Accelerating Approximations via Slack Recycling

Gokul Subramanian Ravi
gravi@wisc.edu

Mikko Lipasti
mikko@engr.wisc.edu

Abstract

Motivated by limitations in current approximate hardware, we propose to extend the notion of timing slack into approximate computation. We propose Axl: disciplined but increased aggressiveness in timing slack estimates at an operation granularity - accounting for slack from PVT, data, as well as approximation tolerance. This results in improved performance/efficiency within application-specific tolerable error rates.

1 Introduction

In order to operate reliably and produce expected outputs, modern processors set timing margins conservatively at design time to support extreme variations in workload (data) and environment (PVT) [4, 6]. In the common non-critical cases, this creates clock cycle *slack* - the fraction of the clock cycle performing no useful work. In prior works, we proposed LAGS and ReDSOC (both under submission; based on [9]) which use transparent latching techniques to effectively recycle multiple forms of slack and improve performance while maintaining required reliability constraints. These mechanisms focused on timing slack reduction for accurate computing - we believe there is significant potential to extend these proposals into the realm of approximate computing.

There have been multiple proposals in the software space to unearth opportunities to leverage approximate computing for speedup or efficiency improvements, but we believe that there is much scope for improvement in hardware design. Our proposal Axl, is motivated by the limitations in current approximate hardware, with specific focus on timing approximation, discussed in Sec.3.

Axl is effective in avoiding prior limitations, by piggybacking approximation on our prior work on slack recycling (Sec.2). Converting tolerable approximation into a slack component allows the use of LAGS/ReDSOC slack recycling techniques to accelerate sequences of computations.

2 Recycling Slack in Accurate Computing

In this section we summarize our prior proposals for recycling PVT Slack (from PVT variations), Data Slack (from operation type, and operand data widths/types) using a transparent latching mechanism, targeting accurate computing, as illustrated in Fig.1

① Simple asynchronous execution engines are integrated seamlessly into synchronous pipelines. ② Asynchronous engines implemented as transparent (latched) pipelines with synchronous control, resulting in low design complexity. ③ Multiple asynchronously executable operations, bounded by synchronous boundaries, grouped together into a transparent multi-cycle execution flow, allowing slack to accumulated along the sequences. ④ Transparent latches with slack-aware control are employed between execution units to recycle the slack from a producer operation; by starting the execution of dependent consumer operations at the exact instant of completion of the producer operation. ⑤ Speedup in executing any naive sequence is obtained by recycling accumulated slack: by clocking synchronous boundaries (to sequences) on early cycles, rather than altering voltage/frequency. This neither requires adjacent operations to fit into single cycles nor any rearrangement of operations.

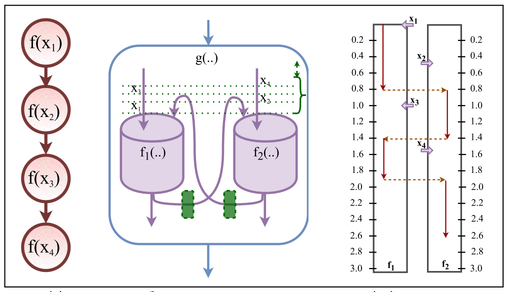

Figure 1. (i) "Asynchronous" Sequence, (ii) Execution units with transparent latching, (iii) Slack recycling

Individual summaries of our two prior proposals below:

LAGS for Spatial Fabrics: Performs timing speculation to cater to the average slack across grouped executions rather than the slack of the most critical operation itself: allowing more aggressive timing speculation. Slack estimation is performed via a feedback mechanism and extended to multi-operation sequences using statistical modeling. It targets spatial compute and employs error checking/recovery.

Recycling Data Slack in OOO Cores: Classifies executing operations into different data slack buckets based on the opcodes, data types and bit-widths. It is timing non-speculative, and thus does not need costly error-detection mechanisms. It is implemented in general OOO cores, atop the data bypass network between ALUs via transparent latching and is suitable for all general-purpose execution.

3 Extending to Approximate Computing

Axl, our proposed design for timing approximation is an extension of the slack recycling mechanism discussed in Sec.2.

① Application-level approximability is translated into operation-level approximability, to allow fine-grained control of approximation (as explored in prior works [1, 2, 5, 12]).

S#	D#	Instruction	PVT	Data	Approx	Tot. Slack
1	1	add.a r3, r1, r2, 95	10%	5%	15%	20%
2	1	or r6, r4, r5	10%	30%	0%	40%
3	1	add.a r9, r7, r8, 99	10%	8%	10%	28%
1	2	add.a r3, r1, r2, 95	10%	10%	15%	35%

Table 1. Slack breakdown of dynamic instances of multiple accurate/approximate operations.

Examples are shown in Table.1 ② Potential slack for each executing operation is identified - accounting for PVT/Data Slack and Approximate Slack (from approximation the computation can handle, if any). Timing slack vs error-rates have been analyzed to some degree by prior work [4, 10, 11] ③ The total cumulative slack is tracked across sequences of operations. ④ The sequences are then accelerated by recycling the total slack via the transparent latch based data-flow and early clocking mechanisms as shown in Fig.1.

Limitations in current approximate hardware (with specific focus on timing approximation) and benefits of our proposed design are discussed in detail below.

First order benefit: power vs. performance:

Challenge: While proposals for approximate compute hardware are numerous, primary benefits from most prior proposals is power savings (via power gating or low-voltage rails) [3, 5, 7, 8, 12]. While saving power is an important goal, so is performance speedup - accelerating approximate computations can, for instance, help meet QoS requirements.

Proposal: Our slack recycling proposals can accelerate sequences of operations via early clocking when sufficient slack accumulate (Sec.2). There is potential for performance speedup as and when sufficient slack exists. The fraction of approximate slack varies depending on each computation's approximation tolerance. The greater the slack, the more potential for speedup.

Approximate vs. accurate compute resources:

Challenge: Most applications amenable to energy-accuracy trade-offs have a fine grained inter-mingling of approximate and precise components [5]. To support this, prior general-purpose solutions employ duplicate hardware compute units (separate sets for accurate and approximate), eg. Truffle [5].They incur significant design overheads - extra resources, multiple voltage rails, scheduling logic overheads and further suffer from high potential for under-utilization.

Proposal: In our proposal, each functional unit can work as accurate compute or as approximate compute. In the former scenario, the FU only exploits PVT/Data slack but in the latter, it can estimate slack more aggressively - estimating for the required level of approximation as well (Instructions 1 vs. 2 in Table.1). This allows flexible execution of both compute forms without additional type constraints.

Temporal control at fine granularities:

Challenge: Dynamically tuned approximate hardware often employ a feedback mechanism [12] to correct the amount of approximation. In the case of timing approximation hardware (via V/F scaling) the time period of this feedback loop can be limited by granularity at which V/F can be changed. This can result in poor temporal adaptability of feedback based timing approximation hardware.

Proposal: Our proposal can perform timing approximation at very fine temporal granularities, due to the absence of V/F scaling. This feedback makes aggressive or conservative corrections to the slack estimation, and is immediately reflected in the approximate computation.

Inherent precision and environmental effects:

Challenge: Traditional timing approximation solutions lack the ability to adapt to local characteristics such as random PVT variations and inherent operation precision/data-type/data-width. In prior work, when voltage scaling is applied, the voltage is set according to average error rates across phases of execution. Setting a constant low voltage for all approximate computations has limitations, because the actual effect of the timing approximation becomes dependent on the PVT/data slack experienced by the computation. This leads to undisciplined approximation across the approximate operations.

Proposal: In our work, approximate slack is applied atop other forms of slack. Slack is modeled separately for separate compute units and for each operation. Once PVT and data slack is computed, the approximate slack for that operation (depending on the level of approximation) is added. Thus, for instance, multiple operations on the same compute unit, with the same level of approximation might have different total slack depending on their operand values (Instructions 1 vs. 4 in Table.1).

Support for multiple approximation levels:

Challenge: Approximate programs often consist of multiple approximate variables with each tolerating different levels of approximation. In this regard, recent quantitative approximability proposals, both at application [1, 2] and ISA [12] levels, allow for fluid precision reliability control and show the benefits of multiple approximate levels among operations within standard approximate applications. However, current approximate hardware proposals are unable to efficiently design for fluid precision - especially in the case of timing approximate hardware.

Proposal: In our work, multiple levels of approximation can be integrated seamlessly. In the presence of software and ISA support, each instruction carries information on the level of approximation that it requires. This is interpreted at the time of slack estimation for the operation. Similar to the assignment of data slack based on opcode etc, approximation slack is assigned based on this approximation level (Instructions 1 vs. 3 in Table.1).

4 Future Steps

We are currently studying Axl's benefits across error-tolerant applications (eg. Axbench [13]) over standard OOO and spatial substrates. We expect to explore specialized architectures such as neural network accelerators.

References

[1] Brett Boston, Adrian Sampson, Dan Grossman, and Luis Ceze. 2015. Probability Type Inference for Flexible Approximate Programming. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2015)*. ACM, New York, NY, USA, 470–487. https://doi.org/10.1145/2814270.2814301

[2] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. 2013. Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages &#38; Applications (OOPSLA '13)*. ACM, New York, NY, USA, 33–52. https://doi.org/10.1145/2509136.2509546

[3] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar. 2010. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *Design Automation Conference*. 555–560. https://doi.org/10.1145/1837274.1837411

[4] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. 2003. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *MICRO 36*. 7–.

[5] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture Support for Disciplined Approximate Programming. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII)*. ACM, New York, NY, USA, 301–312. https://doi.org/10.1145/2150976.2151008

[6] Meeta S. Gupta, Jude A. Rivers, Pradip Bose, Gu-Yeon Wei, and David Brooks. 2009. Tribeca: Design for PVT Variations with Local Recovery and Fine-grained Adaptation. In *MICRO 42*. 435–446.

[7] R. Hegde and N. R. Shanbhag. 2001. Soft digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9, 6 (Dec 2001), 813–823. https://doi.org/10.1109/92.974895

[8] J. Park, J. H. Choi, and K. Roy. 2010. Dynamic Bit-Width Adaptation in DCT: An Approach to Trade Off Image Quality and Computation Energy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 5 (May 2010), 787–793. https://doi.org/10.1109/TVLSI.2009.2016839

[9] G. S. Ravi and M. Lipasti. 2017. Timing Speculation in Multi-Cycle Data Paths. *IEEE Computer Architecture Letters* 16, 1 (Jan 2017), 84–87. https://doi.org/10.1109/LCA.2016.2580501

[10] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. 2008. VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects. *IEEE Transactions on Semiconductor Manufacturing* 21, 1 (Feb 2008), 3–13. https://doi.org/10.1109/TSM.2007.913186

[11] G. Tziantzioulis, A. M. Gok, S. M. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthasarathy. 2015. b-HiVE: A Bit-level History-based Error Model with Value Correlation for Voltage-scaled Integer and Floating Point Units. In *Proceedings of the 52Nd Annual Design Automation Conference (DAC '15)*. ACM, New York, NY, USA, Article 105, 6 pages. https://doi.org/10.1145/2744769.2744805

[12] Swagath Venkataramani, Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Quality Programmable Vector Processors for Approximate Computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/2540708.2540710

[13] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran. 2017. AxBench: A Multiplatform Benchmark Suite for Approximate Computing. *IEEE Design Test* 34, 2 (April 2017), 60–68. https://doi.org/10.1109/MDAT.2016.2630270