

# Analysis and solutions to Issue Queue Process Variation

Niranjan Soundararajan Aditya Yanamandra Chrysostomos Nicopoulos N. Vijaykrishnan  
Anand Sivasubramaniam Mary Jane Irwin  
Dept. of Computer Science and Engineering  
The Pennsylvania State University  
University Park, PA 16802  
{soundara, yanamand, nicopoul, vijay, anand, mji}@cse.psu.edu

## Abstract

*The last few years have witnessed an unprecedented explosion in transistor densities. Diminutive feature sizes have enabled microprocessor designers to break the billion-transistors per chip mark. However various new reliability challenges such as Process Variation (PV) have emerged that can no longer be ignored by chip designers.*

*In this paper, we provide a comprehensive analysis of the effects of PV on the microprocessor's Issue Queue. Variations can slow down issue queue entries and result in as much as 20.5% performance degradation. To counter this, we look at different solutions that include Instruction Steering, Operand- and Port- switching mechanisms. Given that PV is non-deterministic at design-time, our mechanisms allow the fast and slow issue-queue entries to co-exist in turn enabling instruction dispatch, issue and forwarding to proceed with minimal stalls. Evaluation on a detailed simulation environment indicates that the proposed mechanisms can reduce performance degradation due to PV to a low 1.3%.*

## Keywords

Process Variation, Issue Queue, Pipeline, Microarchitecture, IPC

## Category

Architectures, Hardware/VLSI

## 1. Introduction

The need for more computing power and smaller platforms has led the IC technology towards minuscule feature sizes. The latest chips are starting to surpass the billion transistor mark [8]. This push towards ever-increasing transistor counts has given rise to new challenges and impediments. In particular, Process Variation (PV) [8, 9] has become a major design consideration. PV arises from manufacturing imperfections resulting from sub-wavelength lithography, Random Dopant Fluctuations (RDF), dose, focus, and overlay variations [31]. The aggravation of such manufacturing uncertainties invariably leads to marked deviations in effective gate length, oxide thickness, and transistor threshold voltages [9]. Non-nominal device characteristics may lead to substantial variations in power consumption and timing violations. PV can be a systematic or a random phenomenon. While systematic variations exhibit strong spatial correlations, such that structures close to each

other are affected similarly, random variations can occur anywhere.

PV creates a widening gap between designed and manufactured circuit characteristics. This disconnect adversely affects the chip yield leading to circuit design becoming more probabilistic in future. Several researchers have already looked into the severity of the problem and proposed architectural-level and circuit-level solutions to mitigate its effect [22, 31].

The work presented in this paper focuses on one of the key components of any modern microprocessor, namely the Issue Queue. It will be demonstrated in this paper that the nominal-value deviations imparted by PV could potentially lead to severe degradation in system performance. Designing for worst-case scenarios is certainly not feasible due to its significant impact on performance.

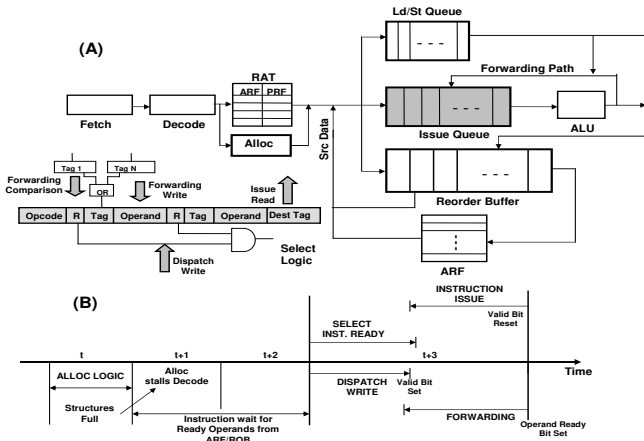
Developing effective solutions to guard against PV artifacts in issue queues is non-trivial. For instance, since the issue queue power density is high, conventional techniques such as Forward Body Biasing (FBB) [4] are not suitable for mitigating PV effects. Unlike other structures studied for variations [20], the issue queue involves multiple activities which include instruction dispatch, issue and forwarding that together make it challenging to synchronize these activities under variation.

The major contributions of this work can be summarized as follows:

- An in-depth analysis of the issue queue operation. As such, various issue queue activities that are susceptible to variations are identified and the impact of slowing them down is analyzed. We show that PV-unaware issue queue operation can lead to 20.5% performance degradation compared to a conventional PV-unaffected system.
- Mechanisms to counter variations have been provided. Since variations affect the entries in a non-deterministic manner, our techniques enable fast and slow entries within the issue queue to co-exist and thereby allow instruction dispatch, issue and forwarding operations to synchronize with each other. These mechanisms bring down the performance degradation to 5%.
- Sub-components within the issue queue entry vary in their operating speed as well. Switching source operands of instructions based on their availability is investigated. Further port-switching, whenever possible, is also shown to be effective. This sub-component analysis further reduces the performance degradation to only 1.3%, highlighting the efficacy of our mechanisms to counter variations.

Section 2 provides a background on issue queue design and associated pipeline activities. Section 3 gives the simulation platform while section 4 discusses the impact of variations with respect to the CAM and SRAM cells within the issue queue. Section 5 talks about the performance impact of slowing down different issue queue activities. Section 6 provides solutions to handle slow entries in the issue queue and reduce their performance impact. Section 7 talks about the testing methodology and the microarchitectural modifications required to support stalling. Section 8 provides the related work and finally we conclude in section 9.

## 2. Background



**Figure 1. (A) Out-of-Order Pipeline highlighting the paths to the Issue Queue. The Figure also shows an Issue Queue entry and all associated activities. (B) Time-line of issue queue activities at cycle-level granularity. Here 't' refers to a cycle. The cycles are not drawn to scale.**

Figure 1(A) shows the pipeline that we are looking at in this work, which is similar to P6-style microarchitecture [27]. Instructions are fetched and decoded in-order in the front-end of the pipeline. Register renaming occurs in the Register Alias Table (RAT), which maintain tags corresponding to the destination architected registers. The Reorder Buffer (ROB) and Physical Register File (PRF) are coupled together and hence the ROB entries correspond to PRF ids. The Architectural Register File (ARF) exists as a separate structure in which instructions retiring from the ROB write their results. Once the instructions get decoded, entries in the issue queue and ROB are allocated for the instruction in the Alloc stage in parallel with the RAT access. Once an instruction is renamed, it accesses the ARF and ROB for ready operand values before getting written into the issue queue. As soon as the instruction completes execution, it writes its results into the ROB from where they are made available to future instructions until the instruction retires. This ROB access takes two cycles [19].

Issue queues can be dispatch-bound or issue-bound, data-capture or non-data capture style [27]. In this work we look at an issue-bound, data-capture style issue queue, where the dispatched instructions fetch their ready operands from the

ARF/ROB before moving into the issue queue. A typical issue queue entry looks like the one shown in Figure 1(A) [24]. Each individual entry consists of six basic components: the opcode, two source operands, their tags, and the destination tag besides the flags that indicate whether an entry is valid and if its operands are ready. The ready operands, the opcode and the tags get written into the issue queue entries at dispatch. For non-ready operands, their tags are compared against those of instructions forwarding their results each cycle and on a match the operand value gets written into the entry. Once all operands become ready, the instruction issues a request signal to the select logic for it to be selected for execution.

Figure 1(B) shows a time-line of activities with respect to the issue queue [16]. The Alloc logic decides whether the decoded instructions can be passed to the back-end based on available entries in ROB, issue queue and load/store queue. Once entries get allocated, the instructions get written into the issue queue. For performance reasons, multiple issue queue activities proceed in a cycle. The dispatch writes of new instructions occurs in the first half of the cycle while forwarding starts in later half. This ordering is important since new instructions could get their source operands from the forwarding path. Once the forwarding match occurs, the operand is written into the entry. Once complete, it sets the operand ready bit. Since forwarded data get broadcast to all entries, the CAM cells of new instructions become effective only after the valid bit is set for these entries. This avoids any meta-stable state in the issue queue entries. Once the operands become ready, the selection and issue of the instruction occur in the next cycle which is extremely important for performance benefits.

Issue queue implementations can be compacting or non-compacting [13]. In compaction-based designs, an instruction issue causes all later entries to move forward. Any new instruction is added only to the tail of the queue implicitly maintaining the oldest-to-youngest instruction order. An alternate implementation is the non-compacting issue queue design, where dispatched instructions are allocated an entry and they remain there until issue. *Holes* created on issue get filled only by newer instructions on dispatch. Past research [13] has proposed selection logic schemes that allow oldest-to-first instruction selection for issue.

In an issue queue, the tags are usually stored in CAM arrays and the data and opcode in SRAM arrays. Prior works have looked into the problems associated with variations in SRAM [3] and CAM cells [6, 23]. Failures in the CAM include search-time, match and SRAM bit failures. Search-time failure is attributed to high  $V_{th}$  in transistors discharging the matchline, while match failure occurs when higher leakage currents cause voltage drops on the matchline. SRAM bit failures can be attributed to various conditions, such as data flipping on a read, writes not able to update cells, or  $V_{th}$  variations causing data access times to fail.

## 3. Simulation Setup

Variation analysis of the delays of all our circuits were performed using HSPICE, a circuit-level simulator. We designed

Baseline Parameters	
Parameter	Value
Fetch/Decode/Issue/Commit Width	6
Fetch Queue Size	128
Branch-Predictor	Combined Predictor
RAS Size	64
BTB Size	2K-entry 4-way
RUU/LSQ/ISQ Size	128/64/24
Integer ALUs	6 (1-cycle latency)
Integer Multipliers/Dividers	4 (3,20)
FP ALUs	6 (2)
FP Mult./Div./Sqrt.	4 (4,12,24)
1 D-Cache Ports	2
L1 D-Cache	64KB, 2-way 32B block (2)
L1 I-Cache	32KB, 2-way 32B block (2)
L2 Unified	512 KB, 4-way 32B line-size (12)
I-TLB	512-entries 4-way
D-TLB	1K-entries 4-way
TLB Miss-Latency	30 cycles
Memory Latency	150 cycles

**Table 1. Simulation parameters. Latencies of ALUs/caches are given in parentheses. All ALU operations are pipelined except division and square-root.**

all the required components and performed a delay analysis by statistically varying the device parameters. Our delay simulations employed the Predictive Technology Model (PTM) device models [32] for the 22nm technology. Architectural experiments were conducted using the SimpleScalar 3.0 [12] on the Alpha ISA. The simulator was heavily modified to support the variation-affected issue queue and the consequent stalls induced in the pipeline. The proposed techniques were evaluated on all 26 SPEC CPU2000 benchmarks after fast-forwarding to the single SimPoint [28] and running them for 100 million instructions. The parameters of our baseline model are shown in Table 1.

Since PV is a random phenomenon, it could affect any of the entries which have varying impact on performance. Looking at all possible cases would mean analyzing  $\binom{TotalEntries}{AffectedEntries}$  cases which would be large. Hence in our simulations we pick one case of PV-affected entries (entries chosen randomly) and study the performance impact across the different schemes. For low IPC benchmarks ( $IPC < 1.0$ ), their performance is determined by factors like memory accesses (mcf), branches (gcc) or floating point units (equake). In these cases, issue queue variations has less impact on overall performance than in the high IPC benchmarks ( $IPC > 1.0$ ). Hence the performance graphs give two values, the average values for all benchmarks and the average specific to the high IPC ones, in the legend. All the graphs group the high IPC (gzip to apsi - 15 benchmarks) and low IPC (swim to twolf - 11 benchmarks) together.

#### 4. Variation analysis on the key issue queue components

An issue queue consists of CAM and SRAM cells operating together in each entry. Works like [3, 6] have shown timing errors to be more dominant than bit flipping errors for SRAM and CAM cells and hence we analyze them in our work. A Monte-Carlo analysis was performed for these structures, custom-designed in 22nm technology, by simulating 5000 instances. In this work, we don't model systematic variation as the relatively

smaller size of the issue queue, compared to caches in which systematic variations are common, makes random variations the dominant cause of process variation. A variation of 5% in gate length and 10% in the threshold voltage is assumed. These values were suggested for the 65nm technology by the ITRS and widely used in the literature [3, 20]. The degree of process variation is expected to increase with newer generations [9] and thus our assumptions are conservative for the 22nm technology. Figures 2(A) and (B) show the variation in the normalized delay for match operation of a 7-bit CAM cell and write operation of a 64-bit SRAM cell respectively. Figure 2(C) shows the variation in the normalized delay of operation of an entire issue queue line. In this work, we assume the issue queue to be the frequency-determining stage as indicated in earlier work [24]. A delay of 15% greater than the nominal value (conservative guardband) was chosen to be the threshold beyond which issue queue entries require an extra clock cycle to complete operations. The maximum time required for an operation even in the worst case is within two cycles. Our results indicate the approximately 40% of the entries require the two cycles to complete operation. In our work, this translates to 10 of the 24 issue queue entries.

#### 5. The effects of PV on issue queue operation

The different pipeline activities that occur every cycle with respect to the issue queue are as follows:

- **Dispatch Writes:** On a dispatch, new instructions are written into their entries. There they wait until they are selected for execution. Dispatching instructions into the issue queue entails CAM writes and SRAM writes.
- **Forwarding Comparison/Write:** When instructions complete execution, they broadcast their results and tags to the issue queue. The dependent instructions use the tag CAM-match logic to identify if their source operands are available in the forwarding paths. Whenever these forwarding comparisons become successful, the operands get written into their entries resulting in a SRAM write.
- **Issue Reads:** When an instruction gets selected for execution, the opcode and operands of the instruction get read (SRAM read operation) from the entry and the instruction is sent for execution.

Each of these activities is important and there is significant interplay between them as well. For example, a slow dispatch write could cause forwarding operations to slow down since the tags do not get written into the entries in time. Similarly if the issue read out of an entry is slow, the destination tags will be read slowly, thus affecting the forwarding paths and any later dependent instruction in the issue queue. To understand the performance impact, we slowed down each of the components and studied the subsequent effect on performance.

Figure 3 shows the results of this experiment. We show each of the activities in descending order of performance impact and compare it against the issue queue performance when there is no process variation - NonPV. In certain cases, slow dispatch

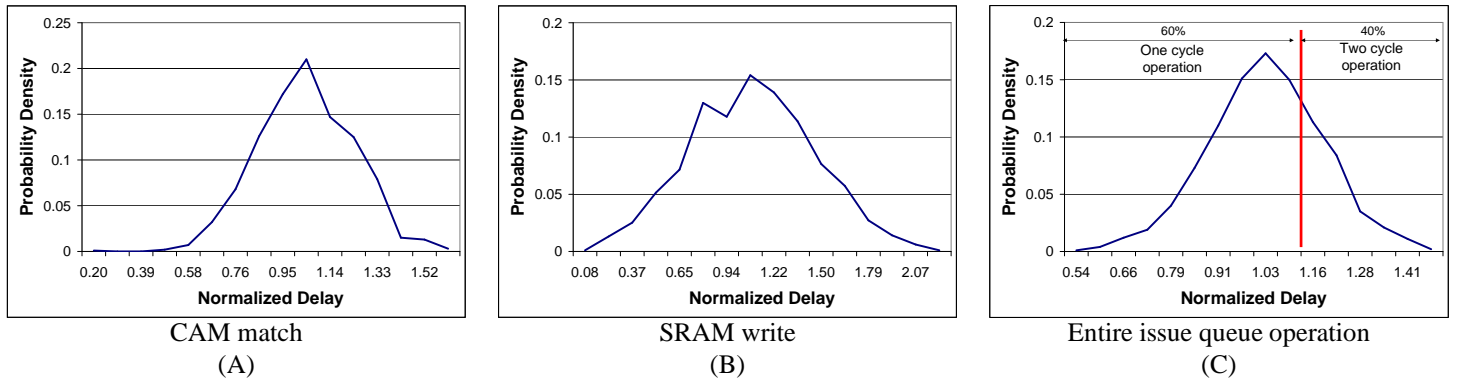


Figure 2. Delay variation in issue queue. (C) shows that 60% of entries operate in 1 cycle while 40% of entries require 2 cycles.

brings down performance significantly because more instructions are needed to exploit the ILP (mesa, bzip2) while in others the issue queue holds enough instructions to meet IPC requirements and hence slowing down forwarding and issue brings down performance (gzip, sixtrack). Figure 3 shows that, across all benchmarks, each of these activities play a vital role in issue queue operation. Hence our variation-tolerant solutions have addressed them together rather than concentrating on one specific activity. Note that slowing down all the activities together, *Slow Dw/Ir/Fw*, has the biggest performance impact, as much as 20%. Our solutions in section 6.2 is motivated based on this observation.

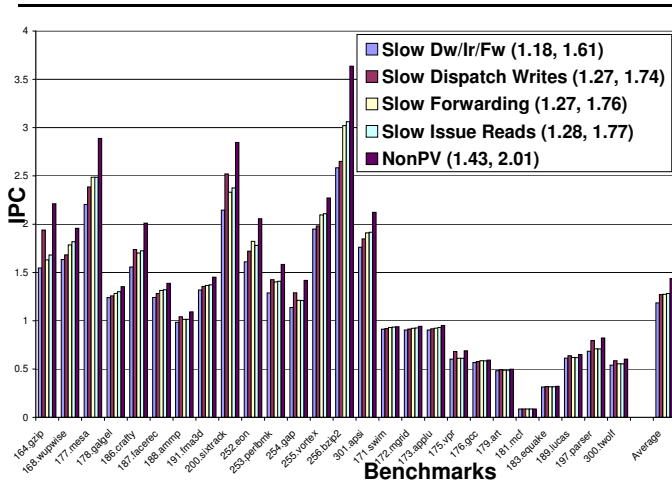


Figure 3. Performance Impact of slowing down different activities that occur in the Issue Queue. Legend shows the average IPC across all benchmarks and the high IPC ones.

**Variation Analysis in Collapsible Issue Queues:** Besides the above-mentioned activities, collapsible issue queues also have additional movement of instructions between neighboring entries. Variations affecting the data movement logic has a significant impact on performance. This is because there is linear dependency in instruction movement between entries. Variations causing slow movement between any 2 entries would

affect all entries after them (slowing them as well), bringing down performance. Since variations is a non-deterministic phenomenon, it is impossible to design the instruction movement logic to minimize impact of variations. Further, [13] shows that the power consumption of a collapsible issue queue is considerably higher than a non-collapsible queue; thus, the latter is more preferable in modern power-aware designs. Hence, we limit our investigation to the impact of process variation in non-collapsible queues.

The select logic also plays an important role in issue queue operation. In non-collapsing issue queue, the select logic is a linear chain of multiplexers [13]. Hence it is less affected by variations compared to the issue queue itself [10]. Also a slow select logic operation affects only the issue reads. Hence it can be modeled as issue read variations. But given the importance of select logic, it warrants further investigation and it is very much part of our future research.

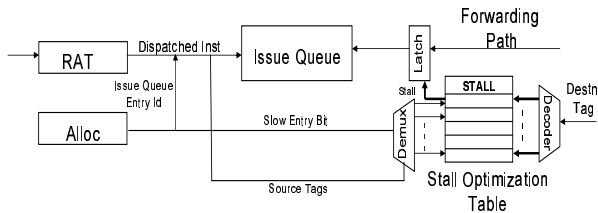
## 6. Battling PV artifacts within the issue queue

Our solutions to handle issue-queue variations progressively move from tackling variations at per-entry level to sub-component analysis. Initially we look at steering schemes that dispatch instructions to entries based on operating-speeds. Optimizing on these schemes, our intra-entry variation analysis looks at switching operands and ports to further mitigate the impact of variations. Towards the end of the section, we also look at how existing solutions, proposed to handle other phenomena, can be tuned for handling variations.

### 6.1. VariSteer: PV-aware instruction steering

Naive PV-unaware issue queue allocation reduces overall performance by about 20%. Since variation is non-deterministic, entries could either be fast or slow. In this section, we look at steering schemes that optimize entry allocation based on operating speeds of the entries to minimize stalls reducing the performance degradation to 5%. The alloc logic maintains the information regarding the slow entries and steers instructions accordingly.

**6.1.1. SpeedSteer: A Speed-aware steering mechanism** Dependent instructions within the issue queue can lie



**Figure 4. Microarchitectural changes required to implement OptiSteer.**

in faster or slower entries. If the instructions lie in slower entries, forwarded results need to be held for an additional cycle in order for slower entries to pick up the data. Since the pipeline does not know if the dependent instructions are lying in faster or slower entries, forwarding can be done every other cycle only bringing down performance greatly.

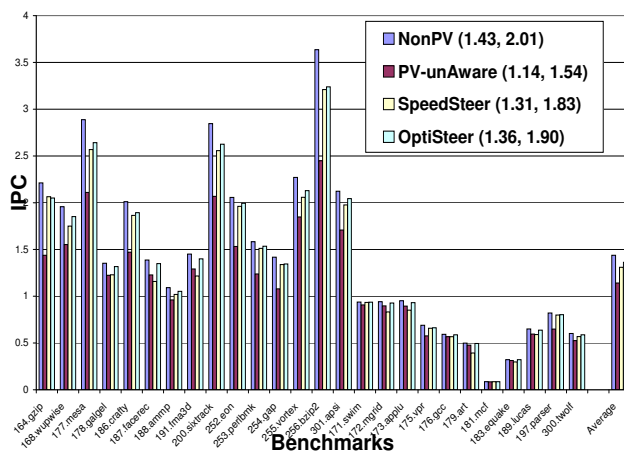
Our proposed SpeedSteer scheme reduces this by maintaining any dependency chain within faster or slower entries. By doing this, the instructions implicitly know that their dependent instructions also move to similar entry. Hence only instructions in slow entries stall the forwarding paths. When the source operands lie in both type of entries, instructions can be dispatched to faster entries only. In such cases, the forwarding paths might be stalled unwantedly by the instruction in slow entry but this was minimal since the instruction could have other dependents in slow entries.

To identify whether the instruction issued from a fast or slow entry, the instruction carries a bit after issuing. This bit can be added to the entries and set on dispatch – negligible change to variation effects due to adding the bit (size of an entry is 160 bits). Alternatively, the select logic can set this bit while issuing instructions, based on the entries from which they were issued.

**6.1.2. OptiSteer: An optimized table-based steering mechanism** Simple priority allocation of instructions to fast entries (slow entries used only when fast entries are not available) is not possible with SpeedSteer. This is because there is no mechanism to stall forwarding of instructions that issued from fast entries for an additional cycle when their dependents lie in slow entries. To enable this, OptiSteer uses a table called the Stall Optimization Table (SOT) whose entries are set, when slow dependents are in the issue queue, to stall forwarding.

The SOT has 128 1-bit entries, corresponding to the destination tags (ROB id) of each instruction. An entry is set if instruction with corresponding source operand tag is in a slow issue queue entry. When an instruction completes execution, it accesses the table to check if forwarding needs to be stalled for next cycle. Figure 4 shows how the SOT stalls the forwarding path based on the source operand tags of instructions. Note that the SOT access is not on the critical path since the value is required not in the current cycle but the next one. Figure 13 shows this in greater detail. Given that SOT is a small structure (small access time as well) and its access is not on a critical path, there are no timing issues due to variations affecting it.

Figure 5 shows the performance benefits in employing the two instruction steering schemes. The conventional scheme shown as *PV-unAware* is oblivious to variation-affected entries and suffers an overall performance loss of 20.5%. For high IPC benchmarks, this value goes upto 24%. The SpeedSteer scheme in turn exhibits 8.8% (9.2% for high IPC benchmarks) overall performance loss, while the OptiSteer scheme imposes only 5.1% (5.5% for high IPC benchmarks) performance overhead. In certain cases like bzip2, sixtrack and mesa, the *PV-unAware* scheme leads to 34% performance degradation while the steering schemes have only about 10% performance loss in those cases. The steering schemes do a better job of utilizing the faster entries compared to *PV-unAware* scheme. Both the steering schemes place about 80% of instructions in fast entries compared to *PV-unAware* scheme that places only 57% of instructions in the fast entries.



**Figure 5. Performance obtained by employing the instruction steering schemes. Legend shows the average IPC across all and high IPC benchmarks clearly showing the requirement for the steering schemes to reduce performance loss due to variations.**

## 6.2. Intra-Entry variations

Each entry of the issue queue consists of multiple sub-components that differ in their functionality (a tag CAM sub-component and operand SRAM sub-component). Naturally, random variations could cause variability in their behavior. Based on this fact and the observation that all sub-components of an entry need not be useful for every instruction, we provide techniques to avoid variation-affected sub-components whenever possible. Random variability also means that performance of sub-components do not get adversely affected with respect to all the ports used to read or write into the entries. Since the maximum bandwidth is not always required, ports can be switched whenever possible.

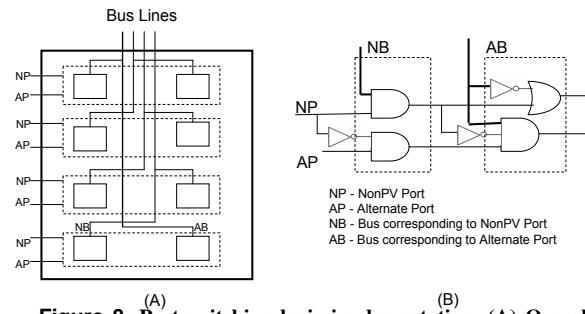
Exploiting varying port-speeds would allow entries to have fast dispatch writes while their issue reads could be slow. By identifying this, unwanted stalls could be reduced. The conventional issue queue design allows these operations to be of varying speeds and yet interface with each other. Figure 1(B) show-



**Implementation** Even though there are potential advantages to port-switching at dispatch, issue, and forwarding, the specific bus on which the data gets forwarded is decided at issue time by the select logic. The select logic does not know dependents of an instruction. Also, the multiple dependents of an instructions could operate fast or slow with respect to a port since forwarded data get broadcast. Based on these observations we opt not to implement port-switching for data forwarding.

Conventionally, all buses operate at the same speed, and their assignment to instructions is inconsequential. With variations, the bus assignment is non-trivial if stalls are to be minimized.

For dispatch writes, the port assignment can be done in the alloc stage. Since the specific entries that the instruction would be allocated is known, port speeds for those entries would be known as well. This information is used to make the port-assignment. For issue reads, the port speeds of individual en-



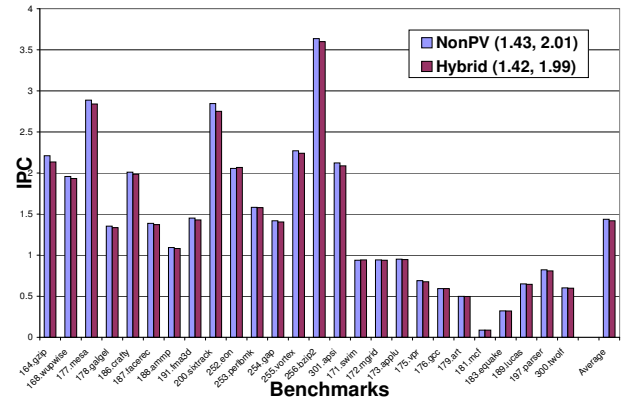
**Figure 8. Port-switching logic implementation. (A) Overall connections between buses available and port speed lines. (B) Logic inside each of boxes assigning either one of the buses based on availability and port speeds.**

tries is maintained in the selection logic. Once an instruction is chosen for issue, the selection logic determines the specific port for the instruction. Note that all six ports are not available for the instructions. Since each port interfaces to specific execution units, port-switching can be done only when instruction can be issued to multiple functional units. Hence we restrict port-switching between two alternate ports. There is one port, *NonPV Port (NP)*, that the instruction would have normally take in a variation-unaffected pipeline. Since this could turn out to be slow, we also check if an alternate faster port, *Alternate Port (AP)*, is available. If it is unassigned to other instructions, the alternate port is taken. Figure 8(A) shows the bus and port interconnections required to implement port-switching in parallel, shown for a four-ported system. Figure 8(B) shows how the actual assignment is done. Note that a '0' indicates a slow port and a non-available bus while a '1' indicates a faster port and an available bus.

The port-switching logic is on the critical path of instruction issue. To incorporate it, the select logic for non-collapsing issue queue involving four stages to select instructions based on their age, is shrunk to three stages. This reduction in select logic makes it less effective to implement oldest-first selection

but port assignment was found to have a more significant impact on performance.

Figure 9 shows the impact of combining OptiSteer steering scheme with port and operand switching, shown as *Hybrid ISQ*. Steering is effective since it avoids entries that have predominantly slow ports and hence cause more stalls in pipeline while port and operand-switching are effective when only a lesser percentage of an entry or sub-component are affected by variations. As we note from the figure, the variation affected issue queue now operates within 1.3% of variation-unaffected issue queue. Even for high IPC benchmarks, the loss is only 1.5% clearly showing the capability of our schemes to operate effectively under variations.



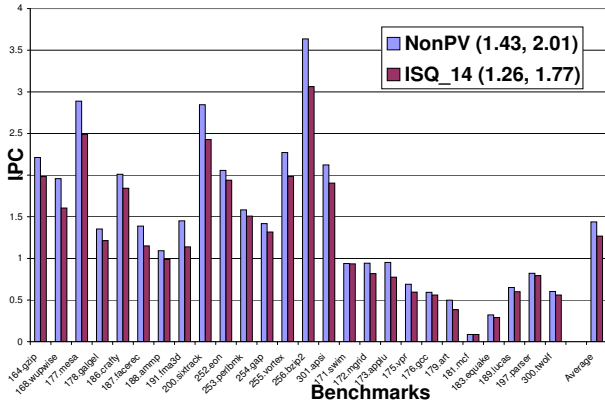
**Figure 9. Performance impact of combining steering with operand and port-switching. Legend clearly shows that across all benchmarks and high IPC ones, the hybrid scheme does well.**

### 6.3. Existing mechanisms for handling variations

This section deals with how some of the existing solutions, proposed for other purposes related to the issue queue, can be tweaked for variation handling as well.

Shutting down issue queue entries based on utilization has been traditionally used to maximize the power/energy savings in a system [1]. Since it involves no significant additional effort, entries that violate the timing requirements can be shut down. Figure 10 shows the performance impact of such a system. *ISQ\_14* indicates the issue queue with 14 entries operating at the desired frequency and rest shutdown. As can be noted from the graph, the performance degradation is 12%, growing upto 16% for bzip2 and sixtrack, compared to the conventional issue queue. Given that variations are only increasing with newer technologies and that in a multi-threading scenario the issue queue's utilization increases, shutdown is not a viable option.

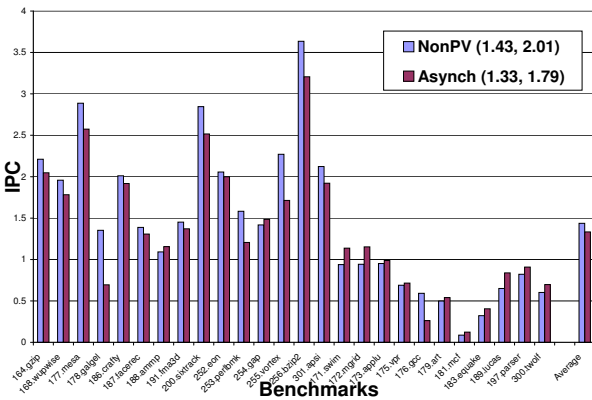
Instead of merely shutting down the issue queue entries, an alternate solution we envision is a multi-clock domain (MCD) asynchronous environment. Solutions in the past have looked at operating the issue queue asynchronously with the rest of the pipeline [26] and adapting the issue queue size [1] for reducing the dynamic power and/or energy. By operating the is-



**Figure 10. Performance Degradation due to shutting down variation-affected entries.** Average values across all benchmarks and high IPC ones show that there is significant performance loss with shutting down entries.

sue queue as a fast 14-entry queue or a slow 24-entry queue (incorporating the slow entries as well) and switching between them based on application runtime IPC, we looked at mitigating performance impact of variations. Figure 11 illustrates that the MCD-solution leads to performance degradation of about 7.3%.

Compared to both these solutions, our mechanisms are clearly more effective in reducing the performance impact of variations.



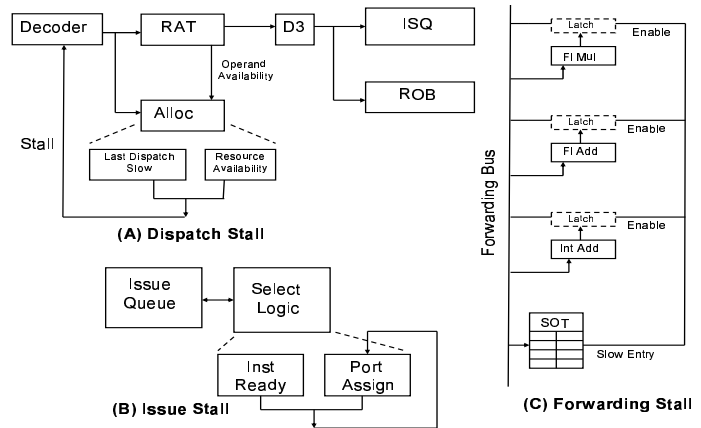
**Figure 11. Performance obtained by operating the Issue Queue in the two modes in a MCD microarchitecture.** The average values show that this might not be an effective solution given the performance loss incurred.

## 7. Epilogue to PV-aware issue queue design

### 7.1. Microarchitectural support for pipeline stalling

Despite all the proposed optimizations for avoiding stalls due to slow entries, there can be scenarios where the pipeline has to be stalled to accommodate the operation of slow entries.

Figure 12 demonstrates how stalling is accomplished due to different issue queue activities. What makes the problem under investigation non-trivial is the fact that an instruction could stall the pipeline when (1) its opcode write during dispatch is slow, or (2) its operand read during an issue read is slow. Thus, the stall logic should keep track of these factors.



**Figure 12. Implementation of the stall logic in the different issue queue operations.**

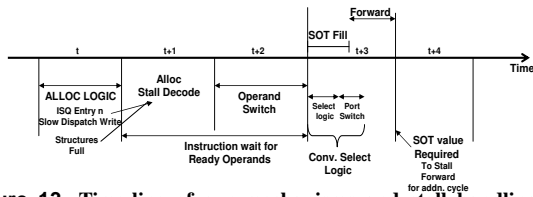
For the dispatch stage, we identify whether stalling is needed once renaming is done. Operand availability is passed from the renaming stage to the alloc stage. Since the alloc stage knows the port availability and speed of operation for different entries, it decides whether the next set of instructions need to be stalled. The logic required for stalling the decode stage is already present in a conventional pipeline when entries in ROB, issue queue or load/store queue are not available. In the issue stage, the conventional select logic already keeps track of port availability which it uses for selecting instructions to issue. For slow issue reads, the specific ports would not be available for an additional cycle.

Forwarding stalls are handled using SOT. For handling the stall signals in the execution units, we use the implementation proposed by Hans Jacobson [17], called Elastic Synchronous Pipeline, which uses the ability of master/slave latches to store two distinct values under stall conditions, while storing only one data element under normal operation.

Figure 13 shows a time-line incorporating all our mechanisms. The figure clearly indicates that our solutions are mostly not on the critical path and in places they are, appropriate pipeline modifications have been made.

### 7.2. Variation testing methodology

Built in self tests (BIST) aware of variations are gaining importance. Variation-aware testing strategies evolved for identifying slow SRAM entries hold good for identifying slow issue queue entries as well [29, 2]. Also works like [5] look at identifying the target gates that are variation-affected and propose techniques to generate test patterns. These methodologies can be applied for the variation-affected issue queue to obtain



**Figure 13.** Time-line of our mechanisms and stall handling at cycle-level granularity. Here 't' refers to a cycle. The cycles are not drawn to scale.

maximum performance. Since the different activities associated with the issue queue have varying operation speeds with respect to an entry, this information is maintained at different stages of pipeline for effective operation. Issue queue BIST identifies the 40% of entries that are affected by variations and hence are slow. Using this information in the Alloc stage would enable the steering schemes to operate. Further, by identifying the port-speed information with respect to the different activities we allow instructions to port-switch. Since port-switching is done both at dispatch and issue, the port-speeds are maintained in a ROM in both the Alloc logic and the select logic.

## 8. Related Work

Process variation has been identified as a major hurdle in the coming technology generations. The International Technology Roadmap for Semiconductors (ITRS) has shown a lack of predictability in several aspects of physical design. The intensity of this problem is going to be further aggravated as feature sizes diminish. It has been observed that the loss in performance due to PV can be equal to the gain by one full technology generation [10]. Bernstein [7] presents a survey of process variation issues. Unsal et al. [31] classify process variation based on source, granularity, manifestation, design parameter and aging. The unpredictability in design due to PV manifests as both random and systematic variations [3]. Lack of predictability in timing characteristics leads to a loss in yield. Strong economic considerations for yield have motivated research at both the circuit and architecture levels to address this problem. PV can also lead to variation in the power consumption of circuits. Borkar et al. [9] show that variation in power consumption can be as high as 20X.

Recent works on process variation have focused on different components such as the register file [20], cache [2] and in redesigning latch elements [15]. Our work is concurrent to [25] which looked at a subset of the issues addressed in this work. An alternate approach is cycle stealing, which allows a PV-affected stage to borrow slack from other stages [30, 20]. While cycle-stealing holds lot of promise in designing variation tolerant circuit, it increases the design complexity and absorbing clock jitter and skew becomes difficult [21]. By removing the variations in issue queue, our work enables the slack to be available for other pipeline stages.

Previous research have looked at breaking the issue queue into multiple structures to make it more scalable [11]. These solutions provide multiple queues operating at different speeds

into which instructions are moved based on various conditions. Note that these solutions are built on the fact that the architect has design-time knowledge of the relative operating speeds of entries. The very fact that variations are non-deterministic at design-time makes our problem unique and challenging. Now the fast and slow entries have to co-exist and all the activities must proceed correctly.

Port-switching [20] has been previously studied in the context of register file reads. Here we apply it for the multiple issue queue activities. Prior works [18, 14] have looked at the fact that even though the issue queue is designed to support the worst-case two-operand instructions in all its entries, that might not always be needed. These solutions use their observations to optimize the energy dissipation due to the wakeup logic and hence took a performance hit. Our goal, though, is to reduce the performance impact of the slow entries. Further our techniques take to account the non-determinism due to variations as well.

## 9. Conclusion and Future work

The unrelenting march towards diminutive feature sizes in the deep sub-micron regime has accentuated reliability concerns in modern digital designs. Process Variation is an emerging threat that can adversely affect both performance and power consumption.

The work presented in this paper investigates the effects of process variation on the issue queue which, to a large extent, determines overall pipeline throughput. Through a detailed analysis of all major sub-components, we identify and quantify the impact of variability on the issue queue to be about 20.5% compared to PV-unaffected issue queue. We demonstrate that solutions targeting individual issue queue operations in isolation are not effective. Hence, we adopt a holistic approach and provide comprehensive solutions that reduce the impact of variations in all pipeline activities associated with the issue queue. The proposed solutions cohesively operate the slower and faster entries of the issue queue in unison, and dynamically optimizes pipeline stalls bringing down the performance degradation to a mere 1.3%.

As part of our future work, we are investigating the impact of variations on alternate issue queue designs and the select logic.

## Acknowledgements

We would like to thank the anonymous reviewers whose detailed comments helped improve the quality of the paper. This research was funded partly by SRC's GSRC Focus Center and NSF grants 0615097, 0621429, 0454123, 0702617.

## References

- [1] J. Abella, R. Canal, and A. Gonzalez. Power- and complexity-aware issue queue designs. volume 23, pages 50–58. IEEE Computer Society, 2003.
- [2] 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 (VLSI) Systems*, 13(1):27–38, January 2005.
- [3] A. Agarwal, B. C. Paul, S. Mukhopadhyay, and K. Roy. Process variation in embedded memories: failure analysis and variation aware architecture. *Solid-State Circuits, IEEE Journal of*, 40:1804–1814, 2005.
- [4] H. Ananthan, C. H. Kim, and K. Roy. Larger-than-vdd forward body bias in sub-0.5v nanoscale cmos. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, pages 8–13. ACM Press, 2004.
- [5] D. Arumí-Delgado, R. Rodríguez-Montanés, J. P. de Gyvez, and G. Gronthoud. Process-variability aware delay fault testing of "vt and weak-open defects. In *ETW '03: Proceedings of the 8th IEEE European Test Workshop*, page 85, Washington, DC, USA, 2003. IEEE Computer Society.
- [6] M. Bennaser and C. A. Moritz. Power and failure analysis of cam cells due to process variations. In *ICECS '06: 13th IEEE International Conference on Electronics, Circuits and Systems*, pages 608 – 611, 2006.
- [7] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM J. Res. Dev.*, 50(4/5):433–449, 2006.
- [8] S. Borkar. Microarchitecture and design challenges for gigascale integration. In *MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–3. IEEE Computer Society, 2004.
- [9] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 338–342. ACM Press, 2003.
- [10] 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. *Solid-State Circuits, IEEE Journal of*, 37(2):183–190, February 2002.
- [11] E. Brekelbaum, J. Rupley, C. Wilkerson, and B. Black. Hierarchical scheduling windows. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 27–36. IEEE Computer Society Press, 2002.
- [12] D. Burger and T. Austin. The SimpleScalar Toolset, Version 3.0. <http://www.simplescalar.com>.
- [13] A. Buyuktosunoglu, A. A. El-Moursy, and D. H. Albonesi. An oldest-first selection logic implementation for non-compacting issue queues. In *15th Annual IEEE International ASIC/SOC Conference*, pages 31 – 35, 2002.
- [14] D. Ernst and T. Austin. Efficient dynamic scheduling through tag elimination. In *ISCA '02: Proceedings of the 29th annual international symposium on Computer architecture*, pages 37–46. IEEE Computer Society, 2002.
- [15] D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Micro Conference*, December 2003.
- [16] K. Ghose. Reducing energy requirements for instruction issue and dispatch in superscalar microprocessors (poster session). In *ISLPED '00: Proceedings of the 2000 international symposium on Low power electronics and design*, pages 231–233. ACM, 2000.
- [17] H. M. Jacobson. Improved clock-gating through transparent pipelining. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, pages 26–31. ACM Press, 2004.
- [18] I. Kim and M. H. Lipasti. Half-price architecture. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 28–38. ACM Press, 2003.
- [19] G. Kucuk, D. Ponomarev, and K. Ghose. Low-complexity reorder buffer architecture. In *ICS '02: Proceedings of the 16th International Conference on Supercomputing*, pages 57–66. ACM Press, 2002.
- [20] X. Liang and D. Brooks. Mitigating the impact of process variations on processor register files and execution units. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 504–514. IEEE Computer Society, 2006.
- [21] S.-Z. E. Lin, C. Changfan, Y.-C. Hsu, and F.-S. Tsai. Optimal time borrowing analysis and timing budgeting optimization for latch-based designs. *ACM Trans. Des. Autom. Electron. Syst.*, 7(1):217–230, 2002.
- [22] D. Marculescu and E. Talpes. Variability and energy awareness: a microarchitecture-level perspective. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 11–16. ACM Press, 2005.
- [23] A. Mupid, M. Mutyam, N. Vijaykrishnan, Y. Xie, and M. J. Irwin. Variation analysis of cam cells. In *ISQED '07: 8th International Symposium on Quality of Electronic Design*, pages 333 – 338. IEEE Computer Society, 2007.
- [24] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *SIGARCH Comput. Archit. News*, volume 25, pages 206–218. ACM Press, 1997.
- [25] K. Raghavendra and M. Mutyam. Process variation aware issue queue design. In *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, 2008.
- [26] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 356–367. IEEE Computer Society Press, 2002.
- [27] J. Shen and M. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors (Beta Edition)*. McGraw Hill, 2003.
- [28] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. *SIGOPS Oper. Syst. Rev.*, 36(5):45–57, 2002.
- [29] 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.
- [30] A. Tiwari, S. R. Sarangi, and J. Torrellas. Recycle: Pipeline adaptation to tolerate process variation. In *ISCA '07: Proceedings of the 30th annual international symposium on Computer architecture*, 2007.
- [31] O. S. Unsal, J. W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin. Impact of parameter variations on circuits and microarchitecture. *IEEE Micro*, 26(6):30–39, 2006.
- [32] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45nm design exploration. In *ISQED '06: Proceedings of the 7th International Symposium on Quality Electronic Design*, pages 585–590. IEEE Computer Society, 2006.