be applied to overly fast components in our design as usual and there is likely to be an interesting tradeoff between more aggressive application of variable-latency architectural approaches and RBB.

6.3 Sensitivity Analysis

Section 2.2.2 describes how the VL-FPU can tolerant large random as well as systematic variations. We now perform a sensitivity anal-

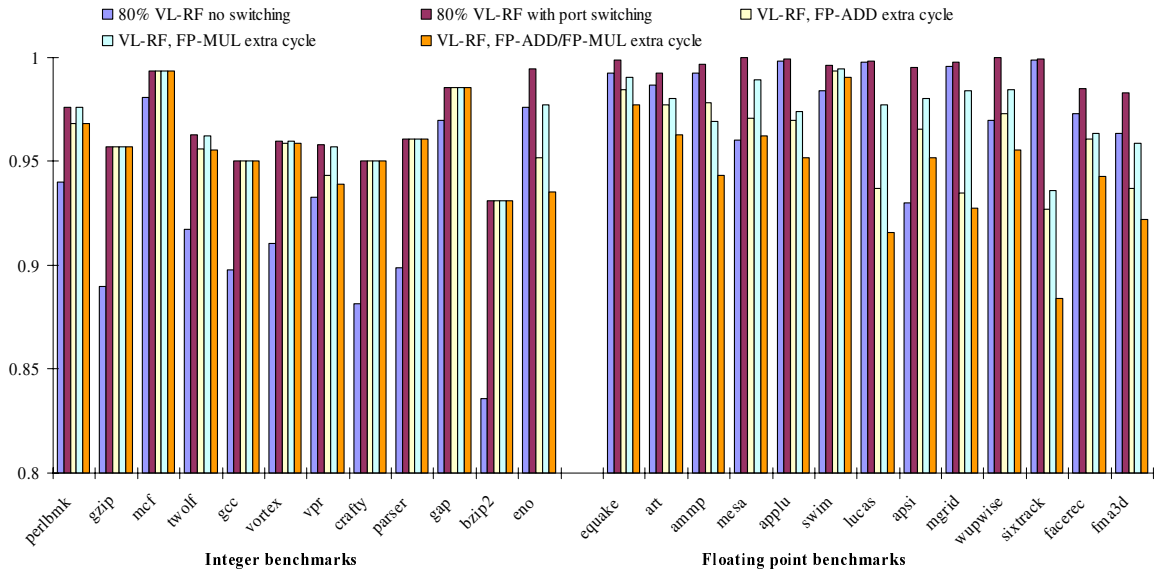

Figure 17. SPEC2000 benchmarks IPC value with applied techniques.

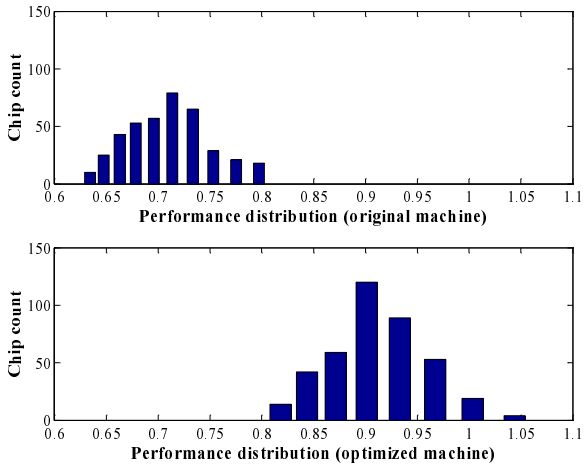

Figure 18. Performance improvement.

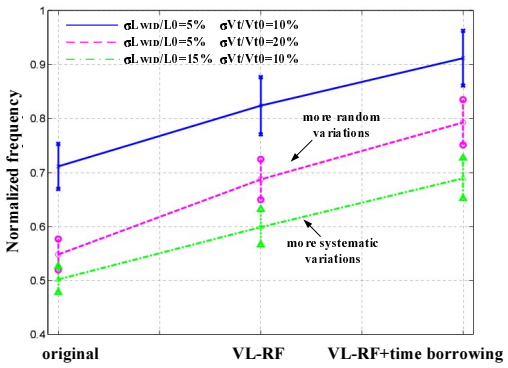

Figure 19. Sensitivity to systematic and random variations.

ysis to study the impact of systematic and random variations on the VL-RF and time borrowing. In this section we consider two additional scenarios which provide additional PV. The first scenario has extremely large random V_t variations with $\sigma V_{th}/V_{th_0} = 20\%$, and the second scenario has extremely large systematic gate length variations with $\sigma L_{WID}/L_0 = 15\%$. All other variation values are left unchanged. We simulate the VL-RF in isolation and when combined with time borrowing under these extreme parameter variation scenarios. The mean and standard deviation (error bars) of chip frequency is shown in Figure 19.

By utilizing a VL-RF, the normalized mean frequency improves from 0.55 to 0.69 for a machine with more random variations while it only improves from 0.5 to 0.6 for machine with significant systematic variations. This suggests that the VL-RF is more effective at correcting random V_{th} variations than the more correlated gate length variations. The reason for this is because the approach can more effectively identify and correct slower paths when variations are dispersed randomly rather than if large blocks comprising many entries in the SRAM are slow. It has been shown that in large SRAMs, threshold voltage variations due to random dopants are the key source of failure [3], and these results show that the VL-RF approach is well-suited to reducing the impact of these variations.

Time borrowing will also lose some efficiency under strong systematic PV. However, because time borrowing is applied across different functional units and the correlation between different units is relatively small, time borrowing can still help reduce the PV frequency impact.

7. Related Work

Bowman, et al. perform some of the first analysis in the area of process variations and point out that a technology generation of performance can be lost due to PV [7]. They also introduce the FMAX model which has been validated against a large number of chips. In recognition of the increasing problem of PV, researchers have developed statistical timing analysis techniques to be applied for deep sub-micron chip designs [1, 2, 21].

Several groups have proposed various circuit solutions for process variation. Narendra et al. [20] and Tschanz et al. [28] propose the use of adaptive body bias (ABB) to mitigate the impact of variation, but these works did not consider system-level effects.

Recently, researchers have begun to explore the system-level impact of PV. Borkar et al. conceptually demonstrate that the performance gain of deeper pipelines decreases due to the impact of within-die process variation [6]. Kim et al. perform a more quantitative analysis on pipeline depth under PV [16]. Recently, Marculescu and Talpes proposed using globally-asynchronous, locally synchronous (GALS) design techniques to design processors under PV [19]. Agarwal et al. propose to use post fabrication resource resizing to combat the impact of PV on caches [3]. Liang and Brooks study the impact of design-time microarchitectural tradeoffs on process variations [18]. Approaches based on Razor latches [12] have the potential to correct circuit level errors with double latching. However, there may be difficulties in applying Razor to a multiported register file because there is a very high probability of experiencing variations. Razor may encounter errors and subsequent pipeline flushes very frequently which is very expensive for deeply pipelined, out-of-order machines.

8. Conclusion

Process variations will be a significant impediment to continued nanoscale technology scaling. We propose the use of variable-latency techniques to mitigate the impact of variations on the register file and execution units in a microprocessor. We also point out that SRAM structures may be more sensitive to PV in future technologies and show that the proposed approach allows designers to balance the delay of slower SRAMs with less PV-sensitive logic structures. Our method effectively removes the register file and execution units from the set of delay critical components with minimal IPC impact.

Future research will be needed to develop microprocessors that are resilient to extreme process variations. We believe that the general concept of flexible, variable-latency pipeline structures will be applicable to other microarchitectural units in the machine such as the front-end and memory hierarchy, although the implementation challenges and IPC-tradeoffs are likely to be quite different.

Acknowledgments

This work is supported by NSF grants CCF-0048313 (CAREER), CCF-0429782, Intel, and IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF, Intel or IBM.

References

[1] A. Agarwal, D. Blaauw, S. Sundareswaran, V. Zolotov, M. Zhou, K. Gala, and R. Panda. Path-based statistical timing analysis considering inter and intra-die correlations. In *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, June 2002.

[2] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *International Conference on Computer-Aided Design*, November 2003.

[3] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Transactions on Very Large Scale Integration Systems*, 13(1), January 2005.

[4] B. Amrutur and M. Horowitz. Speed and power scaling of SRAM's. *Journal of Solid-State Circuits*, 35, February 2000.

[5] S. Borkar. Microarchitecture and design challenges for gigascale integration. In *Keynote Speech, 37th International Symposium on Microarchitecture*, December 2004.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variation and impact on circuits and microarchitecture. In *40th Design Automation Conference*, June 2003.

[7] K. Bowman, S. Duvall, and J. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Journal of Solid-State Circuits*, 37(2), February 2002.

[8] K. Bowman, X. Tang, J. Eble, and J. Meindl. Impact of extrinsic and intrinsic parameter fluctuation on CMOS circuit performance. *Journal of Solid-State Circuits*, August 2000.

[9] R. Desikan, D. Burger, S. Keckler, and T. Austin. Sim-Alpha: a validated, execution-driven Alpha 21264 simulator. In *TR-01-23, CS Department, University of Texas*, 2001.

[10] Device Research Group, UC Berkeley. BPTM homepage. `http://www-device.eecs.berkeley.edu/~ptm/mosfet.html`.

[11] S. Duvall. Statistical circuit modeling and optimization. In *5th International Workshop Statistical Metrology*, June 2000.

[12] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *International Symposium on Microarchitecture*, 2003.

[13] P. Friedberg, W. Cheung, and C. J. Spanos. Spatial variability of critical dimensions. In *Proceedings of VLSI/ULSI Multilevel Interconnection Conference*, 2005.

[14] M. Hrishikesh, D. Burger, N. P. Jouppi, K. I. Farkas, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *International Symposium on Computer Architecture*, 2002.

[15] R. Kessler. The Alpha 21264 microprocessor. *IEEE MICRO*, 19(2), 1999.

[16] N. S. Kim, T. Kgil, K. Bowman, V. De, and T. Mudge. Total power-optimal pipelining and parallel processing under process variations in nanometer technology. In *International Conference on Computer-Aided Design*, November 2005.

[17] X. Liang and D. Brooks. Latency adaptation for multiported register files to mitigate the impact of process variations. In *Workshop on Architectural Support for Gigascale Integration (ASGI-06, held in conjuction with ISCA-33)*, June 2006.

[18] X. Liang and D. Brooks. Microarchitecture parameter selection to optimize system performance under process variation. In *International Conference on Computer-Aided Design*, November 2006.

[19] D. Marculescu and E. Talpes. Variability and energy awareness: A microarchitecture-level perspective. In *42nd Design Automation Conference*, June 2005.

[20] S. Narendra, A. Keshavarzi, B. Bloechel, S. Borkar, and V. De. Forward body bias for microprocessors in 130-nm technology generation and beyond. In *IEEE Journal of Solid-State Circuits, Vol. 38, No. 5*, May 2003.

[21] M. Orshansky, L. Milor, P. Chen, K. Keutzer, and C. Hu. Impact of spatial intrachip gate length variability on the performance of high-speed digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(5), May 2002.

[22] D. Patil, S. Yun, S. Kim, A. Cheung, M. Horowitz, and S. Boyd. A new method for design of robust digital circuits. In *International Symposium on Quality of Electronic Design*, 2005.

[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[24] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, 2003.

[25] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma. Optimizing pipelines for power and performance. In *International Symposium on Microarchitecture*, 2002.

[26] J. Stark, M. Brown, and Y. Patt. On pipelining dynamic instruction scheduling logic. In *International Symposium on Microarchitecture*, 2000.

[27] M. Tehranipour, Z. Navabi, and S. Falkhrai. An efficient BIST method for testing of embedded SRAMs. In *Proceedings of IEEE International Symposium on Circuits and Systems*, 2001.

[28] J. Tschanz, J. Kao, and S. Narendra. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. In *Journal of Solid-State Circuits, Vol. 37, No. 11*, November 2002.

[29] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective, Third Edition*. Addison Wesley, 2004.