Modeling dynamic register redistribution in Accel-Sim

François Trahay[1] Pawel Lacek[2]
[1]Télécom SudParis
[2]Institut Polytechnique de Paris
francois.trahay@telecom-sudparis.eu

Keywords: GPU architecture, Hopper, register allocation, warp specialization, trace-driven simulation, performance modeling.

I Context

Modern GPU kernels assign different tasks to groups of threads: some move data, while others perform matrix computation. These roles can need very different amounts of register storage. Hopper permits runtime adjustment of register allowances through `setmaxnreg`. A useful simulator must represent the resulting capacity transfers and execution delays [4].

Accel-Sim 2.0 provides Hopper execution models, including asynchronous matrix operations and synchronization, using the GPGPU-Sim performance model [1, 2, 3]. In the inspected frontend, however, `USETMAXREG` maps to `OP_NOP`. Existing block-admission accounting reserves registers but does not itself implement runtime redistribution. This provides a concrete starting point for a new simulator capability [5, 6, 7].

II Goal of the Project

Develop and evaluate an Accel-Sim extension for dynamic register redistribution in Hopper kernels, beginning with controlled examples and progressing to warp-specialized matrix workloads. The main tasks are:

- Analyze compiled instructions and traces to identify register-management modes, target counts, initial allocation, and synchronization or retry behavior. Distinguish documented semantics from hardware observations and modeling assumptions [4].
- Implement instruction decoding and a register pool for each resident threadblock. Integrate acquisition, release, and waiting or retries with warp scheduling, preserving allocation state throughout the block's lifetime and keeping redistribution separate from block-admission reservations.
- Evaluate correctness, acquisition delays, execution time, and simulation overhead using controlled and warp-specialized workloads. Compare with hardware measurements where available, and provide reproducible experiments covering delayed release, repeated adjustments, independent blocks, and resource reuse.
- Propose extensions for physical register placement, register-bank effects, and additional GPU architectures, outlining their feasibility and remaining challenges.

The initial scope is redistribution within a threadblock on one Hopper GPU. Support a defined set of compiler-generated modes; distinguish any intermediate blocking approximation from compiled retry behavior. Physical register remapping and Blackwell support are possible future directions.

III What You Will Learn

- GPU registers, residency, warp specialization, and resource lifetimes;
- native-instruction tracing and the distinction between PTX and SASS;
- simulator decoding, instruction issue, runtime state, and synchronization;

- invariant-based testing, microbenchmark design, and hardware correlation.

IV What You Should Like

- C++ systems programming and computer architecture;
- tracing execution through a substantial existing codebase;
- designing experiments that explain subtle performance behavior.

References

[1] Junrui Pan, Weili An, Cesar Avalos Baddouh, Christin David Bose, Ni Kang, Aaron Barnes, Ahmad Alawneh, Fangjia Shen, Yechen Liu, Anusuya Nallathambi, Atthin Chandrashekar, and Timothy G. Rogers. *Architecting the Next Generation of Asynchronous, Distributed GPUs for the AI Era*. arXiv:2608.22602, 2026. https://arxiv.org/abs/2608.22602.

[2] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. *Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling*. ISCA, 2020. https://doi.org/10.1109/ISCA45697.2020.00047.

[3] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. *Analyzing CUDA Workloads Using a Detailed GPU Simulator*. ISPASS, 2009. https://doi.org/10.1109/ISPASS.2009.4919648.

[4] NVIDIA. *PTX ISA: setmaxnreg*. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#miscellaneous-instructions-setmaxnreg.

[5] Accel-Sim. *Version 2.0.0 release*. https://github.com/accel-sim/accel-sim-framework/releases/tag/v2.0.0.

[6] Accel-Sim frontend, commit `64653015`. https://github.com/accel-sim/accel-sim-framework/blob/64653015f85fb5664c84a10f48527e8897d289d0/gpu-simulator/ISA_Def/hopper_opcode.h.

[7] Accel-Sim GPGPU-Sim backend, commit `91880c53`. https://github.com/accel-sim/gpgpu-sim_distribution/blob/91880c53383d5a6a6742bfb1be2c5f34e39c7871/src/gpgpu-sim/gpu-sim.cc.