Synctium: a Near-Threshold Stream Processor for Energy-Constrained Parallel Applications

Evgeni Krimer, *Student Member, IEEE,* Robert Pawlowski, *Student Member, IEEE,*
Mattan Erez, *Member, IEEE,* and Patrick Chiang, *Member, IEEE*

Abstract—While Moore's law scaling continues to double transistor density every technology generation, supply voltage reduction has essentially stopped, increasing both power density and total energy consumed in conventional microprocessors. Therefore, future processors will require an architecture that can: a) take advantage of the massive amount of transistors that will be available; and b) operate these transistors in the near-threshold supply domain, thereby achieving near optimal energy/computation by balancing the leakage and dynamic energy consumption. Unfortunately, this optimality is typically achieved while running at very low frequencies (i.e. $0.1 - 10MHz$) and with only one computation executing per cycle, such that performance is limited. Further, near-threshold designs suffer from severe process variability that can introduce extremely large delay variations. *In this paper, we propose a near energy-optimal, stream processor family that relies on massively parallel, near-threshold VLSI circuits and interconnect, incorporating cooperative circuit/architecture techniques to tolerate the expected large delay variations.* Initial estimations from circuit simulations show that it is possible to achieve greater than 1 Giga-Operations per second (1GOP/s) with less than $1mW$ total power consumption, enabling a new class of energy-constrained, high-throughput computing applications.

1 INTRODUCTION

PROCESSORS at all market segments are increasingly power and energy-constrained. While current designs can be further optimized incrementally, some applications demand orders of magnitude better efficiency while providing high performance at the same time, requiring radical changes at both the system and circuit levels. For example, in some system-on-chip (SoCs) applications such as video-enabled sensor networks [1], implantable medical devices [3], [15], and mobile devices, the largest energy consumer is typically the interface with the external world, such as the analog-digital converters, wireless transceivers, and displays. In these systems, it is possible to reduce the input/output energy without compromising system goals by performing sophisticated and demanding on-chip computation. For example, recent work on neural implants shows that local on-chip parallel DSP algorithms such as feature extraction and clustering can improve the total system power by factors of 4.3 and 26, respectively [3]. Similarly, parallel video and image processing can extract features from rich sensors and reduce I/O needs, while parallel half-toning algorithms can improve the efficiency of low-power bi-stable displays [14]. Therefore, our goal is to design a new class of processors that target extremely low power while meeting the large computational demands required to shift the energy consumption burden from the I/O to the on-chip computation.

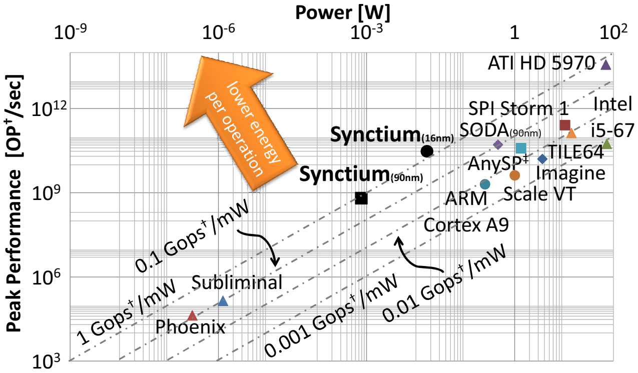


Fig. 1: Power and performance († normalized to 16-bit operations) of a range of processors showing the need for a relatively high-performance $\sim 1mW$ processor. ‡ AnySP numbers are from fig. 14 in [19].

Figure 1 shows the performance and power characteristics of several academic and industry processors. There is currently a gap in performance/efficiency tradeoffs. To support the energy optimizations mentioned above, a new class of processors that combines the energy efficiency of near-threshold processors with the high-performance throughput of massively parallel architectures is needed. With this paper, we present our initial work on addressing the major challenges of a programmable, massively parallel and ultra-efficient near-threshold processor. Specifically, we present *decoupled SIMD parallel pipelines* (DPSP) and *pipeline weaving* to overcome the very high dynamic and static timing variations that can significantly reduce the efficiency of parallel near-threshold processors.

Prior work on sub-/near-threshold processors and dynamic variability tolerance has almost exclusively focused on simple scalar pipelines [4], [6], [16], coarse-grained approaches to variability [17], and designs with large timing and device margins [9], [18]. In this paper, we re-examine this prior work in the context of programmable parallel processors that can attain both performance and efficiency by utilizing parallel stream architectures along with near-threshold design principles. We show how timing speculation and dynamic integrated recovery [6] developed for scalar pipelines must be re-architected for efficient SIMD architectures.

Initial simulations in a standard 90nm-CMOS process show a 10MHz clock frequency with a performance of 640 16-bit MOP/s while consuming less than $1mW$ in under $4mm^2$ of die area ($1.2pJ$/op). Furthermore, scaling this design to a $16nm$-CMOS process can yield 30GOP/s at $17mW$ and still

- *E. Krimer and M. Erez are with ECE at UT Austin; R. Pawlowski and P. Chiang are with EECS at Oregon State University.*
- Synctium *is inspired by* syncytium*: a cell-like structure with multiple cooperating nuclei.*

Posted to the IEEE & CSDL on 5/25/2010
DOI 10.1109/L-CA.2010.5 1556-6056/10/$26.00 © 2010 Published by the IEEE Computer Society

under $4mm^2$– a computational efficiency of a lean $560fJ/$op. In order to meet these low power metrics, new architectural mechanisms presented in this paper will be required to enable parallel operation of near-threshold computational units.

2 Background and Related Work

Research has shown that operating in the sub-threshold and near-threshold regimes leads to an order of magnitude better energy efficiency than conventional processors at the same technology node [8]. For example, a 16-bit 1024-point $.18\mu m$ FFT test-chip operating at a $V_{dd}$of $0.35V$ (with V_{th}= $0.45V$) achieves $155nJ$ per FFT at $10KHz$ [18] (equivalent to roughly $1.5pJ$ per 16-bit arithmetic operation), while an 8-bit processor achieves $540fJ/$operation at a $V_{dd}$of $0.3V$ and a frequency of $160KHz$ [9].

Unfortunately, obtaining these high computational efficiencies uncovers two critical architectural challenges: (1) the operating frequency is very low ($\sim 0.1 - 10MHz$); (2) timing variability and violation are accentuated because transistors in the near-threshold domain are extremely sensitive to process variation such as random dopant fluctuations [8].

The low clock frequency dictates the use of highly-parallel architectures for attaining the required performance. Further, in order to maximize the efficiency attained with near-threshold, programmable architectures must rely on efficient control, such as wide SIMD/vector execution [12], [19]. Unfortunately, as we explain later, wide SIMD with a large number of ALUs exacerbates the timing variability problem. Operating in the near-threshold domain is limited by variability, which continues to degrade with future CMOS scaling. Random dopant fluctuations (RDF), such as threshold voltage mismatch due to statistical fluctuations of the dopant atoms severely alter the transistor I_{on} current, thereby affecting the logic delay. As the supply voltage is reduced, the transistor no longer acts as a traditional velocity-saturated transistor and begins to behave as a bipolar device. As shown in our experiments (Figure 2) and in [8], the strong dependence of I_{on} on V_{th}in near-threshold results in as much as an 18× degradation in transistor current and logic delay across Monte Carlo variation. Even worse, because these critical RDF threshold variations are not related to any spatial distances, parallel functional units will not exhibit spatial correlations that would enable similar timing delays between vector units [5]. Prior work [9], [18] has shown that using non-minimal device sizes and longer logic chains can reduce variation, but this is still expected to be a very large problem in deep sub-micron nodes. Further, increasing transistor sizes diminishes the benefits of technology scaling.

Process-induced timing variations may be both static (requiring functional unit sparing as described later) and dynamic, input vector dependent (see bottom of Figure 2). Prior research identified mechanisms to tolerate dynamic timing variations within a scalar pipeline [2], [6]. The idea of these techniques is to dynamically identify timing errors and correct them in one of two ways: (1) flushing the pipeline and re-executing the instruction with more relaxed timing; or (2) stalling the pipeline for one cycle while waiting for the correct result to be generated, and then proceeding with execution. The first approach has a larger performance penalty, but is easier to implement in high-speed and complex pipelines, while the second approach has a smaller performance and power penalty.

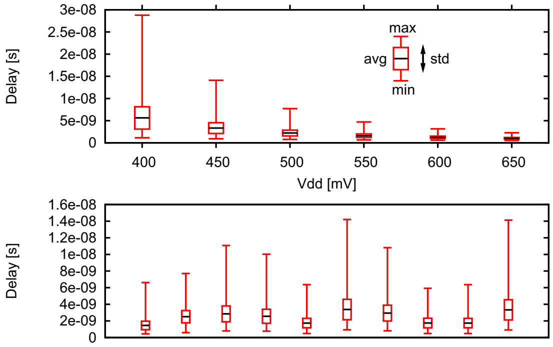


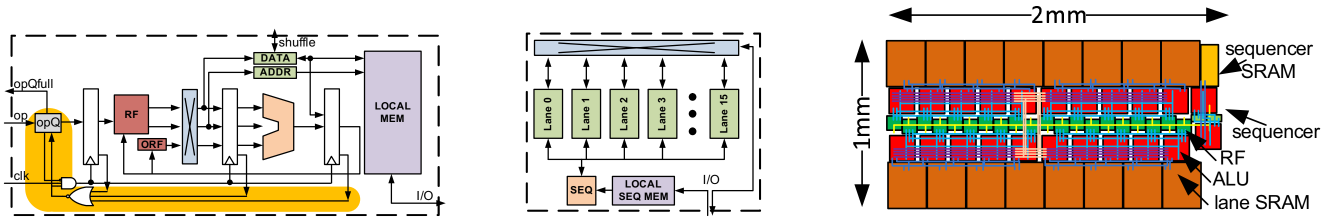
Fig. 2: Results of random-dopant fluctuation Monte Carlo experiments for delay of a 16-bit multiplier ($90nm$, V_{th}=0.4V): worst-case delay as function of V_{dd}(top) and delay of 10 random input vectors at V_{dd}= $0.45V$ (bottom)

Unfortunately, extending either the stall or flush approach to conventional wide parallel SIMD pipelines is problematic. In a conventional parallel design, all parallel functional units operate in lock step off of a single clock, such that any error encountered in single stage will result in a stall or flush across all the functional units within that processing element. This multiplies the performance/power penalty of tolerating an error by the pipeline width. This overhead is compounded by the fact that the likelihood of an error occurring is effectively larger with a parallel pipeline because the chance of a timing violation in one functional unit is independent from the other functional units that are processing different inputs (see Figure 2). This is depicted in Figure 4a that shows preliminary calculations of the expected resulting fraction of peak throughput as a function of the width of a 5-stage pipeline and the probability of error. The wider the pipeline, the steeper the degradation rate with respect to timing violation probability.

3 Synctium Processor

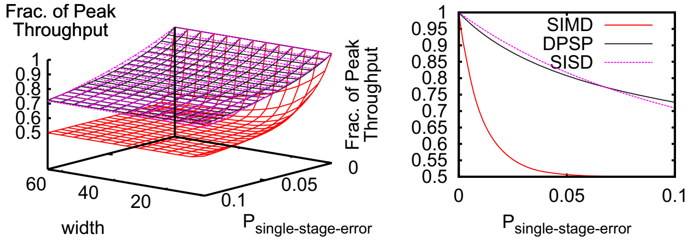
Synctium is a parallel near-threshold stream processor, that achieves near-optimal energy efficiency with high throughput by relying on parallel, low-frequency, near-threshold circuits. Because of the low operating frequency, it is paramount to provide a large number of ALUs and, unlike conventional super-threshold designs, Synctium is constrained by area rather than power. Therefore, to emphasize simplicity and area efficiency, the baseline architecture is a fairly-traditional, multi-core wide-SIMD processor. Each core has multiple *Processing Elements* (PEs), each composed of an instruction sequencer and 16 execution lanes with each lane accessing a 16KB local memory. Initial estimates of area and power for a future $16nm$ CMOS processor (to be detailed in a full-length paper) are under $4mm^2$ of area with a peak performance of 30 16-bit GOP/s consuming less than $17mW$. We also propose a $90nm$ configuration that we intend to prototype. This configuration will consume less than $1mW$, achieve up to 640 16-bit MOP/s, and also require just $4mm^2$ of die area. The analysis was done by synthesizing the RTL for a 16 bit MADD unit and obtaining its energy and area costs from extracted layout simulations in $90nm$. We used a conventional 2-read, 1-write, RF generator and a 1-read, 1-write SRAM generator and scaled the results up by 30% to account for modifications needed for near-threshold operation. Wires are modeled as lumped RC-wires, with a 250fF/mm capacitance extrapolated from the ITRS roadmap.

Our architectural innovations are not in this straight-forward baseline architecture, but rather in the way in which we ad-



(a) Baseline PE SIMD lane (b) Baseline PE (c) PE floor plan

Fig. 3: Overall design of a Synctium lane, processing element, and initial floorplan. Each lane contains a fused 16-bit MADD ALU, with two inputs provided by a standard RF and the third input by a small and efficient operand register file. The highlighted paths and components within a lane enable DPSP to tolerate dynamic timing variability. Note that the floorplan includes redundant components and paths. Sizes mentioned are for 90nm technology.



(a) (b)

Fig. 4: Expected fraction of throughput of a 5-stage SIMD pipeline and a 5-stage decoupled parallel SIMD pipelines with decoupling queues as a function of the total probability of error compared to a SISD pipeline (legend for both figures appears in (b)): (a) for varying SIMD width; (b) for 16-wide SIMD.

dress the challenges of extreme static and dynamic timing variations in near-threshold. We propose two architectural mechanisms that complement prior work on variation tolerance in parallel architectures [10]. *Decoupled parallel SIMD pipelines* extends the work on timing speculation to parallel pipelines, providing tolerance of input-dependent and dynamic variations. *Pipeline weaving* provides efficient fine-grained spatial redundancy within the parallel pipeline.

Tolerating Dynamic Timing Variations Using DPSP

In order to address dynamic variations, where the ALU pipeline delays dynamically depend on specific instructions and data inputs, we propose a *decoupled parallel SIMD pipeline* (DPSP) microarchitecture. With DPSP, all functional units in the SIMD organization still execute the same instructions in the same order, but parallel pipelines are allowed to slip with respect to one another so that they can tolerate timing violations independently. Thus, timing violations which are input dependent will occur randomly in all parallel pipes, such that the average throughput will be optimal. To enable independent timing recovery in each lane, we use *decoupling queues* and *micro-barriers*.

First, we replace the pipeline latch between the decode stage and the register access and execute stages of the sequential parallel pipeline with a set of shallow FIFO *decoupling queues* (one per SIMD lane). When a timing violation is detected in one of the lanes, only that particular lane stalls and initiates local recovery. The sequencer continues to place instructions into the decoupling queues and all non-faulting lanes execute normally. If timing violations are equally distributed between operations and lanes, the average execution rate would be identical across the PE. The entire parallel pipeline stalls only if violations are not balanced between the lanes and one of the decoupling queues fills up. The DPSP concept, shown in Figure 3a can be generalized by placing additional queues between additional pipeline stages to allow for re-balancing of the violations internally within each pipeline. Preliminary results based on a simple throughput simulation indicate that DPSP can indeed maintain high parallel throughput in the face of input-dependent timing variations (Figure 4a). Figure 4b shows a cross-section of the 3D curve for a 16-wide SIMD pipeline and demonstrates how DPSP experiences much lower and more gradual degradation in throughput as the likelihood of a violation grows, maintaining a $50-80\%$ better throughput than a simple SIMD with recovery. Note that these queues are for pipeline registers and control and are entirely different from prior approaches that use queues to communicate data for dataflow style computation.

Additional mechanisms are necessary to ensure the correct execution of shuffle operations, as well as cross-lane scatters and gathers. For shuffles, Each lane must wait for its producer lane to place the correct value on the interconnect (switch or shuffle network) before it can complete the shuffle operation. Similarly, gathers cannot access memory that is shared between lanes until all previous writes complete, while scatters must store values in memory in program order. To address these issues, we introduce *micro-barriers* between the SIMD lanes of a single PE. In their simplest form, micro-barriers require each communication operation (a shuffle, a scatter, or a gather that follows a store) to block the DPSP and act as implicit barrier synchronization points between the lanes. Each lane notifies the other lanes that it has reached a blocking operation, and then waits until all lanes are ready to execute the same instruction in lock step. Because they are performed within a near-threshold PE, the micro-barriers can be done in less than a cycle and only impact performance if the lanes experience unbalanced timing violations between synchronizations.

Pipeline Weaving for Addressing Static Timing Uncertainties

While DPSP can tolerate dynamic variations, process-dependent static variations require a different approach. For example, if a single wide-SIMD pipeline has large delay variability between ALUs, it will have poor energy efficiency, as faster components will have higher leakage and thus need to be clocked faster to reach optimal energy-per-computation targets. Additionally, process variations can lead to non-functioning pipeline components, which may reduce both yield and performance. Prior approaches addressed this challenge at a coarse granularity, adjusting the supply voltage for entire PEs [10], [17]. Fine-grained approaches, such as voltage interpolation [13], are problematic because of the large overheads involved in providing the numerous supplies necessary for the massive number of low-power computational units.

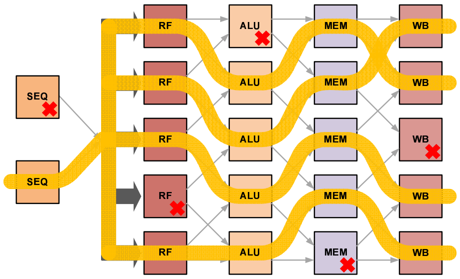


Fig. 5: Weaved-pipeline sparing for DPSP and SIMD PEs. Shared pipeline components are replicated once and, in this example, a sparing of $\frac{1}{4}$ is added to the parallel SIMD path. Components that failed testing are marked with an X and are disabled along with their wires.

In addition to utilizing coarse-grained techniques, we propose a new complementary technique for intra-PE sparing called *pipeline weaving* (Figure 5). With this technique, we duplicate the shared fetch and decode stages of the parallel SIMD pipeline, but only add a small number of redundant parallel components. Each component is connected to two other downstream components, such that the overall parallel pipeline exhibits a 'weave' pattern of local wires. This is an adaptation of the two-dimensional PE-array sparing design originally presented in [11] for a SIMD pipeline. Figure 5 also shows how this weaved pipeline design can be disabled and reconfigured around units or pipeline stages that do not meet minimum specification during configuration time. Shaded components are disabled as are all the wires that connect them (indicated with a red X in the figure).

Pipeline weaving is similar to the recently presented StageNet [7] architecture, with two important differences. First, we specialize our technique for the DPSP pipeline described above and do not replicate all components of the pipeline. Second, the weaved SIMD pipeline approach does not use centralized switches as required by StageNet and instead relies entirely on local wires. This is particularly important in the near-threshold domain, because of the much higher energy cost of communication (relative to computation).

4 Summary and Future Work

This paper briefly presents Synctium stream architecture that achieves near energy-optimal operation by combining efficient parallel computation using near-threshold circuits. Synctium fills a gap in the programmable processor tradeoff space by offering the efficiency of near-threshold circuits (which have so far been constrained to simple sequential control pipelines), combined with the potential of massively-parallel processing. To the best of our knowledge, this is the first attempt at a programmable massively parallel processor entirely in near-threshold. The combination of parallelism and near-threshold operation raises multiple challenges, with the most important architectural challenge being the high static and dynamic timing variability. We propose two new techniques that extend and complement prior work: *Decoupled SIMD Parallel Pipelines* for dynamic variation handling, and *Pipeline Weaving* to compensate for the static variations. When combined with coarse-grained approaches for managing PE voltage and frequency, the overall architecture offers unparalleled efficiency while still meeting the performance requirements of important ultra-low-power applications.

Our next research steps are to refine the architectural and circuit tradeoffs, while more carefully exploring the application space. Concurrently, we are working towards fabricating a $90nm$ prototype chip to validate our unique techniques for massively-parallel near-threshold operation and demonstrate the potential of Synctium. We will also verify near-threshold process variations models using the prototype to better guide and evaluate decisions on DPSP width and depth and weaving redundancy.

References

[1] I. Akyildiz, T. Melodia, and K. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921–960, 2007.

[2] K. Bowman, J. Tschanz, C. Wilkerson, S. Lu, T. Karnik, V. De, and S. Borkar, "Circuit techniques for dynamic variation tolerance," in *DAC'09*. ACM, 2009, pp. 4–7.

[3] R. Chandler, S. Gibson, V. Karkare, S. Farshchi, D. Markovic, and J. Judy, "A system-level view of optimizing high-channel-count wireless biosignal telemetry." in *EMBC'09*, 2009.

[4] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in CMOS circuits," *Proc. of the IEEE*, vol. 83, no. 4, pp. 498–523, 1995.

[5] N. Drego, A. Chandrakasan, and D. Boning, "All-digital circuits for measurement of spatial variation in digital circuits," *IEEE JSSC*, vol. 45, pp. 640–651, March 2010.

[6] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," *IEEE MICRO*, pp. 10–20, 2004.

[7] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke, "The StageNet fabric for constructing resilient multicore systems," in *MICRO'08*. IEEE Computer Society, 2008, pp. 141–151.

[8] S. Hanson, B. Zhai, K. Bernstein, D. Blaauw, A. Bryant, L. Chang, K. K. Das, W. Haensch, E. J. Nowak, and D. M. Sylvester, "Ultralow-voltage, minimum-energy cmos," *IBM J. Res. Dev.*, vol. 50, no. 4/5, pp. 469–490, 2006.

[9] S. Hanson, B. Zhai, M. Seok, B. Cline, K. Zhou, M. Singhal, M. Minuth, J. Olson, L. Nazhandali, T. Austin, D. Sylvester, and D. Blaauw, "Exploring variability and performance in a sub-200 mv processor," *JSSC*, no. 4, April 2008.

[10] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "A 300mV 494GOPS/W Reconfigurable Dual-Supply 4-Way SIMD Vector Processing Accelerator in 45nm CMOS," in *ISSCC'09*, 2009, pp. 260–261.

[11] I. Koren and A. Singh, "Fault tolerance in VLSI circuits," *Computer*, vol. 23, no. 7, pp. 73–83, 1990.

[12] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanovic, "The Vector-Thread Architecture," in *ISCA'04*, 2004.

[13] X. Liang, G. Wei, and D. Brooks, "Revival: A variation-tolerant architecture using voltage interpolation and variable latency," in *ISCA'08*, 2008.

[14] Mirasol, "MEMS Drive IMOD Reflective Technology," http://www.mirasoldisplays.com/mobile-display-imod-technology, 2010.

[15] K. Oweiss, A. Mason, Y. Suhail, A. Kamboh, and K. Thomson, "A scalable wavelet transform VLSI architecture for real-time signal processing in high-density intra-cortical implants," *IEEE Tran. Cir. and Sys. I*, vol. 54, no. 6, pp. 1266–1278, 2007.

[16] M. Seok, S. Hanson, Y. Lin, Z. Foo, D. Kim, Y. Lee, N. Liu, D. Sylvester, and D. Blaauw, "The Phoenix Processor: A 30pW platform for sensor applications," in *VLSI Circuits Symposium*, 2008, pp. 188–189.

[17] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, A. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, A. Tran, Z. Xiao *et al.*, "A 167-Processor Computational Platform in 65 nm CMOS," *JSSC*, vol. 44, no. 4, p. 1, 2009.

[18] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *JSSC*, vol. 40, no. 1, pp. 310–319, 2005.

[19] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "AnySP: Anytime Anywhere Anyway Signal Processing," *ISCA'09*, 2009.