19.4 An Adaptive Clock Management Scheme Exploiting Instruction-Based Dynamic Timing Slack for a General-Purpose Graphics Processor Unit with Deep Pipeline and Out-of-Order Execution

Tianyu Jia, Russ Joseph, Jie Gu

Northwestern University, Evanston, IL

Cycle-by-cycle dynamic timing slack (DTS), which represents extra timing margin from the critical-path timing slack reported by the static timing analysis (STA), has been observed at both program level and instruction level. Conventional dynamic voltage and frequency scaling (DVFS) works at the program level and does not provide adequate frequency-scaling granularity for instruction-level timing management [1]. Razor-based techniques leverage error detection to exploit the DTS on a cycle-by-cycle basis [2]. However, it requires additional error-detection circuits and architecture-level co-design for error recovery [3]. Supply droop-based adaptive clocking was used to reduce timing margin under PVT variation, but does not address the instruction-level timing variation [4]. Recently, instruction-based adaptive clock schemes have been introduced to enhance a CPU's operation [5-6]. For example, instruction types at the execution stage were used to provide timing control for a simple pipeline structure. However, this scheme lacks adequate consideration for other pipeline stages whose timing may not be opcode dependent [5]. In [6], the instruction-execution sequence was evaluated at the compiler level with the timing encoded into the instruction code. The scheme considers all pipeline stages but relies on in-order execution of instructions for proper timing encoding from the compiler.

Several unaddressed critical issues limit the exploitation of DTS into more complex CPU or general-purpose graphics-processor unit (GPGPU) platforms, as shown in Fig. 19.4.1. First of all, all the existing work are based on a simple in-order pipeline structure, where the instruction execution sequence can be predicted ahead of time. For a more complex GPU architecture, there could be multiple vector SIMD units within one compute unit. The instructions issued to each SIMD and other execution units are dynamically scheduled across multiple wavefronts, leading to the difficulties predicting the runtime execution of instructions at the compiler stage. Secondly, existing works are based on shallow pipeline stages with a single core. Fig.19.4.1 shows the simulated dynamic timing slack distribution across 10 pipeline stages of a GPGPU core used in this work [7]. With a deep pipeline and multicore operation, every stage may become a timing bottleneck rendering difficulty in timing management tasks across pipeline stages and requiring clock synchronization across domains, e.g. multi-cores and memory. Thirdly, solutions are lacking when non-execution pipeline stages, e.g. the fetch stage, become the timing bottleneck, which limits the performance benefit by applying DTS techniques.

To overcome the challenges, we present an instruction-driven adaptive clock management scheme with the following features: 1) Use of on-chip critical-path (CP) "messengers", which are different from critical-path monitors, to help predict the appearance of the critical path one cycle earlier for dynamic clock scaling. The combination of real-time CP messengers and multi-stage timing arbiter circuits provide an in-situ cycle-by-cycle clock management. 2) Hierarchical clock circuitry including a global PLL, local delay-locked lock (DLL), and an asynchronous clock interface are used for multi-core clock management across clock domains. 3) An elastic pipeline clocking scheme, developed to mitigate timing bottlenecks within non-execution stages and enhance error-tolerant machine-learning (ML) applications.

Figure 19.4.2 shows the system diagram and the pipeline architecture of the implemented processor, which is a simplified design of an open-source GPGPU architecture following AMD Southern Islands ISA [7]. The chip includes two compute units (CUs), each with 10 pipeline stages including Fetch, Wavepool, Decode, Issue (two stages), Execution (four stages) and Writeback. Inside the execution stage, there are 4 SIMD vector modules with both integer and floating-point modules, and a scalar ALU. The two CUs share an L2 memory bank for the data communication through an asynchronous interface. Fig. 19.4.2 shows the critical-path messengers, which are developed to pessimistically predict the appearance of critical paths in the next clock cycle. Two types of messengers are used. For non-execution stages, e.g. Fetch, Wavepool, etc., the critical paths are traced at the roots of the operation, i.e. D pin of flip-flops. The messenger is formed by detecting transitions on critical signals that will trigger the critical paths, delivering a clock-cycle lead time for the clock controller to react. For the Execution stage, the real-time issued instruction opcodes are used as messengers. The selection of messengers is determined through an internally developed transitional static timing analysis approach, where cycle-by-cycle critical paths are identified at given pipeline conditions. Overall, less than 1% of registers are selected as messengers, leading to negligible area overhead. All the messengers are combined inside a timing arbiter, which issues the final clock period value based on the worst-case timing from all messengers.

Figure 19.4.3 shows the hierarchical clocking scheme enabling instruction-driven clock management. A global PLL delivers the clock to the local DLL for each CU and associated L1 cache. Compared with a PLL, the use of the DLL reduces area overhead needed for clocking each CU. All the phases on the DLL delay chain are equally delayed and carefully matched, with one of them dynamically selected through a glitch-free multiplexer for cycle-by-cycle clock period adjustment. The DLL can be locked with a programmable number of stages, from 30 to 60, providing wide locking range and fine phase selection resolution. At each CU, a timing arbiter combines "messengers" from all pipeline stages and arbitrates the clock period for the current cycle. The timing adjustment step for each messenger is provided by a small register file. For simplicity, only dynamic shrinking of the clock period is allowed. Due to the dynamic clock period of each CU, an asynchronous interface is used to establish data communication between two CUs and L2 caches. A double-triggered data buffer is used to ensure proper latching of the incoming data even with misalignment of clock phases.

As shown in Fig.19.4.4, an elastic clocking mode is created to allow additional timing margin for non-Execution stages to mitigate critical timing bottlenecks. Essentially, the clock period for each pipeline stage can be redistributed. Tunable delay modules are inserted at the clock root of each pipeline stage rendering elasticity of timing margin for each stage. As a result, the DLL output clock period remains short due to redistributed pipeline clock. To enhance the performance of the GPGPU for ML applications, e.g. multiplication operations of neural networks, the elastic clocking scheme can apply more aggressive timing for the execution stage where timing errors are tolerable."

Figure 19.4.5 shows the measurement results. The clock for each CU core and selected critical-path messengers were routed out on PCB boards for probing. Measured waveforms confirmed the proper clock management based on the detected CP messengers. Data from RFs and caches were scanned out for verification of proper execution. The instruction-driven adaptive clock scheme has been tested using eight kernel programs, achieving up to 18.2% performance improvement or equivalently 30.4% energy savings using a lower supply voltage. Fig. 19.4.6 shows the additional energy benefit of using the elastic pipeline clocking for ML. The MNIST dataset is tested using floating point SIMD for inference. When allowing timing error to happen, 4.6% additional energy savings is obtained without accuracy degradation. At 2% accuracy loss, an additional 8% energy saving can be achieved. The scheme was also tested from nominal 1V down to 0.4V, showing similar performance gains. Comparing with prior work [1], this work achieves fine-grained instruction-level clock management for a GPU processor and obtains significant benefits beyond program-level DVFS. Comparing with Razor techniques [2], there is no error detection flip-flops or pipeline recovery needed. The benefit can be maintained down to 0.4V as the improvement does not trade off hold timing. Compared with previous supply droop-based adaptive clocking [4], this work exploits deterministic instruction-level timing variation, and overcomes multicore, deep pipeline, multi-thread execution challenges, faced by previous simple CPU operations [5-6]. For each CU, the timing arbiter and messengers contribute to 1.7% area overhead. 1.6% additional area was introduced by the min-delay padding for the special elastic-clocking mode. The DLL design with elastic-tunable buffers consumes about 1.5% area of the CU. Fig. 19.4.7 shows the die micrograph.

Acknowledgements:
This work was supported in part by the National Science Foundation under grant numbers CCF-1116610 and CCF-1618065.

References:

[1] P. Meinerzhagen, et al., "An Energy-Efficient Graphics Processor Featuring Fine-Grain DVFS with Integrated Voltage Regulators, Execution-Unit Turbo, and Retentive Sleep in 14nm Tri-Gate CMOS," *ISSCC*, pp. 38-39, 2018.
[2] Y. Zhang, et al., "iRazor: 3-Transistor Current-based Error Detection and Correction in an ARM Cortex-R4 Processor," *ISSCC*, pp. 160-161, 2016.
[3] S. Kim, and M. Seok, "Variation-Tolerant, Ultra-Low-Voltage Microprocessor with a Low-Overhead, Within-a-Cycle In-Situ Timing-Error Detection and Correction Technique," *IEEE JSSC*, vol. 50, no. 6, pp. 1478-1490, 2015.
[4] K. Bowman, et al, "A 16nm Auto-Calibrating Dynamically Adaptive Clock Distribution for Maximizing Supply-Voltage-Droop Tolerance Across a Wide Operating Range," *ISSCC*, pp. 152-153, 2015.
[5] J. Constantin, et al., "DynOR: A 32-bit Microprocessor in 28 nm FD-SOI with Cycle-by-Cycle Dynamic Clock Adjustment," *ESSCIRC*, pp. 261-264, 2016.
[6] T. Jia, et al., "An Instruction Driven Adaptive Clock Phase Scaling with Timing Encoding and Online Instruction Calibration for a Low Power Microprocessor," *ESSCIRC*, pp. 94-97, 2018.
[7] R. Balasubramanian, et al., "Enabling GPGPU Low-Level Hardware Explorations with MIAOW: An Open-Source RTL Implementation of a GPGPU," *ACM TACO*, vol. 12, no. 2, article 21, 2015.

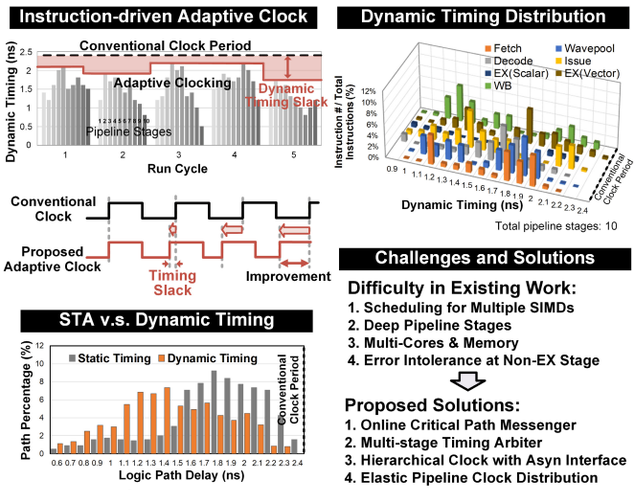


Figure 19.4.1: Simulated dynamic timing slack at instruction level for a 10-stage GPGPU architecture, and the adaptive clock challenges to exploit DTS for the deep pipeline, multicore GPU architectures.

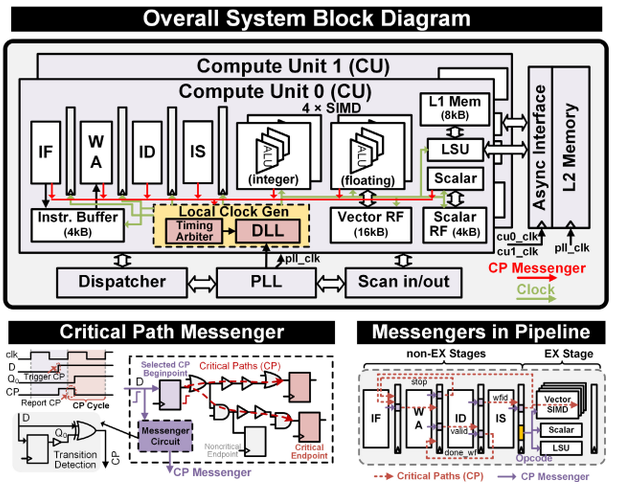


Figure 19.4.2: Block diagram of the implemented open-source GPU processor, and the critical-path messenger design, which predicts that the CP will be exercised one cycle in advance.

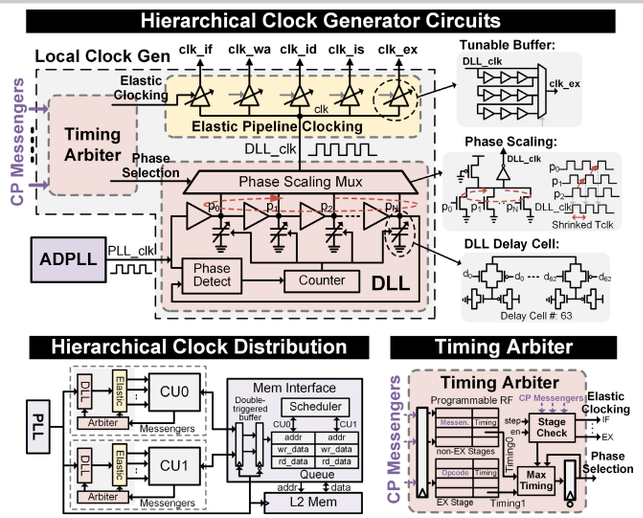


Figure 19.4.3: Design details of the proposed hierarchical clock-management scheme.

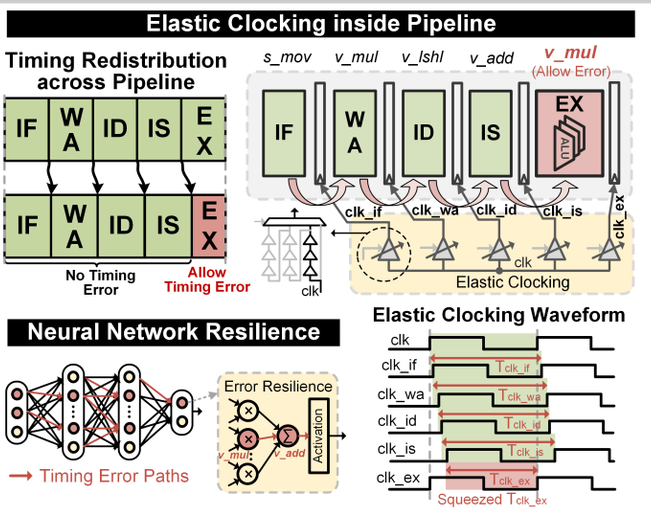


Figure 19.4.4: Elastic pipeline timing distribution across all stages to exploit machine-learning error resilience.

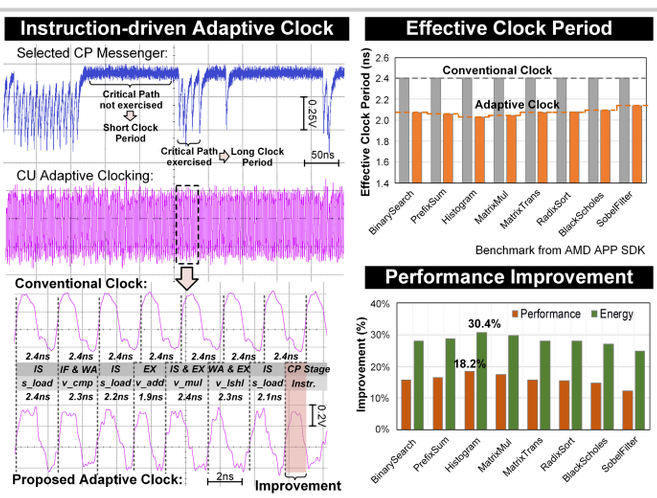
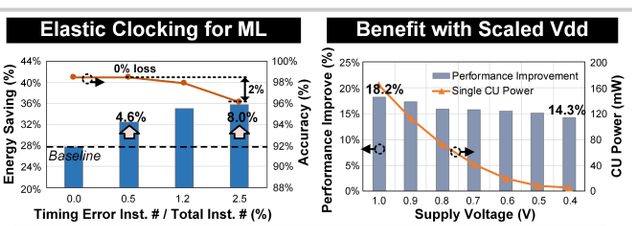


Figure 19.4.5: Measured instruction-driven adaptive clock guided by the messenger signals (waveform distortion due to impedance matching on PCB board) and the performance improvement for eight kernel programs.

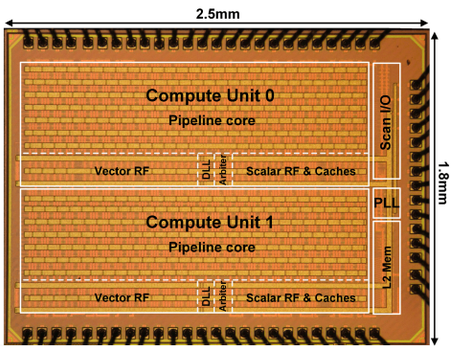


Comparison with Prior Work

	[1]	[2]	[5]	[6]	This work
Process	14nm	40nm	28nm	65nm	65nm
Processor	GPU	CPU	CPU	CPU	GPU
Granularity	Program	Instruction	Instruction	Instruction	Instruction
Pipeline Stages	N/A	8	6	6	10
Core Number	18	1	1	1	2
Fsw (MHz)	400	843	509	625	417
Energy Saving	32%	41%	15%	28%	30.4%
Clock/Voltage Management	SCVR+ DLDO	PLL+ Razor	DCO+ LUT	PLL+ Compiler	PLL+DLL+ Messenger
ML Enhanced	No	No	No	No	Yes
Power (mW)	N/A	48@572M	112@1.3G	107@625M	468@417M
Area Overhead	N/A	13.6%	~7.3%	5.4%	1.7% (Arbiter) 1.6% (Elastic Hold)

Figure 19.4.6: Benefit from the elastic pipeline clocking for machine-learning applications, and the improvement with scaled voltage levels.

19



Technology	65nm CMOS
Die Area	4.5mm^2
CU Power	164mW (pipeline) 52mW (RF & Cache)
PLL Power	6.8mW
Total Power	468mW
Base Freq.	417MHz
Nom Vdd	1V
Vdd Range	0.4-1V

Figure 19.4.7: Die micrograph and design details.

978-1-5386-8531-0/19/$31.00 ©2019 IEEE