VAIL: Variation-Aware Issue Logic and Performance Binning for Processor Yield and Profit Improvement

Somnath Paul
Case Western Reserve Univ.
Cleveland, Ohio, USA
sxp190@case.edu

Swarup Bhunia
Case Western Reserve Univ.
Cleveland, Ohio, USA
skb21@case.edu

ABSTRACT

With increasing parameter variations, functional units (FUs) in a chip experience considerable local variations in maximum operating frequency. Effect of such within-die variations in a superscalar processor if addressed by worst-case frequency assignment, results in overly pessimistic yield in high-frequency bins. In this paper, we propose *VAIL* - a novel low-overhead instruction scheduling strategy that assigns best-case frequency by issuing the narrow-width (NW) operations to slower units. This exploits the abundance of NW operations (>70%) in a typical program and the fact that the critical path in FUs are not activated for these operations. Compared to existing vari-cycle approach, the proposed scheme demonstrates a large improvement in yield ($\sim$ 27% at highest performance bin) and profit (10-15%) for a set of benchmark applications. It also improves the thermal profile for the FUs. Finally, it provides large opportunistic power saving ($\sim$ 43%) in the slow units using supply gating of inactive bit-slices.

Categories and Subject Descriptors

B.2.3 [**Hardware**]: Arithmetic and Logic Structures—*Reliability and Fault Tolerance*

General Terms

Design, Reliability

Keywords

Within-die variation, Superscalar processor, narrow-width operand

1. INTRODUCTION

Die-to-die and within-die (WID) parameter variations have emerged as major design concerns in the sub-100nm technology regime. For modern processors fabricated using nanoscale CMOS technologies, such variations cause large spread in the frequency of operation [2], which, in turn, affects the parametric yield and revenue of a product. With aggressive technology scaling, both die-to-die and WID variation effects are increasing. Effects of such variations increase as the logic depth in processor pipeline stages decreases with processor generation [1]. A major impact of such WID variations is that, identical FUs inside a given processor will experience local variations in circuit delay. Conventional wisdom dictates assigning the worst-case frequency of the FUs to the processor in order to avoid delay failure. However, such an approach is overly pessimistic [1]. It is, therefore, important to explore low-overhead circuit/micro-architecture level design solutions such that processor yield and revenue can be maximized under large WID variations.

In a typical heterogeneous processor pipeline, the effect of WID variations is likely to be more pronounced in delay-critical stages, namely, integer and floating point execution units and wake-up logic [9]. Critical delay in execution units or functional units (FUs) is logic dominated [4] while that in wake-up logic is wire-dominated. Hence, the execution units are more vulnerable to inter-die and WID parameter variations [4]. Recently, a number of approaches have tried to adapt to local variations by scaling either global [7] (e.g. voltage/frequency) and/or local parameters (e.g. operational latency) [1, 4]. However, these techniques incur significant design modifications (e.g. redesign of FUs) or hardware overhead due to the adaptation hardware [1, 7]. Under-utilization of the slower FUs also leads to throughput degradation for high WID variation scenarios [4].

In this work, we propose a novel, WID variation-aware instruction scheduling strategy, referred as *VAIL*, which exploits the abundance of narrow-width (NW) operands in typical programs and the operand width dependence of critical delay in FUs, and improves the throughput in a modern superscalar processor under severe WID variations. The

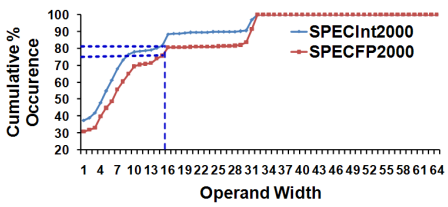

Figure 1: Bitwidths for SPECInt2000 and SPECFP2000 benchmarks compiled for 64-b Alpha.

ISLPED'10, August 18–20, 2010, Austin, Texas, USA.

basic idea is to assign the best-case frequency of the FUs (instead of the worst-case one) as the processor operating frequency and then modify the instruction scheduling policy such that operations with varying width are matched to the delay of the FUs so that none of the FUs suffer from delay failure. We note that the critical delay of a FU typically depends on the operand width - e.g. a 16-bit add/mult operation will have considerably less delay than their 64-bit counterpart. Such delay difference ensures that under large WID variations, slow units can perform correct operations for narrow-width (e.g. 16-bit or 8-bit) operations while fast units can work reliably for operations of any width.

Compared to the worst-case frequency assignment or the vari-cycle approach [4] (where slow units run at increased latency), the proposed approach improves performance by allowing the slow units to run at the frequency of the fast units without requiring to increase latency. If in a given window, the maximum operand width for the set of ready instructions exceed the number of FUs which can support the max operand width, ready instructions are delayed until FUs which can support the relevant operand size, become free. Our simulations demonstrate that due to abundance of NW operands in general purpose applications, the performance degradation with the proposed *VAIL* approach is negligible compared to the worst-case frequency assignment and even less when compared to the vari-cycle approach [4]. Such frequency allocation along with the scheduling policy can result in varying performance for the same frequency depending on the distribution of FU frequencies. This gives rise to the concept of *performance binning* instead of the traditional frequency binning, where the chips are binned in terms of their actual throughput.

In particular, the paper makes the following contributions:

1. It proposes a novel instruction issue approach which exploits the abundance of NW operands and width-dependence of delay of execution units to improve processor performance under large WID variations. The proposed approach assigns full-width operations to fast FUs and NW operations to slow units, thereby allowing the slow units to operate at normal latency.

2. It presents a detailed hardware scheme that allows width dependant FU selection. It also proposes appropriate design steps to increase the delay difference between narrow-width and full-width operations.

3. It also presents a novel binning approach based on performance instead of frequency. With simulation results for realistic processor and variation models, it shows that the *VAIL* approach along with effective performance binning can significantly increase the high-frequency yield and profit under large WID variations. It can also achieve opportunistic energy saving by supply gating the inactive parts of a slow unit.

4. It compares the energy and thermal profile of the FUs between *VAIL* and the vari-cycle approach [4] and shows that the proposed approach avoids hotspot formation by evenly distributing load across all FUs.

2. BACKGROUND AND MOTIVATION

Existing solutions for tolerating the effect of parameter variations in processor logic can be classified into two broad types:

1. Adaptation techniques such as *Razor* [7] control global parameters such as voltage/frequency for mitigating the effect of variation. Razor [7] uses dynamic detection and correction of circuit timing errors to adjust the supply voltage, which potentially eliminates the requirement of delay margin during design phase. Such adaptation schemes, however, do not account for the local delay distribution for the FUs. Moreover, they require an expensive recovery phase when error occurs.

2. Techniques that use local parameters for adaptation can be subdivided into dynamic and static approaches. Dynamic approaches typically check the inputs to the FUs at run time for possible delay failure conditions. If such conditions are satisfied, the execution unit is assigned two-cycles for correct computation of its results [3]. They are therefore commonly referred to as the vari-cycle approach. However in vari-cycle approach, both issue latency (*issue_lat*) and operational latency (*op_lat*) are doubled which degrades processor throughput. Moreover, it demands extra clock gating logic [4]. Static approaches use adaptive body biasing (ABB) techniques [1] for delay correction. The effectiveness of ABB has however been observed to decrease with technology scaling.

In this paper, we propose a low-overhead dynamic instruction scheduling approach, which exploits the WID delay distribution among the FUs. In order to evaluate the effect of inter-die and WID variation on the FU delays inside a superscalar processor, we modeled the FU delay using a 9-stage FO4 delay model similar to [4]. This model was simulated for PTM 45nm technology model [6] considering $\sigma_{inter-die} = 25\%$, $\sigma_{WID-systematic} = 15\%$, $\sigma_{WID-random} = 10\%$ for 1000 instances of the same chip. The processor model was adopted from [8]. Based on Monte Carlo runs, 1000 instances of the chip were divided into 5 frequency bins where the center frequency changes by 15% of the nominal FU operating frequency. However, from our simulations, we note that the σ/μ of the FU delay in each bin can be as high as 18.42%.

3. EXPLOITING NW OPERANDS FOR DELAY FAILURE AVOIDANCE

It has been observed that for general-purpose benchmark applications, majority of the operands are narrow-width [5], which means the operand sizes are limited to 8 or 16-b. Although the observation was made for SpecInt95 benchmark suite [5], we note that the same trend persists in more recent SpecInt2000 and SpecFP2000 benchmarks. As shown in Fig. 1, almost 81% and 75% of the operands in SPECInt2000 and SPECFP2000 benchmark suites have bit-widths of 16-b or less. Due to large delay slack for NW operations, a 64-b ALU operating on narrow operands does not suffer delay failure even under large variations. Different FU architectures which exploit NW operands for power/performance improvement has been widely researched [5]. However, exploiting this diversity for tolerating WID variation in FUs remains unexplored.

3.1 Adder and Multiplier designs

A 64-b parallel prefix integer adder was synthesized using Synopsys Design Compiler. A 2-stage pipelined 32x32 integer multiplier was also synthesized and their delays were noted for a 45-nm technology library. Fig. 2(a) shows

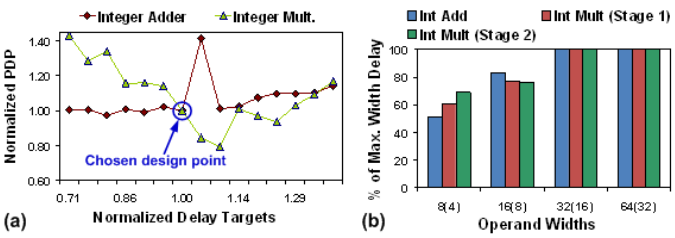

Figure 2: a) Normalized PDP and delay target values for Integer Adder and Multiplier designs; b) Delay through the Integer adder and multiplier units for input operands of different sizes.

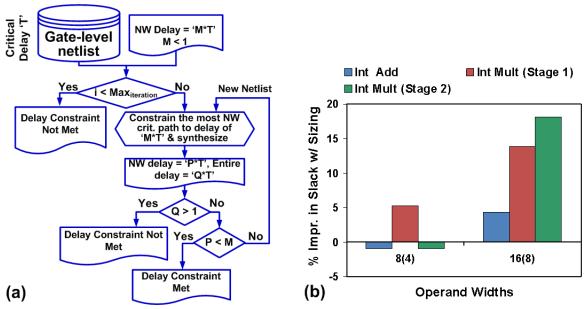

Figure 3: a) Design methodology using gate resizing in order to improve slack for NW operands; b) Improvement in slack for NW operands due to gate sizing.

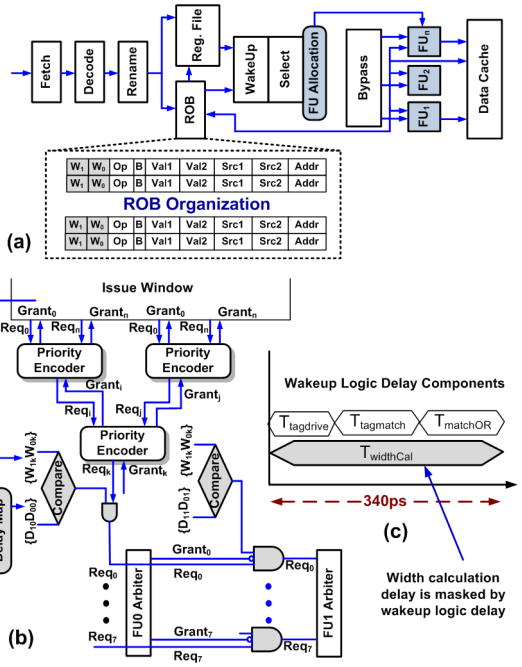

Figure 4: a) Integration of the FU Select Logic in the superscalar processor pipeline; b) Detailed implementation of the Select Logic; c) Delay for operand width calculation can be masked by the wakeup logic delay components.

the power-delay-product (PDP) distribution for the adder and the multiplier for delay targets normalized to a value of 500ps. Fig. 2(b) shows the delay distribution for varying $OpWidth_{MAX}$ where $OpWidth_{MAX}$ denotes the maximum of the widths for the two operands at the input of the FU. For the integer adder, widths 1-4 denote operand size of 8-b, 16-b, 32-b and 64-b. For the 2 stages of the pipelined multiplier, widths 1-4 stand for 4-b, 8-b, 16-b and 32-b respectively. From Fig. 2(b), we note that delays for $OpWidth_{MAX}$ = 8-b and 16-b are only 51% and 83% of the delay required for $OpWidth_{MAX}$ = 64-b respectively. Similarly for stage 1 of the multiplier, delays for $OpWidth_{MAX}$ = 4-b and 8-b are 60% and 77% respectively. For stage 2 of the multiplier, the values are 68% and 76%. From the above values we note that there exists considerable delay margin in the computation of NW operands.

The delay critical sections of the FU-adder and multiplier comprise of 23-b and 52-b integer add and multiply operations for single and double precision input operands. Due to large logic delays, these sections are most likely to suffer from variation induced delay failures. Hence any asymmetry in the computation time for different operand sizes can be exploited at these stages of the FP FUs.

3.2 Increase in slack for Narrow-Width operands using gate sizing

Selective gate sizing can be applied to minimize the delay for narrow-width operations (8-b or 16-b) without sacrificing the delay for wide-operands (64-b). For the integer adder, worst-case timing path for 16-b operations was separately constrained so as to have a delay slack of more than 20% compared to the 17% obtained from the previously synthesized design. The design was then re-synthesized with this new constraint. For the revised design, delay for 8-b operation was observed to increase from 51% to 52% whereas the delay for 16-b operation improved from 83% to 79%. The methodology adopted to re-size the synthesized netlist to improve the slack for NW operands is shown in Fig. 3(a). Corresponding improvements for adder and multiplier units with 8-b and 16-b operands are shown in Fig. 3(b).

4. VARIATION-AWARE INSTRUCTION SCHEDULING

4.1 Width-aware Instruction Issue Policy

Under severe WID variation, the worst-case delays for the 6 integer ALUs in our baseline processor can be considerably different from one another. Let us assume that the worst-case delays post-fabrication are $0.9 \times T_o$, $0.95 \times T_o$, $1.0 \times T_o$, $1.04 \times T_o$, $1.1 \times T_o$ and $1.12 \times T_o$, where T_o denotes the worst-case delay prior to fabrication. Clearly, the ALUs with delays $0.9 \times T_o$, $0.95 \times T_o$ and $1.0 \times T_o$ will be able to support operations with $OpWidth_{MAX}$ = 64-b, while ALUs with delays $1.04 \times T_o$, $1.1 \times T_o$ and $1.12 \times T_o$ can effectively support 8-b or 16-b operations. We therefore propose a preferential instruction issue policy which assigns NW operands to FUs with higher worst-case delay. Wide operands are only executed on the FUs with the best critical path delay. Note that compared to [4] which always attempts to schedule operations on the fastest FU, *VAIL* utilizes the slower FUs by scheduling them for NW operations.

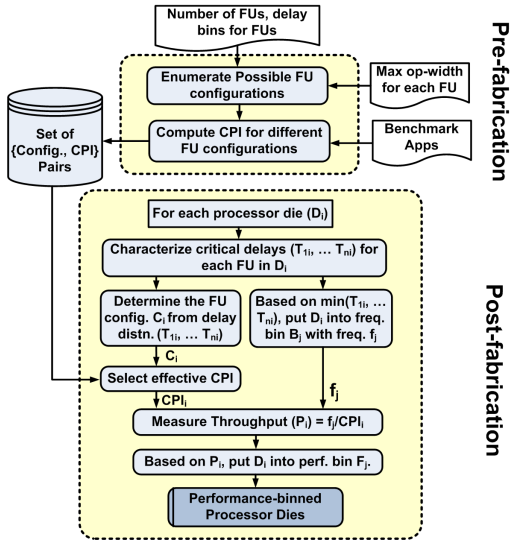

Figure 5: Flowchart showing the methodology for performance binning.

4.2 Hardware Modifications for Variation-Aware Issue Logic

Fig. 4(a) shows the integration of the proposed FU selection logic in a reservation station based superscalar pipeline model [9]. In such a model, operands issued to the FUs are read from the Reorder Buffer (ROB). Logic for selecting ready instructions from issue window is implemented as a tree of priority encoders (Fig. 4(b)). Fig. 4(a) shows that for realizing the width-dependent instruction issue, each ROB entry needs to be augmented with 2 extra bits (W_1W_0) for storing the max operand width corresponding to the two source operands Src_1 and Src_2. In our encoding scheme, $W_1W_0 = 00$, $W_1W_0 = 01$ and $W_1W_0 = 10$ denote maximum operand sizes upto 8-b, upto 16-b and more than 16-b respectively. We note that the latency for max operand width calculation can be masked by the delay components of the wakeup logic stage. This is illustrated in Fig. 4(c). Considering 2-input OR-gates, the W_1W_0 calculation for 64-b operands will require $\sim$ 6 logic stages, the delay for which can be easily masked by the wakeup delay for 128-entry, 8-wide issue ROB which is $\sim 350ps$ at 45nm technology node [9]. Note that read delay for W_1W_0 values is also masked by generation of the *Req* signal. In the proposed scheme, the request selected by the root of the priority encoder tree (Fig. 4(b)), is sent only to those FUs which are fast enough to compute operands with $OpWidth_{MAX} = W_1W_0$. This is achieved by encoding the $OpWidth_{MAX}$ for each FU with 2 bits denoted as D_1D_0 and ensuring that $D_1D_0 >= W_1W_0$. Shift in the delay for a FU due to WID variation, and hence the value of D_1D_0 can be accurately estimated by combining on-chip Delay Measurement Hardware (DMH) and Built-In-Self-Test (BIST) techniques [4]. After manufacturing test, D_1D_0 values for all the FUs are loaded into a small non-volatile memory (32-bit wide for 16 FUs). From our simulations, we note that introduction of the comparison logic introduces a mere $17ps$ additional delay in the critical path. Area for the extra comparison logic is $1283\mu m^2$.

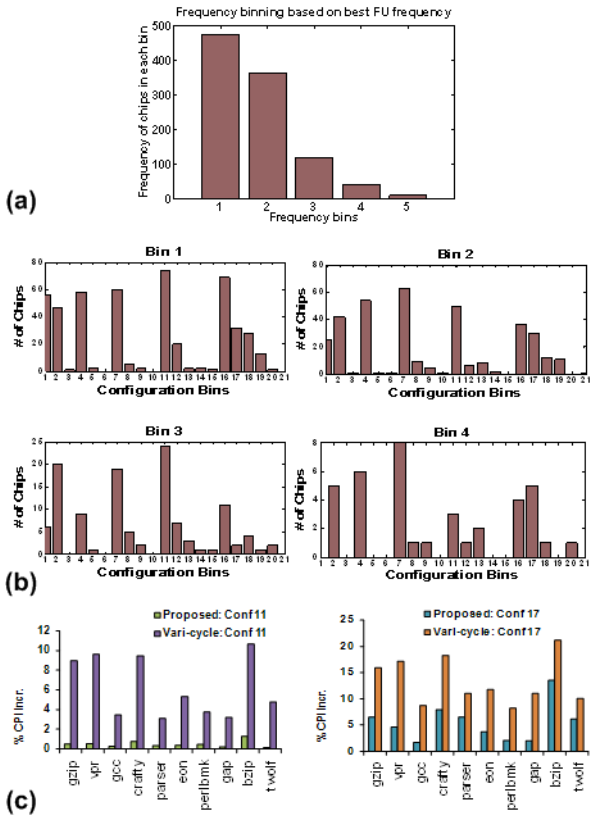

Figure 6: a) Frequency binning of dies based on the best FU frequency; b) FU configurations depending on the degree of WID variation in each bin; c) CPI overhead for *VAIL* and the vari-cycle approach [4].

5. "PERFORMANCE BINNING" PROCEDURE

Superscalar processors employing techniques to tolerate WID variation cannot be simply binned on the basis of operating frequency alone. The reason is for a given variation tolerance technique, processors in a particular frequency bin can have worse throughput when compared to another processor in a lower frequency bin. We therefore introduce the concept of "*performance binning*" where performance is measured in terms of the total execution time given as $T_{exec} = (\#ofInstr) \times (CPI)/(Op.freq)$.

The overall methodology for performance binning (Fig. 5) is divided into pre-fabrication and post-fabrication stages. Before fabrication, considering the scenarios where a variable number of FUs may move to a slower frequency bin after fabrication, a number of FU configurations are derived. By *FU configuration* we mean the ratio of fast to slow FUs in a processor. The effective CPI for each of these configurations is then determined by simulating the processor model for a set of benchmark applications. Post-fabrication, for each die D_i, critical delay for each FU on the die is determined. Let the delays be denoted as $T_{1i}...T_{ni}$. Now based on the minimum FU delay, the processor is placed into frequency bin B_j with operating frequency f_j. Next, based on the number of fast and slow FUs, each die D_i is then classified to have a particular FU configuration C_i. For this configuration C_i,

the corresponding CPI_i is then looked-up from the simulation results obtained prior to fabrication. With CPI_i and the operating frequency f_j, the effective throughput (P_i) for the die D_i is then found out as $P_i = \frac{f_j}{CPI_i}$. Based on P_i, D_i is finally placed into a particular performance bin F_j.

Fig. 6(a) shows the frequency binning for the 1000 dies based on the best FU frequency in each die. However, due to WID variation, dies should now be classified depending on the number of FUs which can support 8-b, 16-b or more than 16-b operations. For such classification, we assume that the FUs which have critical delays 17% and 49% more than the nominal delay will be able to support upto 16-b and 8-b of operation, respectively. Conf. 1 corresponds to the case when all FUs in a given die can support 64-b operation. Conf. 2 corresponds to the case when all but one FU supports wide operands, the remaining supporting upto 16-b operation. Conf. 3 corresponds to the case when single FU supports upto 8-b operation while others support 64-b operation. Since we have considered 6 integer adders, we have a total of 21 configurations where Conf. 21 corresponds to the case when all but one can only support upto 8-b operation. Note that due to frequency binning based on the best FU frequency there will always be atleast one FU which can support 64-b operation. Fig. 6(c) shows the CPI degradation for the proposed policy corresponding to two chosen FU configurations.

6. RESULTS AND DISCUSSION

Performance and yield improvement for *VAIL* was compared against the vari-cycle approach [4] as well as with worst-case frequency assignment for SPEC2000 applications. Throughput for multiple FU configurations for two schemes were determined by fastforwarding 500M instructions and running 1B instructions using Simplescalar Toolset 3.0 [10].

6.1 Performance Improvement over Vari-cycle approach

6.1.1 Integer Benchmarks

Fig. 6(c) compares the CPI degradation for two representative FU configurations. Compared to the no-variation scenario, average CPI increase for Conf. 11, 16, 17 and 20 was observed to be *0.48%*, *5.45%*, *5.47%* and *6.06%*, respectively. Conf. 11 and 17 in the vari-cycle scheme (as shown in Fig. 6(c)) corresponds to the case when the number of single cycle FUs is 2 and 1, respectively. For these two configurations, the average CPI improvement with *VAIL* is *5.74%* and *7.86%* respectively. For *VAIL*, average CPI increase in the worst-case (Conf. 21) is *6.11%*. This is a considerable improvement over the worst-case CPI degradation (*13.33%*) in the vari-cycle approach. Throughput degradation was also with integer adders at Conf. 11 and one of the multiplier can only support $OpWidth_{MAX} = 8$. However, due to the rareness of two or more integer multiplications occurring in the same issue window, the CPI increase compared to adder-only Conf. 11 is only *0.003%*.

6.1.2 FP Benchmarks

Since there are 4 FP-adders, the number of possible configurations based on the number of slow FP-adders is 10. Since the configuration space considering the variation in both the integer and the FP adder is considerably large

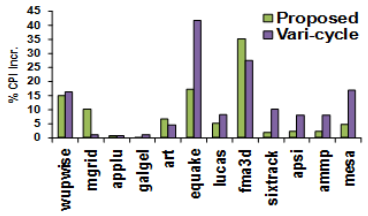

Figure 7: Comparison of CPI increase between the proposed and vari-cycle approaches when both integer and FP units suffer high WID variation.

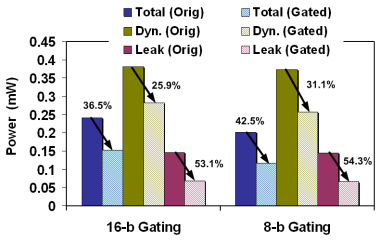

Figure 8: Percentage savings in total, dynamic and static power when a 64-b adder is opportunistically gated based on NW operands.

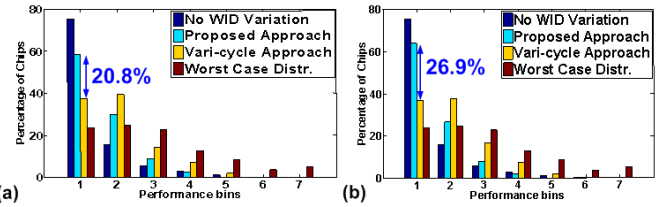

Figure 9: Variation in performance yield for frequency binning performed at intervals of a) 7% and b) 3.5% of the best die frequency.

$(21 \times 10 = 210)$, we have estimated the CPI degradation for the FP benchmarks for the following 3 scenarios: i) *Int:Conf 17, FP:Conf 1*; ii) *Int:Conf 1, FP:Conf 7* and iii) *Int:Conf 17, FP:Conf 7*. For *VAIL*, the average CPI increase for these three scenarios are *7.12%*, *1.71%* and *8.39%* respectively. On the other hand, the vari-cycle approach incurs an average CPI increase of *12.09%* for scenario 3 (Fig. 7).

6.2 Overhead Estimation

Considering the ROB occupies almost 15% of the processor die area [11] and the extra ROB entries increase the ROB area by *1.4%* (due to addition of 2 extra bits in 142 bit ROB entry), a pessimistic estimate of the increase in the total processor area due to the proposed scheme is *0.21%*. Similarly, considering that the ROB consumes almost 27% of the processor's total dynamic power [11], the extra W_1W_0 read increases the total dynamic power by approximately *0.38%*. For a base processor frequency of 1.6GHz, the extra comparison logic for preferential instruction issue introduces a delay overhead of *2.7%*.

6.3 Improvement in Energy Consumption

The proposed approach provides opportunity to save power

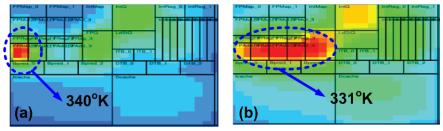

Figure 10: Thermal profile for a die with vari-cycle and *VAIL* approaches running (a,b) "gap" application.

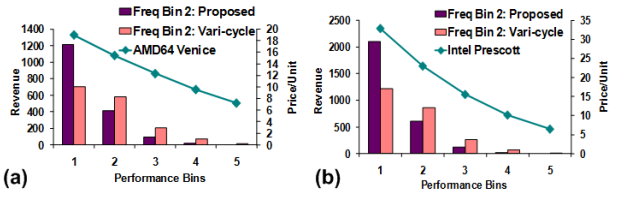

Figure 11: Improvement in revenue compared to vari-cycle approach considering both AMD and Intel Price/Unit profile.

in the slow units where larger operand width (e.g. >= 32) are not assigned. Such units are built to be 64-bit wide and hence one can turn-off the logic responsible for generating the most significant bits of the output. Such gating not only prevents dynamic power consumption due to spurious transitions in the unutilized logic, but also minimizes leakage consumption at high temperature. Fig. 8 shows that opportunistic supply gating based on operand width can achieve *36.5%* and *42.5%* saving in the total power consumption when the FU is operated in a 16-b or 8-b mode, respectively.

6.4 Improvement in FU thermal profile

Under severe WID variations, the vari-cycle approach issues most of the instructions to the faster execution units. This leads to larger power being dissipated in these units which eventually leads to localized hotspots. We simulated the steady state temperature profile for both vari-cycle and *VAIL* using Hotspot 5.0 [13]. As we may note from Fig. 10, compared to the vari-cycle approach (Fig. 10(a) and (c)), the proposed scheme (Fig. 10(b) and (d)) improves the worst-case FU temperature. We observe that in case of "gap" application, temperature improves by $9°C$.

6.5 Improvement in Yield

Fig. 9 shows the performance binning for *VAIL* and the vari-cycle approach for 2 different frequency binning schemes performed at intervals of 7% and 3.5% of the best die frequency. The number of performance bins was kept constant at 7, with each bin having 10% more throughput degradation compared to the base throughput of 1. As we note from Fig. 9, the yield improvement with the *VAIL* approach increases as the frequency bins are spaced more closely. For the second frequency binning scenario, the yield improvement in the highest performance bin is $\sim$ 27%. Compared to the worst-case frequency assignment approach, the high performance yield improves by *146%* and *170%*, respectively.

6.6 Improvement in Revenue

Increasing the number of dies in the highest frequency bin increases the revenue for the manufacturer. We have evaluated the impact of revenue considering two separate price/unit profiles as reported in [12] . As may be noted from Fig. 11, with the price/unit profiles for AMD64 Venice and Intel Prescott processors, improvements over the vari-cycle approach are *9.7%* and *15.1%* respectively. Compared to the worst-case frequency assignment, the revenue improvements are *30.3%* and *49%*, respectively.

7. CONCLUSION

In this paper we have presented a novel variation-aware instruction scheduling approach in modern superscalar processors that leverages on the abundance of narrow-width operands in general-purpose applications. The proposed approach preferentially assigns operands of smaller sizes to slower FUs, thus avoiding worst-case frequency assignment. We have also presented the concept of *performance binning*, where the dies are binned with respect to their throughput instead of the conventional approach of frequency binning. The requirement for such binning comes from the fact that many chips with the same frequency can have varying performances with variation-aware scheduling due to different WID variation scenarios. Compared to both the conventional worst-case frequency assignment and a vari-cycle approach, *VAIL* achieves considerably better yield and profit. Besides, *VAIL* achieves significantly better thermal profile than vari-cycle approach as well as opportunistic power saving in the slow units by employing power gating.

8. REFERENCES

[1] J. Tschanz et al., "Variation-Tolerant Circuits: Circuit Solutions and Techniques", *DAC*, 2005.

[2] S. Borkar et al., "Parameter Variation and Impact on Circuits and Microarchitecture", *DAC*, 2003.

[3] L. Benini et al., "Telescopic Units: A New Paradigm for Performance Optimization of VLSI Designs", *IEEE TCAD*, Vol. 17, No. 3, 1998.

[4] P. Ndai et al., "Within-Die Variation-Aware Scheduling in Superscalar Processors for Improved Throughput", *IEEE TComp*, Vol. 57, No. 7, 2008.

[5] D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance", *HPCA*, 1999.

[6] Predictive Technology Models (PTM). [Online] `http://www.eas.asu.edu/~ptm/modelcard/45nm_MGK.pm`

[7] D. Ernst et al, "Razor: A Low-power Pipeline Based on Circuit-Level Timing Speculation", *IEEE Micro*, 2003.

[8] H. Li et al. "DCG: Deterministic Clock-Gating for Low-Power Microprocessor Design", *IEEE TVLSI*, Vol. 12, No. 3, 2004.

[9] S. Palacharla et al., "Complexity-Effective Superscalar Processors", *ISCA*, 1997.

[10] Simplescalar Toolset V3.0: [Online] `http://www.simplescalar.com`

[11] G. Kucuk et al., "Complexity-effective reorder buffer designs for superscalar processors", *IEEE TComp*, Vol. 53, No. 6, 2004.

[12] A. Datta et al., "Profit Aware Circuit Design Under Process Variations Considering Speed Binning", *IEEE TVLSI*, Vol.16, No.7, 2008.

[13] Hotspot V5.0: [Online] `http://lava.cs.virginia.edu/HotSpot`