Memory Persistency Models for GPUs

Peiyi Li
pli11@ncsu.edu

Abstract

Persistent memory or non-volatile memory (NVM) has high density and high speed compared to DRAM. So the paper [1] explores memory persistency models for GPUs, and adapts, re-architect and optimizes CPU persistency models for GPUs. The goal of this project is to implement the paper [1], and evaluates results using GPGPU-Sim 4.0 [2].

Introduction

The paper [1] proposes strict persistency and relaxed (epoch) persistency models for GPUs. And for the epoch persistency model, three epoch granularities: kernel level, CTA level and loop level are proposed based on the characteristics of a kernel. Finally, the paper [1] evaluates results using GPGPU-Sim simulator version 3.2.2 [3].

Compared to the paper [1], this work evaluates results using the new version of GPGPU-Sim 4.0 instead of using the old version 3.2.2. And in order to evaluate results on GPGPU-Sim 4.0, GPGPU-Sim 4.0 is modified to support CLWB, L2WB, PCOMMIT instructions which would be used in memory persistency models.

Implementation

A. Implementing CLWB instruction

CLWB instruction was implemented on the GPGPU-Sim in order to write the dirty cache line to the memory controller. The main changes are made in gpgpu-sim_distribution/src/gpgpu-sim/gpu-cache.cc. The basic idea is that if the CLWB instruction was detected in the process of cache access, we will check if the cache line is a dirty cache line and send the dirty cache line to the memory.

B. Implementing L2WB instruction

L2WB instruction was implemented on the GPGPU-Sim in order to write back all dirty cache lines in the L2 into the memory controller. The main changes are made in gpgpu-sim_distribution/src/gpgpu-sim/gpu-cache.cc. New function data_cache::nvm_wb_timing was added in order to check all the cache lines in L2, and if there are dirty cache lines, all the dirty cache lines will be sent to the memory.

C. Implementing PCOMMIT instruction

PCOMMIT instruction was implemented on the GPGPU-Sim in order to write the data in the write pending queue (WPQ) to persistent memory if the WPQ in the memory controller is not durable. PCOMMIT instruction needs to be broadcasted to all memory controllers to flush all their WPQs. The process of flushing WPQs is in the function of memory_partition_unit::dram_cycle(). When the PCOMMIT instruction is detected in the front of WPQ, an acknowledgement will be sent back to the shader, meaning all the data in the WPQ which before this PCOMMIT instruction have been written to persistency memory.

D. Implementing membar instruction

The membar/fence instructions enforce an ordering of memory operations. And in the GPGPU-Sim, the membar instruction was implemented by modifying the function shader_core_ctx::warp_waiting_at_mem_barrier, so that at a member, we will wait its prior memory operation from the same warp receive their acknowledgement.

Experimental Methodology

Experiments are done by using GPGPU-Sim 4.0 [3]. The simulation configurations of GPGPU-Sim are shown in table Ⅰ. All the benchmarks are compiled using NVCC version 10.1.

Table Ⅰ: Baseline architecture configuration

Architecture	NVDIA Volta QV100
Number of SMs	80
Per-SM warp schedulers	4 Greedy-Then-Oldest schedulers
Per-SM limit	2048 threads, 64 warps, 32 CTAs
Per-SM L1D-cache	32KB
Per-SM SMEM	96KB
L2 cache	6MB
Memory Controller	32 channels, 2 L2 banks/channel, FR-FCFS scheduler
NVMM latency	Read: 200 cycles; Write: 600 cycles
DRAM latency	Read: 150 cycles; Write: 150 cycles

Parboil benchmarks [4] are used in the experiments. And the kernels are listed in Table Ⅱ. As each benchmark may have multiple kernels, a number followed by a benchmark name is used to differentiate these kernels. All the kernels are classified into one of the three categories according to their execution time. Short-running kernels are labelled ‘S’, long-running kernels with short-running CTAs are labelled ‘LS’, and long-running kernels with long running CTAs are labelled ‘LL’.

Table Ⅱ: Parboil Benchmarks

kernel	Kernel name	Type
bfs_v1	BFS_kernel	S
bfs	BFS_in_GPU_kernel	LL
histo1	histo_prescan_kernel	S
histo2	histo_intermediates_kernel	S
histo4	histo_final_kernel	S
mri-gridding-scan1	scan_L1_kernel	S
mri-gridding-scan2	scan_inter1_kernel	S
mri-gridding-sort1	splitSort	LS
mri-gridding-sort2	splitRearrange	LS
mri-q1	ComputePhiMag_GPU	S
mri-q2	ComputeQ_GPU	LS
sad2	larger_sad_calc_8	S
sad1	mb_sad_calc	LS
lbm	performStreamCollide_kernel	LS
stencil	naive_kernel	LS
spmv	spmv_jds	LS
sgemm	mysgemmNT	LS

Besides the Parboil benchmarks, in-memory database Mega-KV [5] and ProgPOW, a proof-of-work algorithm in blockchain applications [6] are also used in the experiments, and the kernels are listed in Table Ⅲ and Table Ⅳ.

Table Ⅲ: Mega-KV Benchmarks

kernel	Kernel name	Type
search	hash_search	S
insert	hash_insert_cuckoo	S
delete	hash_delete	S

Table Ⅳ: ProgPOW Benchmarks

kernel	Kernel name	Type
ethash_calculate_dag_item	ethash_calculate_dag_item	LS
progpow_search	progpow_search	LS

Experimental Results for Parboil Benchmark

A. Short-Running Kernels

For each kernel, we show the normalized execution time to the nvm baseline. And the performance results of the short-running kernels on GPGPU-Sim are shown in Figure 1.

The overhead of the kernel-level epoch persistency model over the nvm baseline is 49.2% and 50.1% on average for EP_K(epoch model with durable WPQs) and EP_K_pct (epoch model with volatile WPQs). And when the kernel has less execution time, the overhead of using a kernel-level epoch persistency model is higher because of the higher cost of using L2WB instruction. For example, kernel mri-q1 has the least execution time among the short-running kernels we used, the overhead of using EP_K is 90.0% and the overhead of using EP_K_pct is

90.6%.

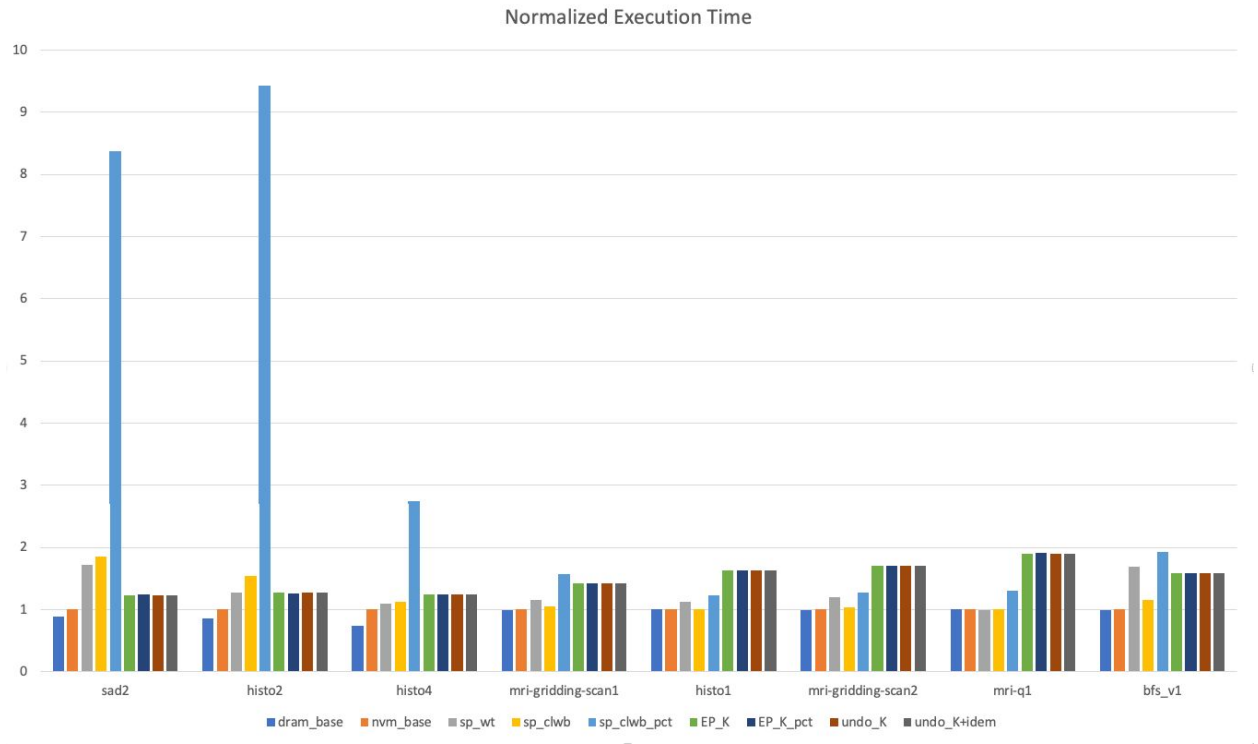

Figure 1: Normalized execution time of short-running kernels with different persistency models on GPGPU-Sim

B. Long-Running Kernels with Short-Running CTAs

The performance results of the long-running kernels with short-running CTAs on GPGPU-Sim are shown in the following figures. Figure 2 shows the simulation results of persistency models, and Figure 3 shows the simulation results of durable transaction models.

From figure 2, we can see that CTA-level epoch persistency models can have lower performance overhead than strict persistency models. The overhead of the CTA-level epoch persistency model over the nvm baseline is 2.5% on average for EP_C_wt, 7.4% on average for EP_C_clwb and 11.5% on average for EP_C_l2wb.

From figure 3, we can see for the idempotent kernels, the idempotency analysis can reduce the performance overhead of undo logging.

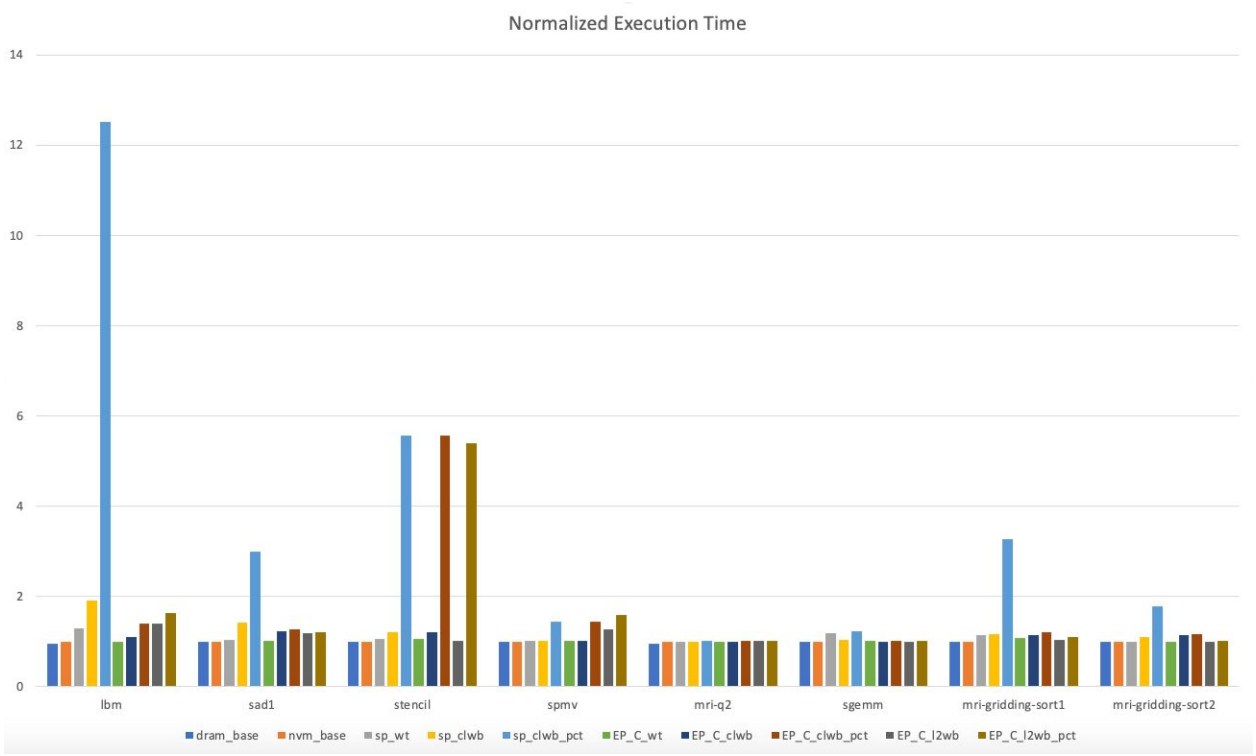

Figure 2: Normalized execution time of long-running kernels with short-running CTAs using different persistency models on GPGPU-Sim

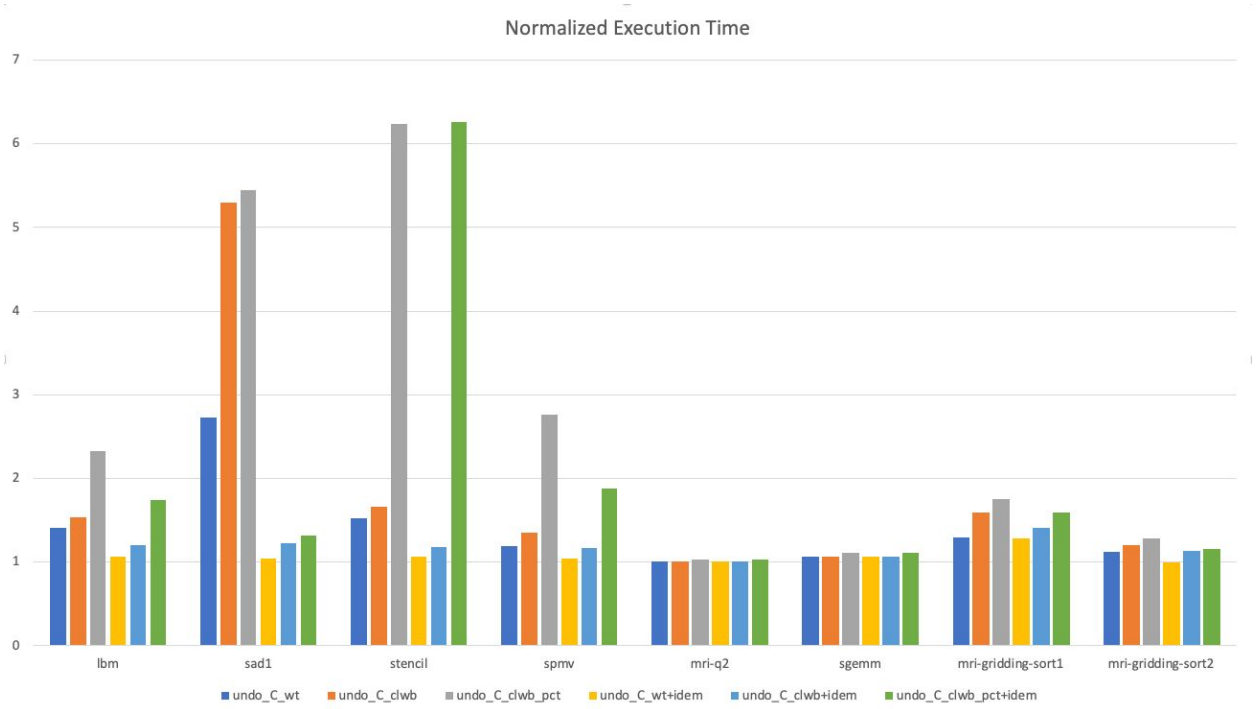

Figure 3: Normalized execution time of long-running kernels with short-running CTAs using different durable transaction models on GPGPU-Sim

C. Long-Running Kernels with Long-Running CTAs

The performance results of the long-running kernels with long-running CTAs on GPGPU-Sim are shown in Figure 4.

From figure 4, we can see that loop-level epoch persistency models have lower performance overhead than strict persistency models. The overhead of the loop-level epoch persistency model over the nvm baseline is 22.6% for EP_L_wt, 12.0% for EP_L_clwb and 12.7% for EP_L_clwb_pct. And since the bfs kernel is not idempotent, the performance overhead of using undo-logging with idempotency analysis is not reduced compared with using undo-logging without idempotency analysis.

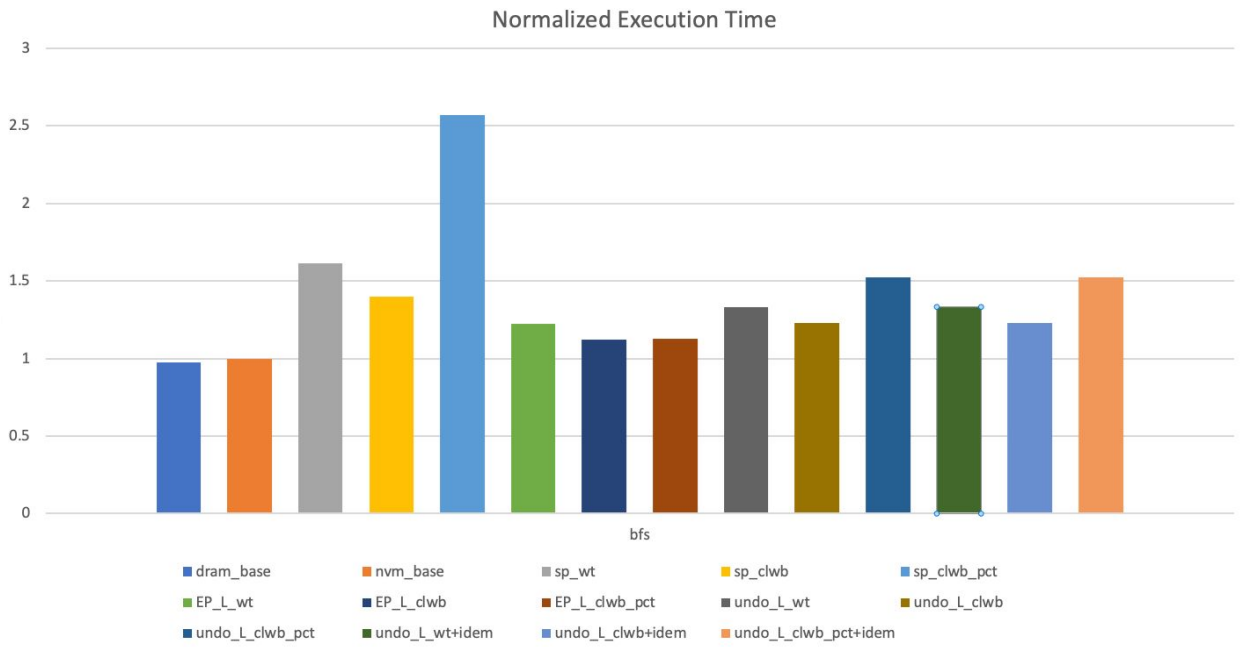

Figure 4: Normalized execution time of long-running kernels with long-running CTAs using different persistency models on GPGPU-Sim

D. Recommended Models

The recommended persistency model and durable transaction model for each kernel are listed in table Ⅴ. The average performance overhead of supporting the memory persistency models is 6.1%, and the average performance overhead of supporting the durable transaction models is 28.0%.

Table Ⅴ: Recommended persistency model and durable transaction model and their performance overhead.

Benchmark	PM Model	PM Overhead	DT Model	DT Overhead

bfs_v1	SP+clwb	15.2%	undo_K	58.8%
bfs	EP_L+clwb	12.0%	undo_L+clwb	22.7%
histo1	SP+clwb	0.5%	undo_K	63.1%
histo2	EP_K	26.9%	undo_K	26.9%
histo4	SP+wt	10.2%	undo_K	24.8%
mri-gridding-scan1	SP+clwb	4.9%	undo_K	42.2%
mri-gridding-scan2	SP+clwb	3.3%	undo_K	70.4%
mri-gridding-sort1	EP_C+l2wb	2.9%	undo_C+wt+idem	28.0%
mri-gridding-sort2	EP_C+l2wb	-0.8%	undo_C+wt+idem	-0.2%
mri-q1	SP+wt	-0.1%	undo_K	90.0%
mri-q2	EP_C+clwb	0.1%	undo_C+clwb	0.3%
sad2	EP_K	23.0%	undo_K	23.0%
sad1	EP_C+wt	1.9%	undo_C+wt+idem	3.7%
lbm	EP_C+wt	0.3%	undo_C+wt+idem	6.0%
stencil	EP_C+l2wb	2.5%	undo_C+wt+idem	6.2%
spmv	EP_C+wt	0.7%	undo_C+wt+idem	4.3%
sgemm	EP_C+l2wb	0.1%	undo_C+wt	6.3%
average		6.1%		28.0%

Experimental Results for Mega-KV Benchmark

For each kernel of the Mega-KV benchmark, we show the normalized execution time to the nvm baseline. And the performance results on GPGPU-Sim are shown in Figure 5.

kernel-level epoch persistency models are used for the kernels of Mega-KV since the kernels are classified into short-running kernels. The overhead of the kernel-level epoch persistency model over the nvm baseline is 3.4% and 3.5% on average for EP_K(epoch model with durable WPQs) and EP_K_pct (epoch model with volatile WPQs). Therefore, it is suitable to use the kernel-level epoch persistency model for Mega-KV benchmark since the overheads are small.

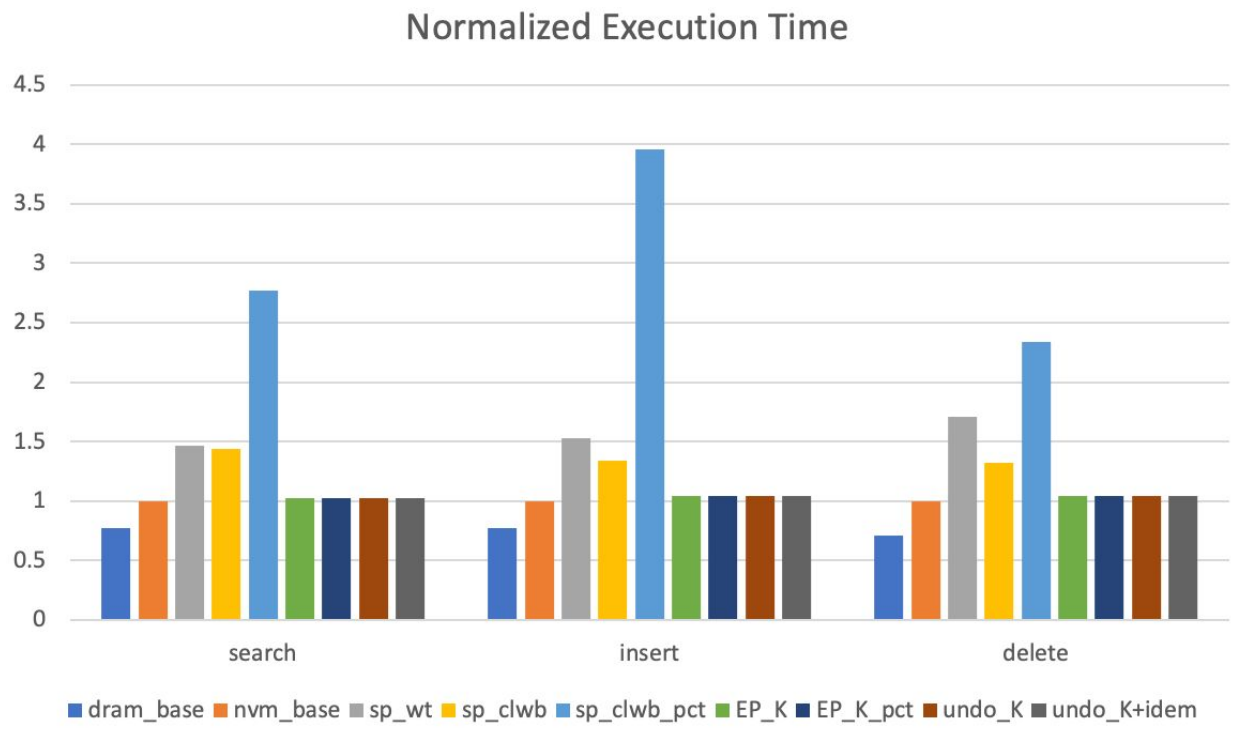

Figure 5: Normalized execution time of short-running kernels with different persistency models on GPGPU-Sim

Experimental Results for ProgPOW Benchmark

For each kernel of the ProgPOW benchmark, we show the normalized execution time to the nvm baseline. And the performance results on GPGPU-Sim are shown in Figure 6.

CTA-level epoch persistency models are used for the kernels of ProgPOW since the kernels are classified into long-running kernels with short-running CTAs. However, from the performance results in figure 6, we can see that the overhead of using strict persistency models is also very small, and the overhead of the strict persistency model over the nvm baseline is -0.1%, 0.1% and 0.4% on average for SP_wt (strict persistency model using store.wt instructions), SP_clwb (strict persistency model using clwb instructions) and SP_clwb_pct (strict persistency model using clwb instructions with volatile WPQs). Therefore, using strict persistency models is also suitable for ProgPOW benchmark since the overheads are very small.

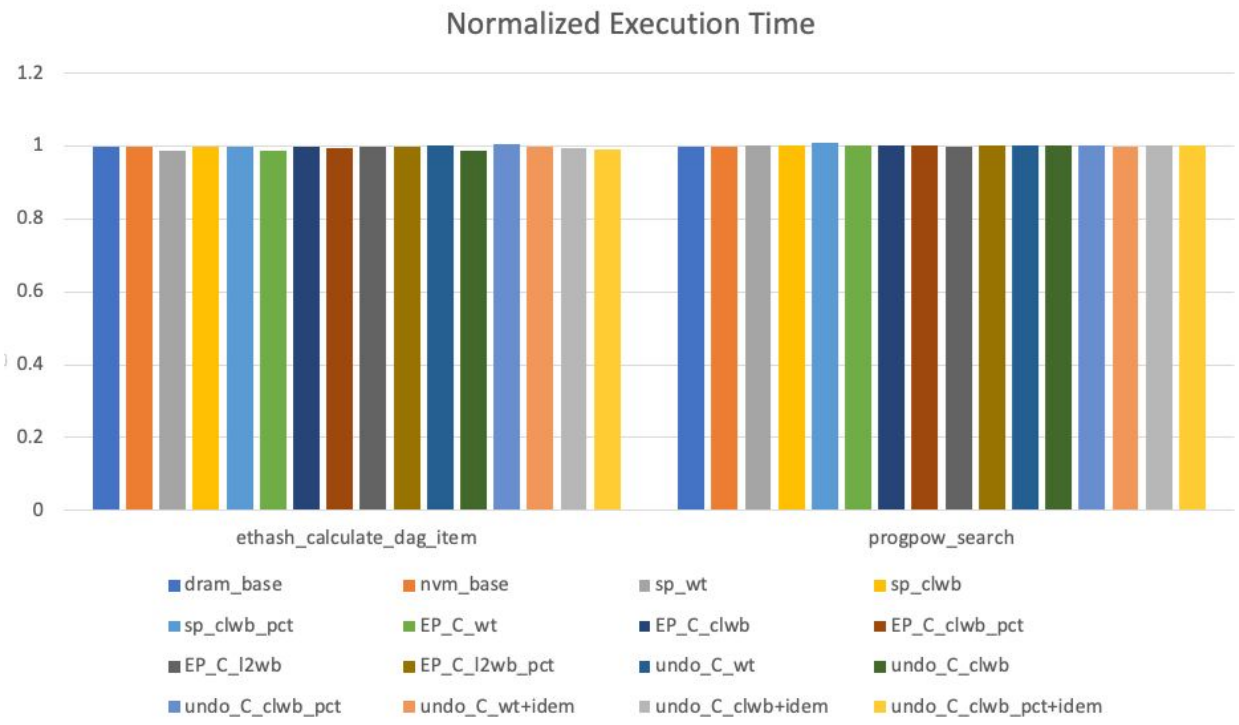

Figure 6: Normalized execution time of long-running kernels with short-running CTAs using different persistency models on GPGPU-Sim

Conclusions

From the results of the experiments, we can see:

(1). For the short-running kernels, using kernel-level epoch persistency models do not always have lower performance overhead than strict persistency models. This is because in the above experiments, when the kernel-level epoch persistency models are used for the short-running kernels, one epoch only includes one kernel. And when the kernel has less execution time, the overhead of using L2WB instruction to persist data will be higher. Therefore, changing the scope of an epoch to make multiple kernels included in one epoch should make kernel-level epoch persistency models more suitable for the kernels with less execution time. And in paper [1], a scope option is used to determine how many kernel to be included in an epoch.

(2). For the short-running kernels, durable transaction model undo_K (undo logging combined with the kernel-level epoch persistency model) and undo_K+idem (undo logging combined with the kernel-level epoch persistency model and the idempotency analysis) are used in the experiments. And undo logging combined with strict persistency models are not implemented in the experiments. However, using undo logging combined with strict persistency models probably will have lower performance overhead than using undo logging combined with kernel-level epoch persistency models. This is because as shown in the experiments of short-running kernels using persistency models without undo logging, some kernels which have

less execution time have shown lower performance overheads of using the strict persistency model than using the kernel-level epoch persistency model.

(3). For long-running kernels with short-running kernels, using CTA-level epoch persistency models have lower performance overhead than strict persistency models;

(4). For long-running kernels with long-running kernels, using loop-level epoch persistency models have lower performance overhead than strict persistency models;

(5). Idempotency analysis can reduce the overhead of undo logging if the kernel is idempotent.

References

[1]. Z. Lin, M. Alshboul, Y. Solihin and H. Zhou, "Exploring memory persistency models for gpus", 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 311-323, 2019.
[2]. Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, Timothy G Rogers. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In proceedings of the 47th IEEE/ACM International Symposium on Computer Architecture (ISCA), May 29 - June 3, 2020.
[3]. A. Bakhoda et al. Analyzing cuda workloads using a detailed gpu simulator. ISPASS-2009.
[4]. J. A. Stratton et al. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. UIUC, Tech. Rep. IMPACT-12-01, March 2012
[5]. K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, and X. Zhang, “Mega-kv: A case for gpus to maximize the throughput of in-memory key-value stores,” Proc. VLDB Endow., vol. 8, no. 11, pp. 1226–1237, Jul. 2015. [Online]. Available: https://doi.org/10.14778/2809974.2809984
[6]. ProgPOW. [Online]. Available: https://github.com/ifdefelse/ProgPOW