An Updated Evaluation of ReCycle

Abhishek Tiwari and Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu
{atiwari,torrellas}@cs.uiuc.edu

Abstract

Process variation reduces a pipeline's maximum attainable frequency by creating unbalance in the stage delays. As a result, the pipeline ends up cycling with a period close to that of the slowest pipeline stage. ReCycle was proposed in ISCA 2007 as a framework for comprehensively applying *cycle time stealing* to balance the stage delays under process variation, thereby allowing the pipeline to cycle with a period close to the average latency of the stages.

This paper duplicates the evaluation of ReCycle with more realistic pipeline and critical path models than in the original paper. Most notably, we do not assign one cycle to the feedback path of each pipeline loop. As a result, our pipeline contains five single-cycle loops. This is significant because these loops are not amenable to cycle time stealing.

In this updated environment, ReCycle is able to regain on average only 40% of the performance lost to process variation. In contrast, in the original paper, ReCycle regained 64%. Moreover, we find that further adding Donor stages does not significantly increase performance. Consequently, we propose to extend the Donor algorithm by applying Forward Body Biasing (FBB) to single-cycle loops when they become critical. With ReCycle, Donor stages, and FBB, we regain on average 90% of the performance lost to variation — still short of the roughly 110% regained by ReCycle and Donor stages alone in the original ReCycle paper.

1. Introduction

Process variation has been identified as a key problem facing microprocessor design in the sub-45nm regime. Process variation refers to the fluctuation in transistor properties such as switching speed, across different transistors on a chip or across chips. Due to process variation, some pipeline stages of a processor end up having longer path delays. In this case, the entire pipeline ends up being clocked at a lower frequency than nominal.

ReCycle was proposed in ISCA 2007 [8] as a technique to mitigate this performance degradation caused by process variation. It relies on the observation that while the processor frequency is determined by the slowest pipeline stage, other stages in the pipeline are faster and end up wasting a significant portion of their cycle time. Consequently, ReCycle comprehensively uses *cycle time stealing* [1] to transfer this timing slack from the fast stages to the slower ones, thereby allowing the slow stages to take more than one clock period to evaluate their results. As a result, the clock period of the processor is no longer equal to the maximum stage delay; it ends up becoming close to the average stage delay in the slowest pipeline loop. ReCycle provided a framework to comprehensively apply cycle time stealing within pipeline loops.

ReCycle further improves the pipeline frequency by inserting empty Donor stages in the slowest (or *critical*) pipeline loop — the one that, by having the longest average stage delay, determines the processor cycle time. Donor stages "donate" timing slack to other stages in the loop and reduce the average stage delay within the loop.

The performance improvements obtained by ReCycle and Donor stages depend on the structure of the pipeline loops. In the original ReCycle paper, the feedback path of each pipeline loop was assumed to take one cycle. This is not very realistic. Moreover, it especially distorts tight, single-cycle pipeline loops, which it transforms into two-cycle loops. Such loops now appear to be less critical than they realistically are, because ReCycle can move time between the pipeline stage cycle and the feedback path cycle. In reality, single-cycle loops are not amenable to cycle time stealing.

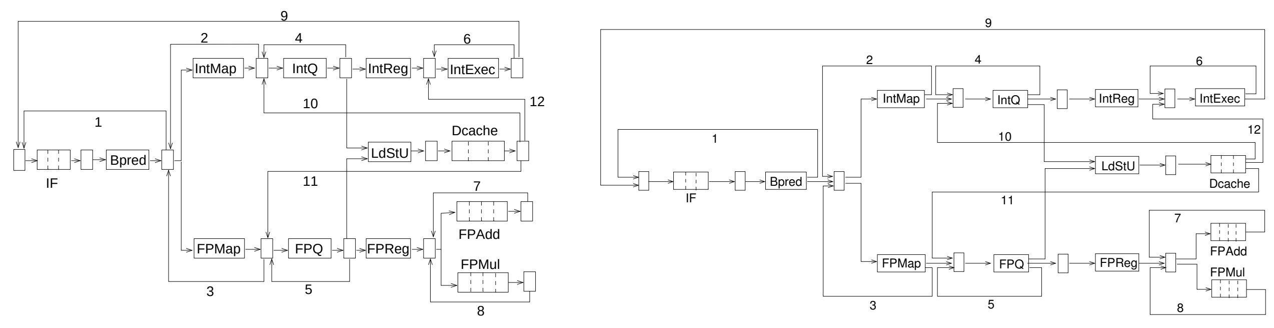
In this paper, we replicate the evaluation of the ReCycle paper from ISCA 2007 with more realistic pipeline and critical path models than in the original paper. Most notably, we do not assign one cycle to the feedback path of each pipeline loop. Instead, the feedback path consumes a fraction of the cycle time assigned to the last stage of the loop. As a result, our pipeline retains several single-cycle loops.

In this new environment, ReCycle recovers on average only 40% of the performance lost to process variation. In contrast, in the original paper, ReCycle regained 64%. Moreover, we find that further adding Donor stages does not significantly increase performance. Consequently, we propose to extend the Donor algorithm by applying Forward Body Biasing (FBB) to single-cycle loops when they become critical. With ReCycle, Donor stages, and FBB, we regain on average 90% of the performance lost to variation — still short of the roughly 110% regained by ReCycle and Donor stages alone in the original ReCycle paper.

2. Background

2.1. Pipeline Evaluated

In this paper, we evaluate ReCycle with a more realistic model of the Alpha 21264 pipeline [3] than in the original ReCycle pa-



(a) Pipeline structure used in ReCycle.

(b) Pipeline structure used in this paper.

Figure 1: Simplified version of the Alpha 21264 pipeline used in the original ReCycle paper (a) and in this paper (b).

per [8]. Specifically, in the original paper, the feedback paths in all of the pipeline loops were modeled to take one pipeline cycle. This is not very realistic and especially distorts tight pipeline loops: it transforms loops composed of a single pipeline stage into two-cycle loops. Such loops now appear to be less critical than they are, because ReCycle can move time between the pipeline stage cycle and the feedback path cycle.

In contrast, in this paper, we model the feedback path to take part of the time assigned to the last pipeline stage of the corresponding loop. Consequently, a feedback path does not add any additional cycle to its loop. In particular, a loop composed of a single pipeline stage truly takes one single cycle. Such a loop now retains its critical effect: it is not possible to transfer time between the logic part of the stage and the feedback path.

Figures 1(a) and (b) show the simplified Alpha 21264 pipeline used in the original ReCycle paper and the one used in this paper, respectively. The long boxes represent pipeline stages, while the short boxes are the pipeline registers between them. Some of the long boxes are in fact multiple pipeline stages separated by pipeline registers, as shown with the dashed lines. For example, the IF has three pipeline stages separated by pipeline registers. We put multiple stages under the same box like in IF to indicate that the critical paths in the stages are so spatially intertwined, that the values of the systematic component of process variation parameters are assumed to be the same for the stages. Lines between logical stages represent communication links.

The figures show the front end of the pipeline followed by the stages in the integer datapath along the top, the stages in the floating point data path along the bottom, and the load-store unit and cache in the middle. While a real processor has more communication links, these figures show only those that were considered most important or most time-critical. For example, the write back links are not shown, since write back is less time-critical. A total of 12 feedback paths (and therefore loops) are shown and labeled. Table 1 describes the loops. In increasing numerical order, these loops will be referred to like in the original ReCycle paper as: *fetch*, *iren*, *fpren*, *iissue*, *fpissue*, *ialu*, *fpadd*, *fpmul*, *bmiss*, *ildsp*, *fpldsp*, and *ldfwd*.

In Figure 1(a), each feedback path starts and ends in a pipeline register and, therefore, is assigned one cycle. In Figure 1(b), each feedback path starts inside a pipeline stage and ends in a pipeline register. It is assigned part of the stage's delay.

In Figure 1(b), there are five single-cycle loops, namely the integer and floating-point rename loops, the integer and floating-point issue loops, and the integer execute loop. In Figure 1(a), these loops took two cycles instead of one because of the cycle-long feedback paths.

Name	Description	Fdbk Path	Components
Fetch	Dependence between PC and Next PC	1	IF, Bpred, 1
Int rename	Dependence between inst. assigning a rename tag and a later one reading the tag	2	IntMap, 2
FP rename		3	FPMap, 3
Int issue	Dependence between the select of a producer inst. and the wakeup of a consumer	4	IntQ, 4
FP issue		5	FPQ, 5
Int ALU	Forwarding from execute to execute	6	IntExec, 6
FPAdd		7	FPAdd, 7
FPMul		8	FPMul, 8
Branch mispred.	Mispredicted branch	9	IF, Bpred, IntMap IntQ, IntReg, IntExec, 9
Int load misspecul	Load miss replay	10	IntQ, LdStU, Dcache, 10
FP load misspecul		11	FPQ, LdStU, Dcache, 11
Load forward	Forwarding from load to integer execute	12	IntExec, 9, IF, Bpred, IntMap, IntQ, LdStU, Dcache, 12

Table 1: Loops in the pipeline considered.

As a result of the changes to the feedback path latency mentioned above, some constraints associated with the pipeline stages also change. Specifically, in the pipeline of Figure 1(a), the setup and hold constraints for a feedback path originating in register i and ending in register j were:

$$\delta_i + T_{feedback\_delay} + T_{setup} \leq T_{CP} + \delta_f$$
$$\delta_i + T_{feedback\_delay} \geq \delta_f + T_{hold}$$

where $T_{feedback\_delay}$ is the delay of the feedback path, T_{CP} is the pipeline's clock period, T_{setup} and T_{hold} are the setup and hold times, respectively, and δ is the clock skew at the corresponding register [8]. In the revised pipeline of Figure 1(b), these constraints are replaced by:

$$\delta_i + T_{stage\_delay} + T_{feedback\_delay} + T_{setup} \leq T_{CP} + \delta_f$$
$$\delta_i + T_{stage\_delay} + T_{feedback\_delay} \geq \delta_f + T_{hold}$$

where T_{stage_delay} is the delay of the logic in the last pipeline stage of the loop. The total number of constraints for ReCycle remains the same.

2.2. Variation Model

To model process variation, we use the same model as in the original ReCycle paper, namely, the model in [6]. We model the within die (WID) variation in transistor threshold voltage (V_{th}) and effective channel length (L_{eff}). The WID variation is subdivided into *random* and *systematic* components. Each of these components is assumed to be normally distributed. The systematic variation is modeled as a multivariate normal distribution with a Spherical spatial correlation; the random variation is modeled as an uncorrelated normal distribution at the transistor level. All assumptions on V_{th} and L_{eff} variation are the same as in the original ReCycle paper [8]. In particular, V_{th}'s variation has an overall σ/μ of 0.09 and a correlation range ϕ for systematic variation of 0.5. We model a CMP with four cores.

The transistor delays are computed from V_{th} and L_{eff} using the alpha power law [5]. To increase accuracy, the distribution and timing of the critical paths in a pipeline stage used in this paper differs slightly from the model used in the original ReCycle paper. Specifically, the latter assumed that the number of critical paths in a pipeline stage is proportional to the area of the stage. In this paper, we use the more realistic critical path distribution and timing described in [6]. That model uses, for logic stages, experimental data from Ernst *et al.* [2] and, for memory stages, extensions to the model of Mukhopadhyay *et al.* [4]. In this model, we set the σ_{extra}/μ of D_{extra} to 0.028.

Table 2 classifies the pipeline stages based on whether they are modeled as containing mostly logic, a small SRAM structure, or a large SRAM structure. SRAM structures are sized using CACTI [7]. Their critical path is composed of decode, wordline, bitline, and sense amplifier. The combination of wordline driver, wordline, pass transistor, bitline, and sense amplifier is modeled as three transistor delays plus wire delay. The access time of large SRAM structures is set to three cycles, of which one is taken by the decoder. The access time of small SRAM structures is set to one cycle, equally divided into decoder delay, wordline delay, and the rest. As in the original ReCycle paper, we assume that wire delay is unaffected by variation. For this reason, considering stages like *Bpred*, *IntMap*, and *FPMap* as logic stages (even though they also contain small tables) is a conservative assumption.

Mostly logic	Bpred, IntMap, FPMap, IntExec, FPAdd, FPMul
Small SRAM	IntQ, LdStU, FPQ, FPReg, IntReg
Large SRAM	Dcache, IF

Table 2: Classification of pipeline stages.

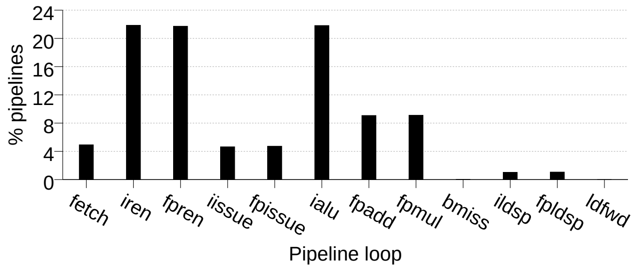
We repeat every experiment 10,000 times. We use a statistics package to generate specific instantiations of the variation maps. Like in the original ReCycle paper, the default pipeline has a per-stage useful logic depth equal to 17FO4.

3. Evaluation

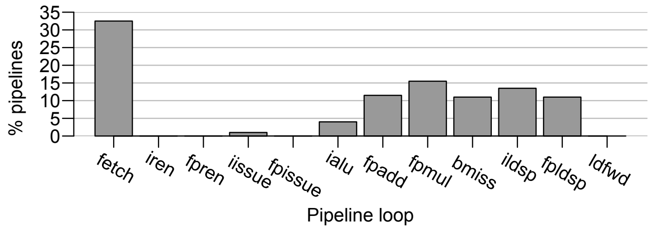
3.1. Timing Issues

In any given pipeline, the loop with the longest average stage delay limits the ability of ReCycle to further reduce the pipeline clock period. This loop is called the *critical* one. Figure 2(a) shows the number of times that each pipeline loop is critical for the batch of 10,000 pipelines considered in this paper.

The probability of criticality of a pipeline loop depends on three factors: (1) the number of pipeline stages in the loop, (2) whether some of these stages share the same value for the systematic component of process variation parameters, and (3) the fraction of wire delay in the loop. Specifically, loops with a large number of stages will be able to average out inter-stage delay variation better. Therefore, they are less likely to be critical. From the figure, we see that the branch misprediction loop (*bmiss*) and the load forwarding loop (*ldfwd*) are almost never critical because of the large number of stages in these loops. On the other hand, single-cycle loops like integer and floating-point rename (*iren* and *fpren*) and integer execute (*ialu*) are frequently critical. The frequent criticality of these loops is clearer in this paper than in Figure 7 of the original Recycle paper [8], which we repeat in Figure 2(b).



(a)



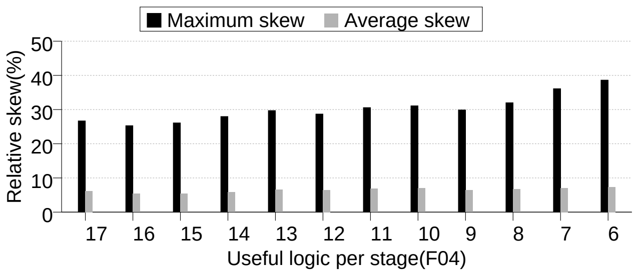
(b)

Figure 2: Histogram of critical pipeline loops in the updated evaluation (a) and the original evaluation (b).

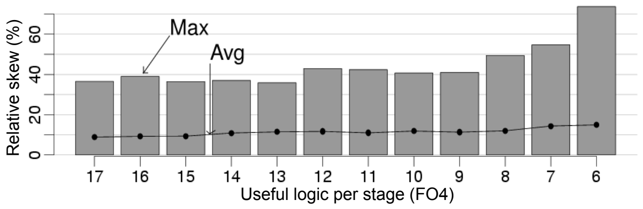
As indicated in Section 2.1, several pipeline stages are modeled to have the same systematic component of variation parameters. This includes, for example, the three stages in IF. These stages are only able to average out their *random* component of variation. Therefore, loops that contain this type of stages are more likely to be critical than other loops with the same total number of stages. We can see this from the *fpadd* and *fpmul* loops. Each of the FPAdd and FPMul functional units has 4 stages, which have the same systematic component of variation. Therefore, these loops are critical more often than the *fetch* loop.

Finally, since wires are not subject to variation in the model, loops with a higher fraction of wire delay will be less affected by variation, and hence less likely to be critical. This explains why the single-cycle integer and floating-point issue loops (*iissue* and *fpissue*) are less frequently critical than the other single-cycle loops: *iissue* and *fpissue* consist of SRAM structures dominated by wire delay, while *iren*, *fpren*, and *ialu* are modeled as logic stages.

Overall, given that single-cycle loops are often critical, ReCycle will necessarily be less effective than in the pipeline model used in the original ReCycle paper. This can be seen from Figure 3(a), which shows the average and maximum time skew that ReCycle inserts per pipeline register, as we reduce the useful logic depth of the pipeline stages from the default 17FO4 to 6FO4. The skews are shown relative to the stage delay of a no-variation pipeline of the same logic depth of the stages. For clarity, Figure 3(b) shows the corresponding figure from the original ReCycle paper, which was Figure 9(b).



(a)



(b)

Figure 3: Skew versus logic depth inserted by ReCycle in the updated evaluation (a) and the original evaluation (b).

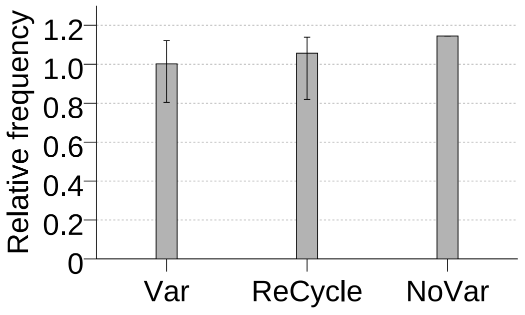
The skew is a measure of the stage unbalance in the pipeline loops. In both figures, both the average and maximum skews tend to increase as we decrease the logic depth. The reason is that, for shorter stages, the random component of the variation is more prominent, increasing the unbalance. However, while the average and maximum skews were 10–15% and around 40%, respectively, in the original paper, they are only 5–7% and around 30%, respectively, in Figure 3(a). Consequently, ReCycle is less effective in this paper.

3.2. Frequency After Applying ReCycle

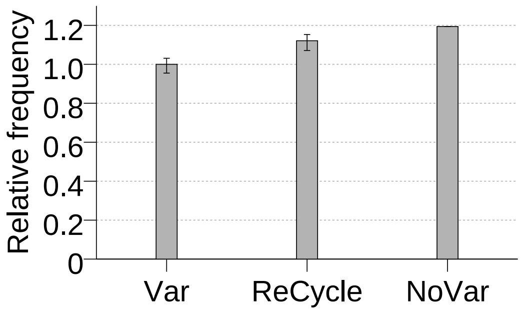
We now evaluate the frequency increases delivered by ReCycle. Figure 4(a) shows the frequency of three different environments: pipeline with process variation and no ReCycle (*Var*), pipeline with variation and ReCycle (*ReCycle*), and pipeline with no variation (*NoVar*). The bars are normalized to the average frequency of *Var*. The *Var* and *ReCycle* bars show the range of frequencies for the different experiments. The figure corresponds to Figure 10 of the original ReCycle paper for ϕ=0.5, which we show as Figure 4(b).

The frequency improvement due to ReCycle depends on which loop contains the slowest stage after variation. If such a loop has many stages and significant inter-stage delay variation, ReCycle is likely to improve the pipeline frequency significantly. On the other hand, if such a stage happens to be in a single-cycle loop, ReCycle does not improve the frequency at all.

From the figure, we see that, on average, ReCycle improves the frequency of the pipeline by 6% over *Var*. Given that *NoVar*'s frequency is 15% higher than *Var*'s, ReCycle recovers 40% of the frequency lost to variation. These figures are smaller than the ones presented in the original ReCycle paper. Indeed, according to Figure 4(b), the average frequency gains delivered by ReCycle over *Var* are 12%, which correspond to 63% of the frequency lost to variation. Overall, the presence of single-cycle loops results in relatively smaller gains of ReCycle over *Var*. Moreover, the corrections made to the critical path models result in slightly smaller differences between *NoVar* and *Var*.



(a)



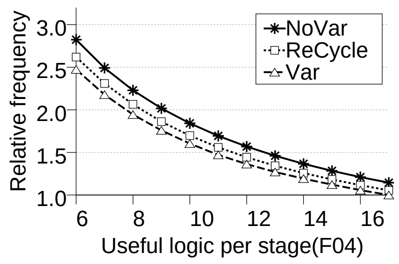
(b)

Figure 4: Pipeline frequency of the environments considered, for the updated evaluation (a) and the original evaluation (b).

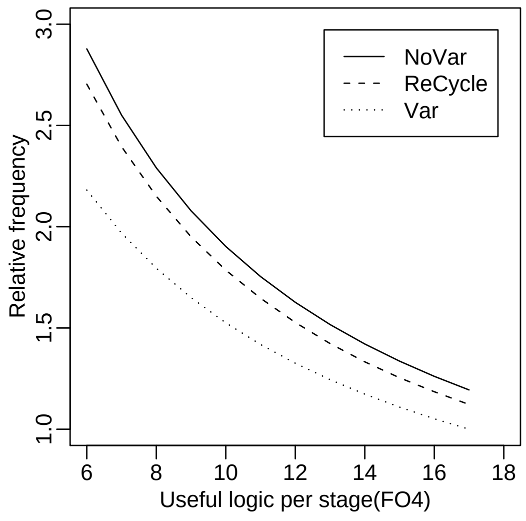
Next, we compare the three environments as we vary the useful logic depth per pipeline stage from 17FO4 to 6FO4. This is shown in Figure 5(a), where the curves are normalized to the *Var* frequency with 17FO4. This figure corresponds to Figure 11 in the original ReCycle paper, which we show as Figure 5(b).

The two figures show the same trends. As we decrease the logic depth per stage, the frequency increases for all three environments. Moreover, the separation between the *Var* and the other curves increases, which means that process variation hurts the frequency of a pipeline more.

The main difference between the two figures is that, in this paper, the *ReCycle* curve is relatively closer to the *Var* curve than in the original ReCycle paper. The reason is that, due to the presence of the single-cycle loops in the pipeline, ReCycle is relatively less effective than before. Overall, *ReCycle* increases the average frequency over *Var* by about 6% across all the design points. Note that the *Var* curve reaches different values in this paper and in the original ReCycle paper. This is due to the corrections in the critical path



(a)



(b)

Figure 5: Pipeline frequency for different useful logic depths per pipeline stage in the updated evaluation (a) and the original evaluation (b).

models.

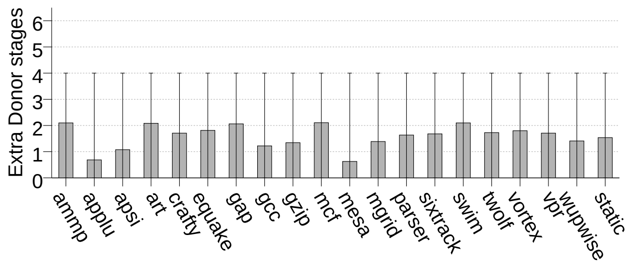
3.3. Adding Donor Stages and Forward Body Bias

The frequency improvement due to ReCycle is constrained by the critical loop in the pipeline. To further improve the pipeline frequency, we must find a way to decrease the average stage delay in the critical loop. In the original ReCycle paper, this was accomplished by inserting a *Donor* stage in the loop. This operation reduces the average stage delay in this loop, making another loop the critical one. Adding a Donor stage increases the pipeline frequency, but makes the pipeline loop longer, thus decreasing the IPC when the loop latencies are exposed. The original paper showed that, with a certain number of loops enhanced with a Donor stage, an overall higher performance can be obtained.

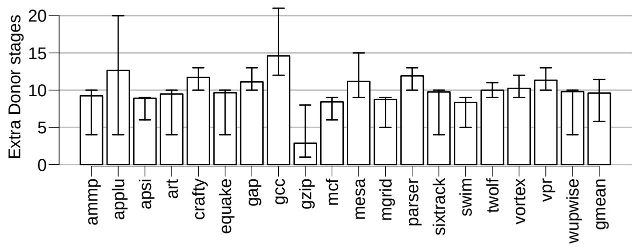
In this paper, we repeat the experiments. However, we find that adding a Donor stage to a single-cycle loop when it is critical results in an unsurmountable IPC penalty. Consequently, we evaluate two approaches. One is to add Donor stages only to critical loops that are longer than one cycle. The second approach is to additionally apply Forward Body Biasing (FBB) to single-cycle loops when they are critical. Applying FBB improves the speed of the stage at the cost of an increase in leakage power [9].

We extend the Donor algorithm so that we add Donor stages to loops as they become critical and, in the second approach, we additionally apply FBB to single-cycle loops as they become critical. We stop when the performance finishes increasing any further, or we reach the power limit of 30W per processor. Note that the optimal configuration of Donor stages and FBB depends on the workload. Such configuration could either be set once after manufacture, based on the assumption of a representative workload mix, or it could be adjusted dynamically based on the currently-running workload. We call such configurations *StDonor+FBB* and *DynDonor+FBB*, respectively.

In our experiments, we find that, if we just apply ReCycle plus Donor stages, we obtain negligible performance gains over ReCycle. This is because single-cycle loops quickly become critical. Consequently, we focus on ReCycle plus Donor stages plus FBB for critical single-cycle loops. In this environment, Figure 6(a) shows the average number of Donor stages added to a pipeline, and the range for the different pipelines. There is a bar for each SPEC application, and a bar labeled *static* for the mean across applications. The latter corresponds to the *StDonor+FBB* environment, while the other bars correspond to the *DynDonor+FBB* environment. The figure also shows the range of values across the experiments. We can see that the number of Donor stages is typically between 1 and 2. This is a smaller number than in the original ReCycle paper, which was around 10 — as shown in Figure 6(b). This is largely because we do not add Donor stages to single-cycle loops. As a result, Donor stages are likely to be less effective in our pipeline.



(a)



(b)

Figure 6: Number of Donor stages used in the updated evaluation (a) and the original evaluation (b).

Figure 7 plots the average number of stages that receive FBB and the range for different pipelines. These stages all belong to single-cycle loops. We can see that we typically apply FBB to about 3 stages. We estimate that, in the worst case, when all 5 stages receive FBB for a maximum ΔV_{th} of -75mV, we add about 7% to the total power consumed by each processor (and private caches).

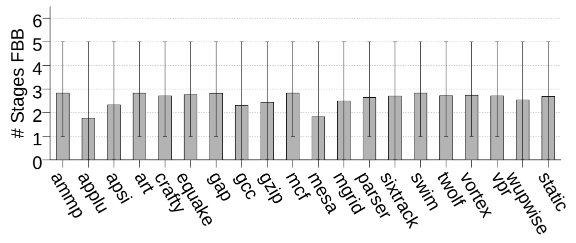
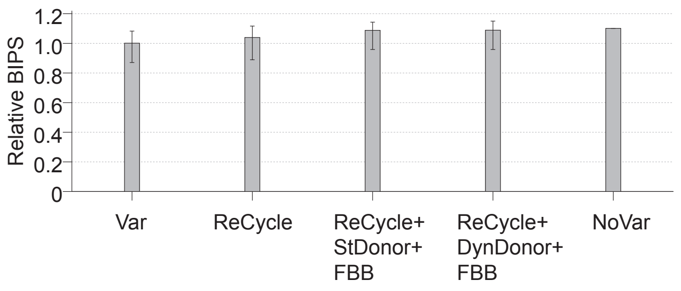


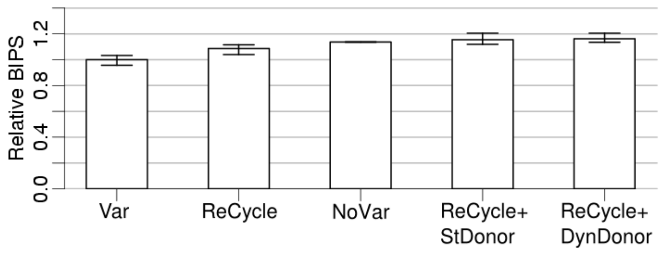
Figure 7: Number of stages with forward body bias.

3.4. Overall Performance Evaluation

Finally, Figure 8(a) compares the performance of *Var*, *NoVar*, *ReCycle*, ReCycle plus the static application of the Donor algorithm plus FBB (*ReCycle + StDonor + FBB*), and ReCycle plus the dynamic application of the Donor algorithm plus FBB (*ReCycle + DynDonor + FBB*). This figure corresponds to Figure 14 of the original ReCycle paper, which is shown as Figure 8(b). The static and dynamic Donor environments are generated as in the original ReCycle paper. Specifically, in *DynDonor*, we rerun the Donor algorithm at the beginning of each application. Moreover, we assume that we know the average IPC impact of adding each Donor stage from a previous profiling run. In the figure, the bars show the average performance and, except for *NoVar*, include a segment with the range of values measured. All bars are normalized to the average *Var*.



(a)



(b)

Figure 8: Performance of different environments for the updated evaluation (a) and the original evaluation (b).

From Figure 8(a), we see that the performance of *ReCycle* and *NoVar* is 4% and 10% higher, respectively, than *Var*. This is in contrast to the original ReCycle paper, where *ReCycle* and *NoVar* were 9% and 14% higher, respectively, than *Var*. The corrected critical path model has decreased the difference between *Var* and *NoVar* slightly. More importantly, the combination of more realistic pipeline model and critical path model has decreased the relative effectiveness of ReCycle: it is now only able to recover 40% of the gap between *Var* and *NoVar* instead of 64% in the original paper.

The figure also shows the impact of applying ReCycle plus Donor stages plus FBB for single-cycle loops. We see that the static and dynamic application of the Donor algorithm deliver approximately the same performance. Such performance is roughly 9% higher than *Var*. Therefore, with this technique, we recover 90% of the performance lost to variation. This is in contrast to the performance recovered with ReCycle and Donor stages in the original paper: 107% and 114% of the performance lost to variation depending on whether we use *StDonor* or *DynDonor*, respectively.

Finally, we note that applying ReCycle plus Donor stages does not improve the performance over ReCycle alone much. We need an additional technique to speed-up the stage in critical single-cycle pipeline loops. In this paper, we used FBB. However, other related techniques such as Adaptive Supply Voltage (ASV) could be used.

4. Conclusions

In this paper, we replicated the evaluation of the ReCycle paper from ISCA 2007 with more realistic pipeline and critical path models. Most notably, the pipeline has five single-cycle loops, which is significant because single-cycle loops are not amenable to cycle time stealing.

In this more realistic environment, ReCycle only recovered on average 40% of the performance lost to process variation. In contrast, in the original paper paper, ReCycle regained 64%. Moreover, we found that further adding Donor stages did not significantly increase performance. Consequently, we proposed to extend the Donor algorithm by applying Forward Body Biasing (FBB) to single-cycle loops when they become critical. With ReCycle, Donor stages, and FBB, we regained on average 90% of the performance lost to variation — still short of the roughly 110% regained by ReCycle plus Donor stages alone in the original ReCycle paper.

References

[1] K. Bernstein, K. M. Carrig, C. M. Durham, P. R. Hansen, D. Hogenmiller, E. J. Nowak, and N. J. Rohrer. *High Speed CMOS Design Styles*. Kluwer Academic Publishers, 1999.

[2] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Zeisler, 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*, December 2003.

[3] R. E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[4] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *IEEE Transactions on Computer-Aided Design*, 24(12):1859–1880, December 2005.

[5] T. Sakurai and R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *Journal of Solid-State Circuits*, 25(2):584–594, 1990.

[6] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: A model of process variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, February 2008.

[7] D. Tarjan, S. Thoziyoor, and N. Jouppi. CACTI 4.0. Technical Report 2006/86, HP Laboratories, June 2006.

[8] A. Tiwari, S. R. Sarangi, and J. Torrellas. ReCycle: Pipeline adaptation to tolerate process variation. In *International Symposium on Computer Architecture*, June 2007.

[9] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *Journal of Solid-State Circuits*, 37(11):1396–1402, 2002.