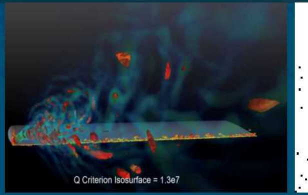
SAND2019-11322C

The Structural Simulation Toolkit and GPGPU-Sim



PRESENTED BY

Clay Hughes, Sandia National Laboratories

SST Development Team, Sandia National Laboratories

Architecture Accelerator Lab, Purdue University

09/22/2019

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2019-XXXX

Welcome!

Part 1: Introduction to SST *1:00 – 2:30*

- SST Overview
  - *Demos: Running a simple simulation; Enabling statistics; Running in parallel*
- SST Element Libraries: A tour

Part 2: GPGPU-Sim 2:30-3:30

- GPGPU-Sim Overview
- New Features

Break *3:30 – 4:00*

Part 3: The SST/GPGPU-Sim Integration ***4:00 - 5:00***

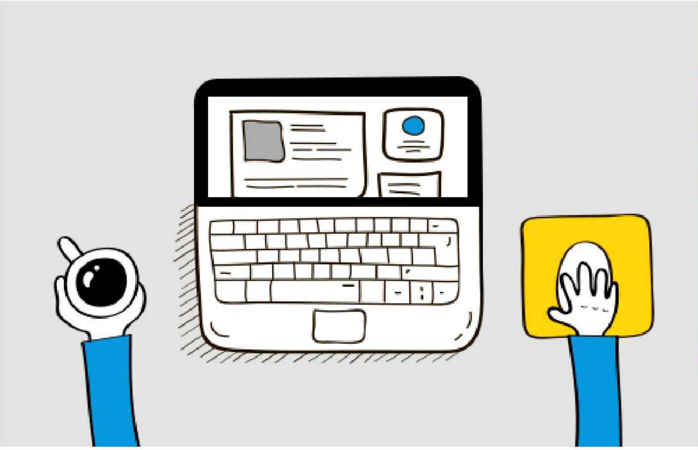
SST GPGPU Modeling Capability

Long history of GPGPU simulators, mostly from academia

- GPGPUSim [Bakhoda, 2009]
- Gem5+GPU [Power, 2014]
- MacSim/Ocelot [Hyesoon, 2012], [Kerr, 2009]
- Multi2Sim v5.0 [Gong, 2017]
- NVArchSim

After a literature review and phone calls with the developers of GPGPUSim and Multi2Sim, the SST team chose GPGPUSim as the integration target

- More amendable to SST integration
  - Software designed around well-defined objects
- More tractable for SST workflows



GPGPUSim Integration

GPGPUSim

- Runs *host* code directly
- Runs *device* code on a functional simulator
- Interfaces with programs through a custom library, replacing native CUDA calls
- Simulation split into functional and timing

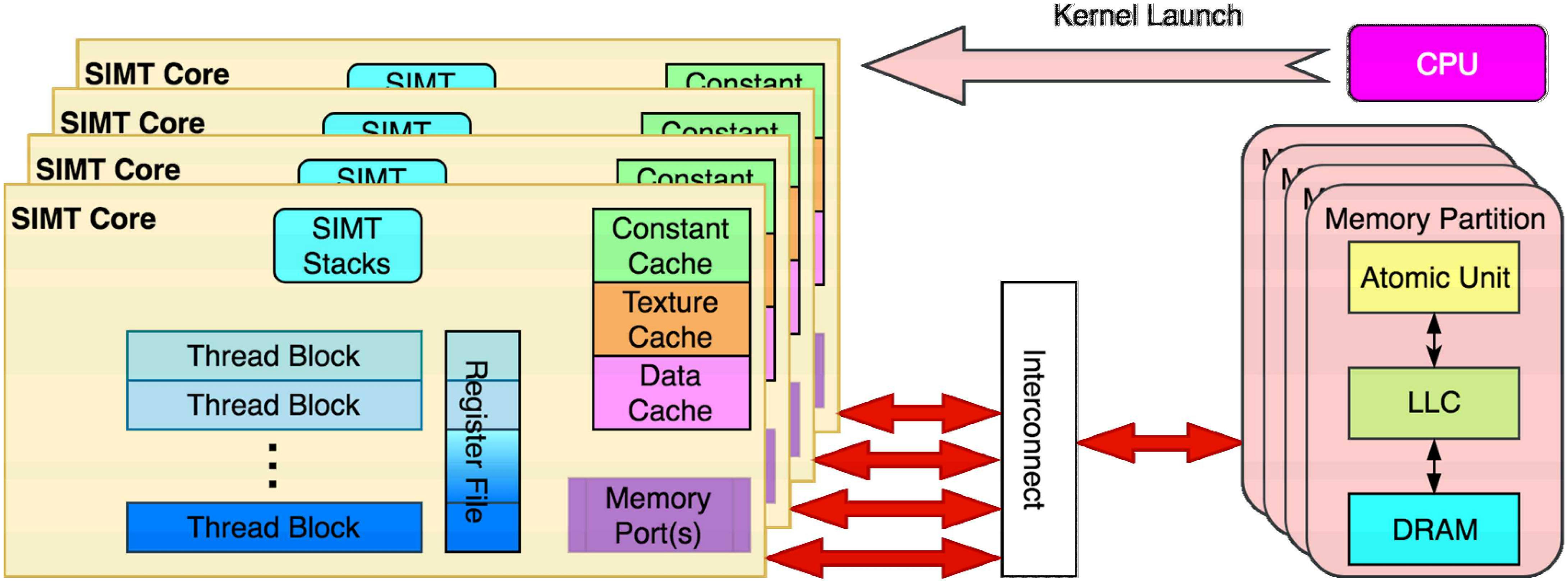
Functional

- CTA
  - Pool of threads with access to shared memory
- Thread
  - PC + registers + allocated memory
- Instruction
  - 1:1 mapping with opcode and implementation function; simulated at issue

Timing

- SIMT Core
  - Roughly equivalent to Fermi's SM
- Cache
  - L1 → Per-core, write-through/no-allocate
  - L2 → Device, write-back/no-allocate
- Interconnect
  - Intersim (Booksim)
- Backing store
  - GDDR3/GDDR5

GPGPUSim Integration



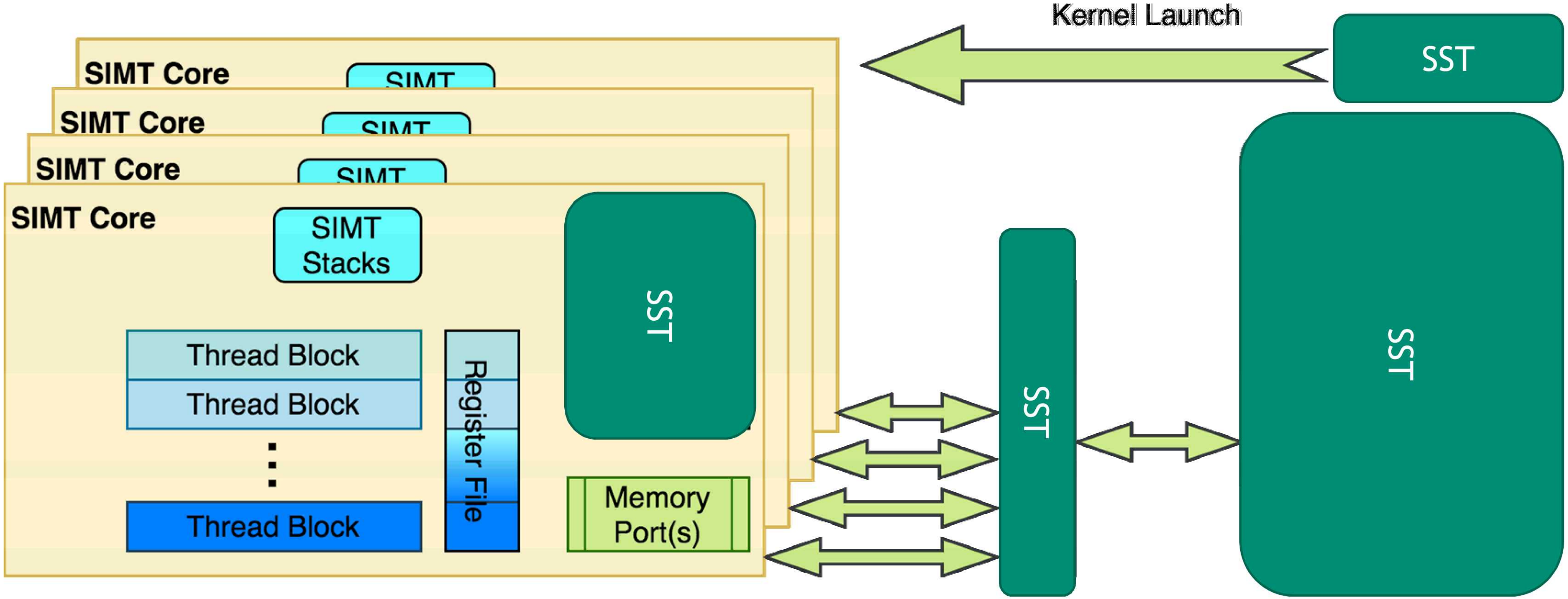
Functional

- SIMT units track default functional model from GPGPUSim

Timing

- SIMT units maintain execution timing; Booksim is used for the interconnect; and simple timing models are used for the memory partition

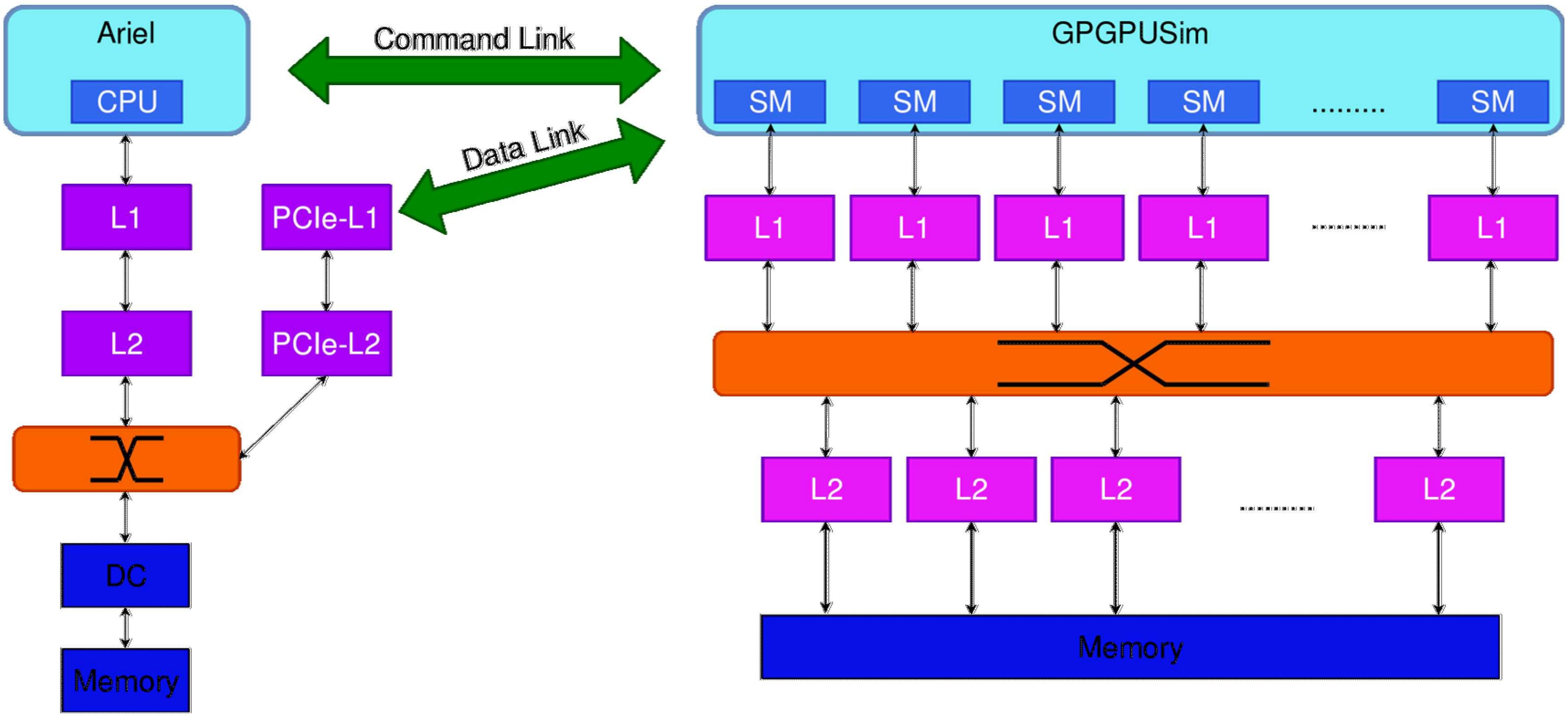
GPGPUSim Integration



Plan to keep the SIMT units and replace everything else

- CPU → SST execution component (Ariel, Juno, etc.)
- Interconnect → SST networking component (Shogun, Merlin, Kingsley, etc.)
- Memory Partition & Caches→ SST memHierarchy component for caches and various other backends for the backing store (DRAMSim, SimpleMem, Cramsim, etc.)

GPGPUSim Integration



Primary SST Components: Ariel

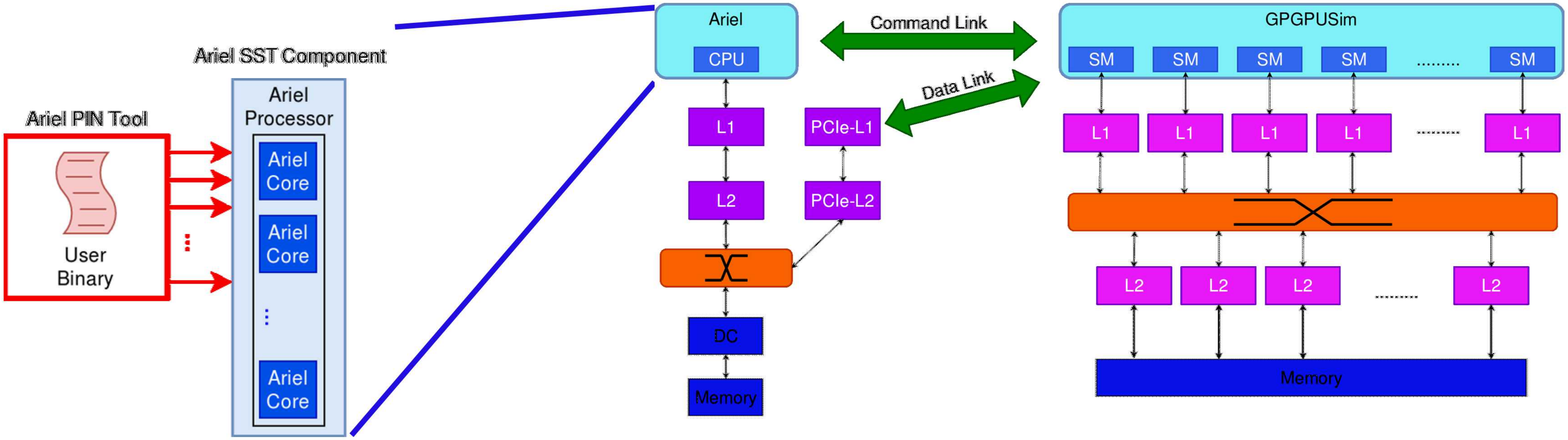
Frontend based on Intel PIN, which passes information to a SST component

- Supports OpenMP and MPI binaries

Faster than cycle-approximate models like Gem5 but slower than trace-based

Reasonable model of thread interactions

- Non-deterministic



Primary SST Components: memHierarchy

Collection of interoperable memory system elements

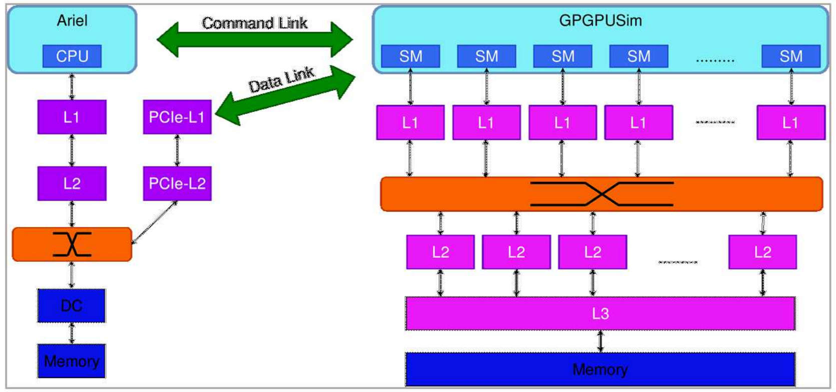
- Caches and directories
- Memory models (DDR, HBM, HMC, NVRAM, etc.)
- Memory controllers & network interfaces for memory (MemNICs)

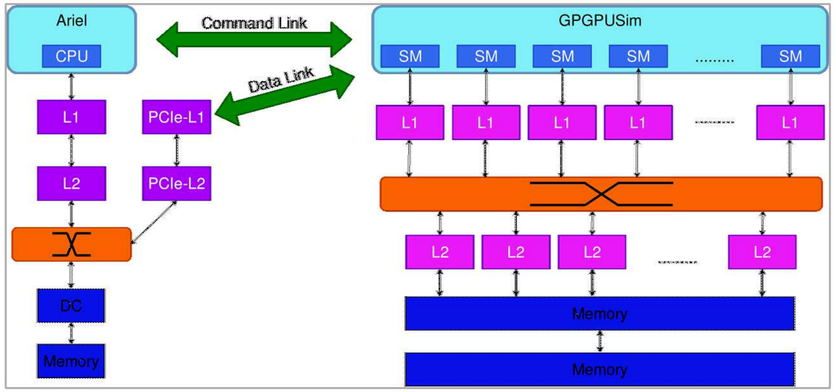
Inter- and intra-socket coherence

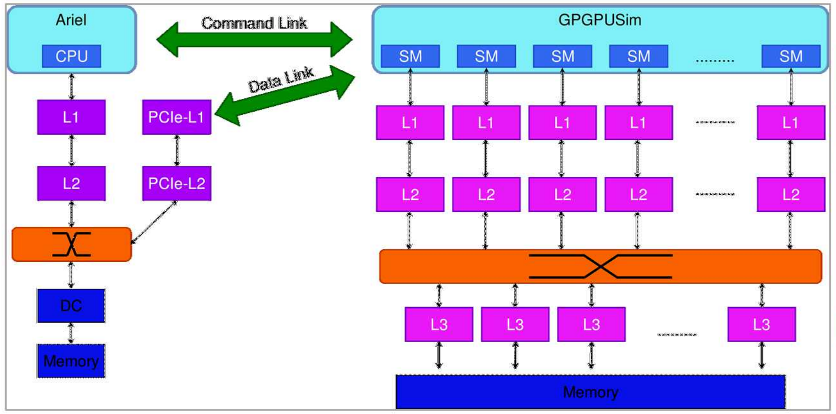
Correlated with modern memory hierarchies

- HBM2/3 Evaluation on Many-core CPU for ECP
- Messier: A Detailed NVM-Based DIMM Model for the SST Simulation Framework

Will allow us to model any sort of memory arrangement that we can imagine!





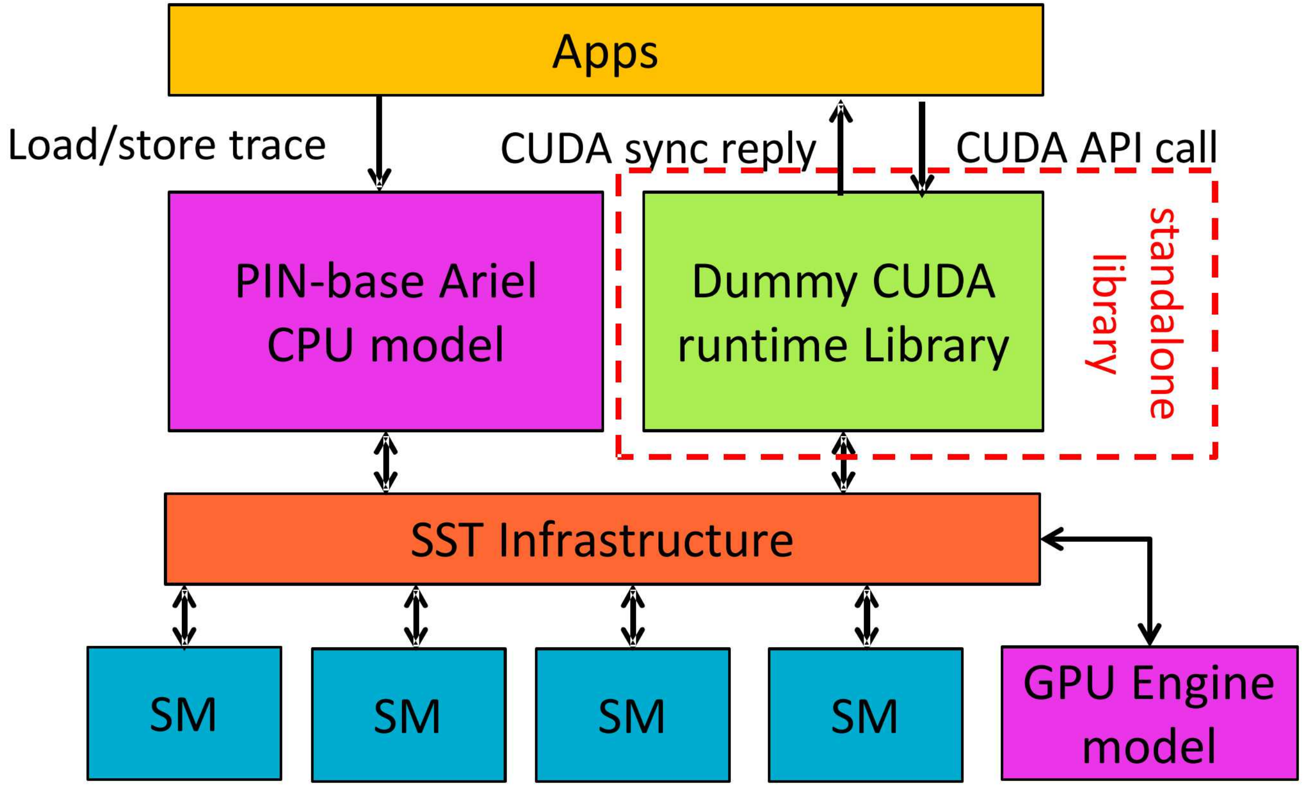


CUDA API interception model

Standalone dummy CUDA lib which is independent of CPU model

Applications link dynamically with lib (shared object)

Dummy lib interacts with GPU engine for kernel launch and CUDA memory copies



API Calls Forwarded to GPU Model	
__cudaRegisterFatBinary	cudaFree
__cudaRegisterFunction	cudaLaunch
cudaMalloc	cudaGetLastError
cudaMemcpy	cudaRegisterVar
cudaConfigureCall	cudaSetupArgument
cudaOccupancyMaxActiveBlocksPerMultiprocessorWithFlags	

Using The GPU Component

Using the Component

Currently hosted in a separate repository within the SST project

To build SST elements with the GPU component (assumes core is built):

1. git clone https://github.com/sstsimulator/sst-elements.git
2. git clone https://github.com/sstsimulator/sst-gpgpusim.git $SST_ELEMENTS_ROOT/src/sst/elements/Gpgpusim
3. . src/sst/elements/Gpgpusim/sst-gpgpusim/setup_environment
4. cd src/sst/elements/Gpgpusim/sst-gpgpusim
5. make
6. cp --preserve=links $GPGPUSIM_ROOT/lib/$GPGPUSIM_CONFIG/libcudart_mod.so $SST_ELEMENTS_ROOT/src/sst/elements/Gpgpusim/
7. cd $SST_ELEMENTS_HOME
8. ./autogen
9. ./configure --prefix=$SST_ELEMENTS_HOME --with-sst-core=$SST_CORE_HOME --with-pin=$PIN_HOME --with-cuda=$CUDA_ROOT
10. make

Applications must be built using the shared cuda runtime library: --cudart=shared

If an application requires cuDNN or cuBLAS, it must be linked against the static Cuda library: -cublas_static, -lcudnn_static

Using the Component

Running a simulation requires the user to specify both a SST model and a GPGPU model

- SST model contains wiring instructions and component definitions
  - Processor model, memory hierarchy, interconnect, etc.
- GPGPU model contains SIMT core definition
  - PTX generation, number of registers, number of warps, number of functional units, etc.

SST Component

```
comp_gpu0 = sst.Component("gpu0", "Gpgpusim.Gpgpusim")
comp_gpu0.addParams({
    "clock" : "1200MHz",
    "gpu_cores" : "84",
    "verbose" : "0"
})
```

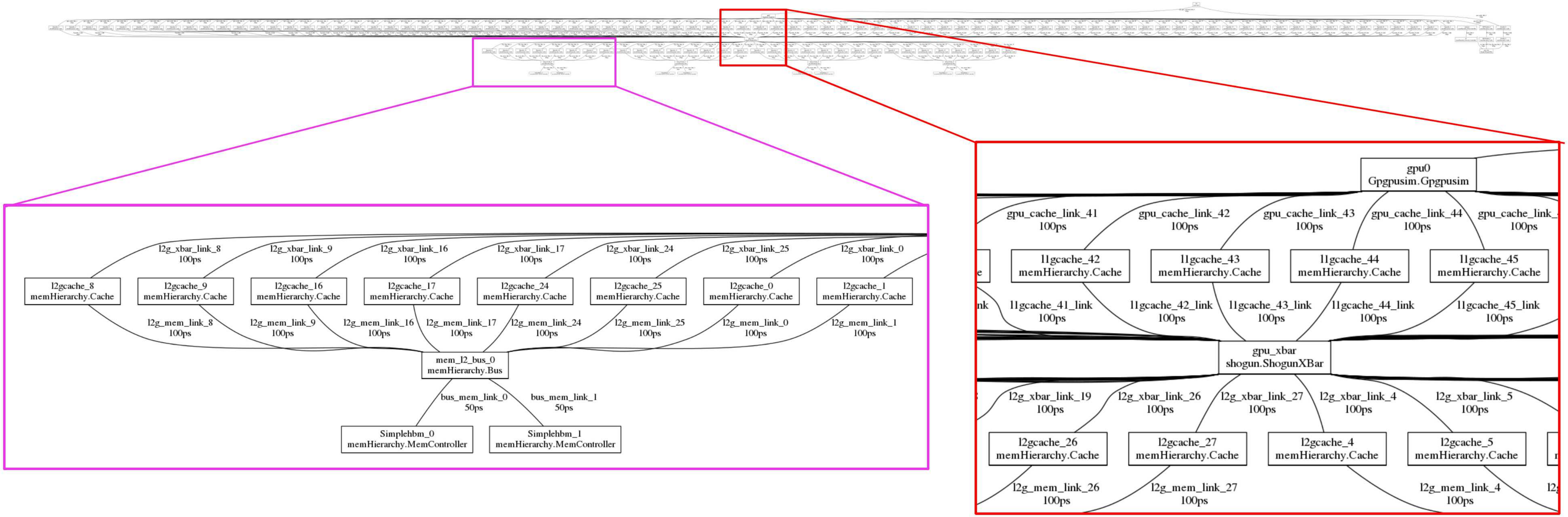
GPGPU Model

```
-SST_mode 1
-gpgpu_ptx_force_max_capability 70
-gpgpu_clock_domains 1200.0:1200.0:1200.0:850.0
-gpgpu_shader_registers 65536
-gpgpu_occupancy_sm_number 70

-gpgpu_pipeline_widths 4,4,4,4,4,4,4,4,4,4,8,4,4
-gpgpu_num_sp_units 4
-gpgpu_num_sfu_units 4
-gpgpu_num_dp_units 4
-gpgpu_num_int_units 4
-gpgpu_tensor_core_avail 1
-gpgpu_num_tensor_core_units 4
```

Using the Component – vectorAdd

sst --model-option="-c ariel-gpu-v100.cfg -s abs_kviTA.csv -x ./vectorAdd-163840" cuda-test.py



Using the Component – vectorAdd

sst --model-option="-c ariel-gpu-v100.cfg -s abs_kviTA.csv -x ./vectorAdd-163840" cuda-test.py

Statistics file for this model contains 19861 entries:

ComponentName, StatisticName, StatisticSubId, StatisticType, SimTime, Rank, Sum, SumSQ, Count, Min, Max
A0, read_requests, 0, Accumulator, 2423910320, 0, 10240, 10240, 10240, 1, 1
A0, write_requests, 0, Accumulator, 2423910320, 0, 20480, 20480, 20480, 1, 1
A0, read_request_sizes, 0, Accumulator, 2423910320, 0, 655360, 41943040, 10240, 64, 64
A0, write_request_sizes, 0, Accumulator, 2423910320, 0, 1310720, 83886080, 20480, 64, 64
A0, instruction_count, 0, Accumulator, 2423910320, 0, 0, 0, 0, 0, 0
A0, cycles, 0, Accumulator, 2423910320, 0, 6446570, 6446570, 6446570, 1, 1
A0, active_cycles, 0, Accumulator, 2423910320, 0, 226, 226, 226, 1, 1

…

…

Performance Analysis

Mean Error

	P9/V100	SST-GPGPU	Error
vectorAdd	5271	5751	9.093682
lud	494519	605685	22.47961
lulesh	12454750	11896477	4.48241

Complex model interactions make it difficult to pinpoint where models are lacking in detail or are incorrect

Scale of simulation – number of components and links present a challenge

- Full debug produces multi-TiB trace
- No errors, so have to isolate anomalies in tens of thousands of statistics

Functional Testing – Kokkos Kernels

Kokkos-Kernels built with:

KOKKOSKERNELS_SCALARS=double
KOKKOSKERNELS_LAYOUTS=LayoutLeft
KOKKOSKERNELS_ORDINALS=int
KOKKOSKERNELS_OFFSETS=int
KOKKOS_DEVICES=Cuda,Serial
KOKKOS_ARCHS=BDW,Pascal60

Tool	Version
gcc	4.8.5
CUDA SDK	8.0.61
Kokkos	2.8.00
Kokkos Kernels	2.8.00
sst-core	devel-d8042ec
sst-elements	devel-b3ba937
Pin	2.14.71313

Compiling for Broadwell and sm_60 for validation testing on Doom

Functional Testing – Kokkos Kernels Unit Test Results

1	abs_double
2	abs_mv_double
3	asum_double
4	axpby_double
5	axpby_mv_double
6	axpy_double
7	axpy_mv_double
8	dot_double
9	dot_mv_double
10	mult_double
11	mult_mv_double
12	nrm1_double
13	nrm1_mv_double
14	nrm2_double
15	nrm2_mv_double
16	nrm2_squared_double
17	nrm2_squared_mv_double
18	nrminf_double
19	nrminf_mv_double
20	reciprocal_double
21	reciprocal_mv_double
22	scal_double
23	scal_mv_double
24	sum_double
25	sum_mv_double
26	update_double
27	update_mv_double
28	gemv_double
29	gemm_double
30	sparse_spgemm_double_int_int_TestExecSpace

31	sparse_spadd_double_int_int_TestExecSpace
32	sparse_gauss_seidel_double_int_int_TestExecSpace
33	sparse_block_gauss_seidel_double_int_int_TestExecSpace
34	sparse_crsmatrix_double_int_int_TestExecSpace
35	sparse_blkcrsmatrix_double_int_int_TestExecSpace
36	sparse_replaceSumIntoLonger_double_int_int_TestExecSpace
37	sparse_replaceSumInto_double_int_int_TestExecSpace
38	graph_graph_color_double_int_int_TestExecSpace
39	graph_graph_color_deterministic_double_int_int_TestExecSpace
40	graph_graph_color_d2_double_int_int_TestExecSpace
41	common_ArithTraits
42	common_set_bit_count
43	common_ffs
44	batched_scalar_serial_set_double_double
45	batched_scalar_serial_scale_double_double
46	batched_scalar_serial_gemm_nt_nt_double_double
47	batched_scalar_serial_gemm_t_nt_double_double
48	batched_scalar_serial_gemm_nt_t_double_double
49	batched_scalar_serial_gemm_t_t_double_double
50	batched_scalar_serial_trsm_l_l_nt_u_double_double
51	batched_scalar_serial_trsm_l_l_nt_n_double_double
52	batched_scalar_serial_trsm_l_u_nt_u_double_double
53	batched_scalar_serial_trsm_l_u_nt_n_double_double
54	batched_scalar_serial_trsm_r_u_nt_u_double_double
55	batched_scalar_serial_trsm_r_u_nt_n_double_double
56	batched_scalar_serial_trsm_l_l_t_u_double_double
57	batched_scalar_serial_trsm_l_l_t_n_double_double
58	batched_scalar_serial_trsm_l_u_t_u_double_double
59	batched_scalar_serial_trsm_l_u_t_n_double_double
60	batched_scalar_serial_gemv_nt_double_double

61	batched_scalar_serial_gemv_t_double_double
62	batched_scalar_serial_trsv_l_nt_u_double_double
63	batched_scalar_serial_trsv_l_nt_n_double_double
64	batched_scalar_serial_trsv_u_nt_u_double_double
65	batched_scalar_serial_trsv_u_nt_n_double_double
66	batched_scalar_team_set_double_double
67	batched_scalar_team_scale_double_double
68	batched_scalar_team_gemm_nt_nt_double_double
69	batched_scalar_team_gemm_t_nt_double_double
70	batched_scalar_team_gemm_nt_t_double_double
71	batched_scalar_team_gemm_t_t_double_double
72	batched_scalar_team_trsm_l_l_nt_u_double_double
73	batched_scalar_team_trsm_l_l_nt_n_double_double
74	batched_scalar_team_trsm_l_u_nt_u_double_double
75	batched_scalar_team_trsm_l_u_nt_n_double_double
76	batched_scalar_team_trsm_r_u_nt_u_double_double
77	batched_scalar_team_trsm_r_u_nt_n_double_double
78	batched_scalar_team_trsm_l_l_t_u_double_double
79	batched_scalar_team_trsm_l_l_t_n_double_double
80	batched_scalar_team_trsm_l_u_t_u_double_double
81	batched_scalar_team_trsm_l_u_t_n_double_double
82	batched_scalar_team_gemv_nt_double_double
83	batched_scalar_team_gemv_t_double_double
84	batched_scalar_serial_lu_double
85	batched_scalar_serial_inverselu_double
86	batched_scalar_serial_solvelu_double
87	batched_scalar_team_lu_double
88	batched_scalar_team_inverselu_double
89	batched_scalar_team_solvelu_double

Currently passing 54/89 tests (60.67%)

Functional Testing – Kokkos Kernels Unit Test Results

1	abs_double	31	sparse_spadd_double_int_int_TestExecSpace	61	batched_scalar_serial_gemv_t_double_double
2	abs_mv_double	32	sparse_gauss_seidel_double_int_int_TestExecSpace	62	batched_scalar_serial_trsv_l_nt_u_double_double
3	asum_double	33	sparse_block_gauss_seidel_double_int_int_TestExecSpace	63	batched_scalar_serial_trsv_l_nt_n_double_double
4	axpby_double	34	sparse_crsmatrix_double_int_int_TestExecSpace	64	batched_scalar_serial_trsv_u_nt_u_double_double
5	axpby_mv_double	35	sparse_blkcrsmatrix_double_int_int_TestExecSpace	65	batched_scalar_serial_trsv_u_nt_n_double_double
6	axpy_double	36	sparse_replaceSumIntoLonger_double_int_int_TestExecSpace	66	batched_scalar_team_set_double_double
7	axpy_mv_double	37	sparse_replaceSumInto_double_int_int_TestExecSpace	67	batched_scalar_team_scale_double_double
8	dot_double	38	graph_graph_color_double_int_int_TestExecSpace	68	batched_scalar_team_gemm_nt_nt_double_double
9	dot_mv_double	39	graph_graph_color_deterministic_double_int_int_TestExecSpace	69	batched_scalar_team_gemm_t_nt_double_double
10	mult_double	40	graph_graph_color_d2_double_int_int_TestExecSpace	70	batched_scalar_team_gemm_nt_t_double_double
11	mult_mv_double	41	common_ArithTraits	71	batched_scalar_team_gemm_t_t_double_double
12	nrm1_double	42	common_set_bit_count	72	batched_scalar_team_trsm_l_l_nt_u_double_double
13	nrm1_mv_double	43	common_ffs	73	batched_scalar_team_trsm_l_l_nt_n_double_double
14	nrm2_double	44	batched_scalar_serial_set_double_double	74	batched_scalar_team_trsm_l_u_nt_u_double_double
15	nrm2_mv_double	45	batched_scalar_serial_scale_double_double	75	batched_scalar_team_trsm_l_u_nt_n_double_double
16	nrm2_squared_double	46	batched_scalar_serial_gemm_nt_nt_double_double	76	batched_scalar_team_trsm_r_u_nt_u_double_double
17	nrm2_squared_mv_double	47	batched_scalar_serial_gemm_t_nt_double_double	77	batched_scalar_team_trsm_r_u_nt_n_double_double
18	nrminf_double	48	batched_scalar_serial_gemm_nt_t_double_double	78	batched_scalar_team_trsm_l_l_t_u_double_double
19	nrminf_mv_double	49	batched_scalar_serial_gemm_t_t_double_double	79	batched_scalar_team_trsm_l_l_t_n_double_double
20	reciprocal_double	50	batched_scalar_serial_trsm_l_l_nt_u_double_double	80	batched_scalar_team_trsm_l_u_t_u_double_double
21	reciprocal_mv_double	51	batched_scalar_serial_trsm_l_l_nt_n_double_double	81	batched_scalar_team_trsm_l_u_t_n_double_double
22	scal_double	52	batched_scalar_serial_trsm_l_u_nt_u_double_double	82	batched_scalar_team_gemv_nt_double_double
23	scal_mv_double	53	batched_scalar_serial_trsm_l_u_nt_n_double_double	83	batched_scalar_team_gemv_t_double_double
24	sum_double	54	batched_scalar_serial_trsm_r_u_nt_u_double_double	84	batched_scalar_serial_lu_double
25	sum_mv_double	55	batched_scalar_serial_trsm_r_u_nt_n_double_double	85	batched_scalar_serial_inverselu_double
26	update_double	56	batched_scalar_serial_trsm_l_l_t_u_double_double	86	batched_scalar_serial_solvelu_double
27	update_mv_double	57	batched_scalar_serial_trsm_l_l_t_n_double_double	87	batched_scalar_team_lu_double
28	gemv_double	58	batched_scalar_serial_trsm_l_u_t_u_double_double	88	batched_scalar_team_inverselu_double
29	gemm_double	59	batched_scalar_serial_trsm_l_u_t_n_double_double	89	batched_scalar_team_solvelu_double
30	sparse_spgemm_double_int_int_TestExecSpace	60	batched_scalar_serial_gemv_nt_double_double		

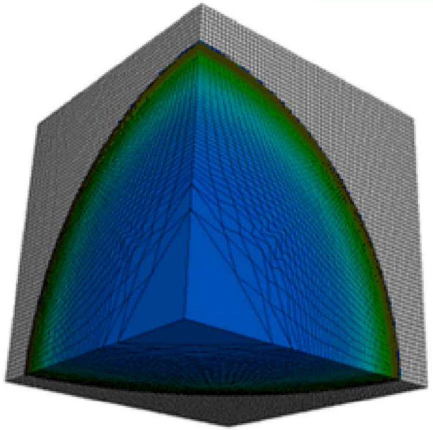
Tests in pink fail because the parser cannot locate a post-dominator in some kernels; this bug is also present in the public branch of GPGPUSim

Functional Testing – Kokkos Kernels Unit Test Results

1	abs_double	31	sparse_spadd_double_int_int_TestExecSpace	61	batched_scalar_serial_gemv_t_double_double
2	abs_mv_double	32	sparse_gauss_seidel_double_int_int_TestExecSpace	62	batched_scalar_serial_trsv_l_nt_u_double_double
3	asum_double	33	sparse_block_gauss_seidel_double_int_int_TestExecSpace	63	batched_scalar_serial_trsv_l_nt_n_double_double
4	axpby_double	34	sparse_crsmatrix_double_int_int_TestExecSpace	64	batched_scalar_serial_trsv_u_nt_u_double_double
5	axpby_mv_double	35	sparse_blkcrsmatrix_double_int_int_TestExecSpace	65	batched_scalar_serial_trsv_u_nt_n_double_double
6	axpy_double	36	sparse_replaceSumIntoLonger_double_int_int_TestExecSpace	66	batched_scalar_team_set_double_double
7	axpy_mv_double	37	sparse_replaceSumInto_double_int_int_TestExecSpace	67	batched_scalar_team_scale_double_double
8	dot_double	38	graph_graph_color_double_int_int_TestExecSpace	68	batched_scalar_team_gemm_nt_nt_double_double
9	dot_mv_double	39	graph_graph_color_deterministic_double_int_int_TestExecSpace	69	batched_scalar_team_gemm_t_nt_double_double
10	mult_double	40	graph_graph_color_d2_double_int_int_TestExecSpace	70	batched_scalar_team_gemm_nt_t_double_double
11	mult_mv_double	41	common_ArithTraits	71	batched_scalar_team_gemm_t_t_double_double
12	nrm1_double	42	common_set_bit_count	72	batched_scalar_team_trsm_l_l_nt_u_double_double
13	nrm1_mv_double	43	common_ffs	73	batched_scalar_team_trsm_l_l_nt_n_double_double
14	nrm2_double	44	batched_scalar_serial_set_double_double	74	batched_scalar_team_trsm_l_u_nt_u_double_double
15	nrm2_mv_double	45	batched_scalar_serial_scale_double_double	75	batched_scalar_team_trsm_l_u_nt_n_double_double
16	nrm2_squared_double	46	batched_scalar_serial_gemm_nt_nt_double_double	76	batched_scalar_team_trsm_r_u_nt_u_double_double
17	nrm2_squared_mv_double	47	batched_scalar_serial_gemm_t_nt_double_double	77	batched_scalar_team_trsm_r_u_nt_n_double_double
18	nrminf_double	48	batched_scalar_serial_gemm_nt_t_double_double	78	batched_scalar_team_trsm_l_l_t_u_double_double
19	nrminf_mv_double	49	batched_scalar_serial_gemm_t_t_double_double	79	batched_scalar_team_trsm_l_l_t_n_double_double
20	reciprocal_double	50	batched_scalar_serial_trsm_l_l_nt_u_double_double	80	batched_scalar_team_trsm_l_u_t_u_double_double
21	reciprocal_mv_double	51	batched_scalar_serial_trsm_l_l_nt_n_double_double	81	batched_scalar_team_trsm_l_u_t_n_double_double
22	scal_double	52	batched_scalar_serial_trsm_l_u_nt_u_double_double	82	batched_scalar_team_gemv_nt_double_double
23	scal_mv_double	53	batched_scalar_serial_trsm_l_u_nt_n_double_double	83	batched_scalar_team_gemv_t_double_double
24	sum_double	54	batched_scalar_serial_trsm_r_u_nt_u_double_double	84	batched_scalar_serial_lu_double
25	sum_mv_double	55	batched_scalar_serial_trsm_r_u_nt_n_double_double	85	batched_scalar_serial_inverselu_double
26	update_double	56	batched_scalar_serial_trsm_l_l_t_u_double_double	86	batched_scalar_serial_solvelu_double
27	update_mv_double	57	batched_scalar_serial_trsm_l_l_t_n_double_double	87	batched_scalar_team_lu_double
28	gemv_double	58	batched_scalar_serial_trsm_l_u_t_u_double_double	88	batched_scalar_team_inverselu_double
29	gemm_double	59	batched_scalar_serial_trsm_l_u_t_n_double_double	89	batched_scalar_team_solvelu_double
30	sparse_spgemm_double_int_int_TestExecSpace	60	batched_scalar_serial_gemv_nt_double_double		

Tests in purple fail because of a bug(s) in the functional model that neither the SST nor the GPGPUSim team has been able to isolate

Performance Analysis – Lulesh



One of the most widely used DOE mini-applications

- Developed by Lawrence Livermore National Laboratory
- Represents challenging hydrodynamics algorithms over unstructured meshes
  - Common in many high-performance computing centers and are particularly prevalent within the NNSA laboratories
  - Routinely counted in the top ten application codes in terms of CPU hours utilized

Lulesh was compiled with:

Tool	Version
gcc	4.8.5
CUDA SDK	9.1.85
Lulesh	1.0
sst-core	devel-d8042ec
sst-elements	devel-b3ba937
Pin	2.14.71313

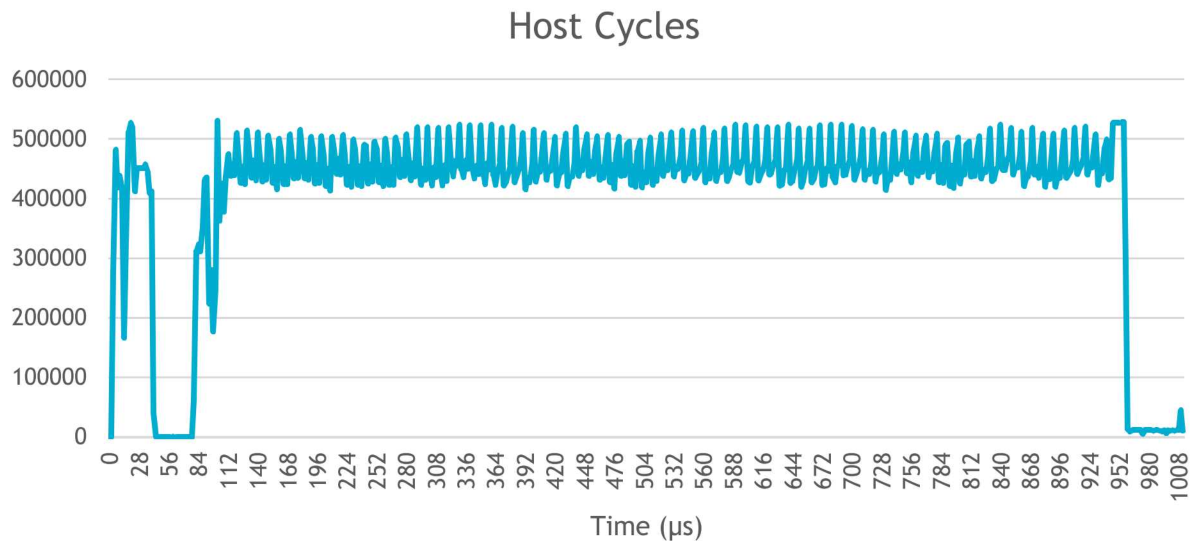
Lulesh was run with:

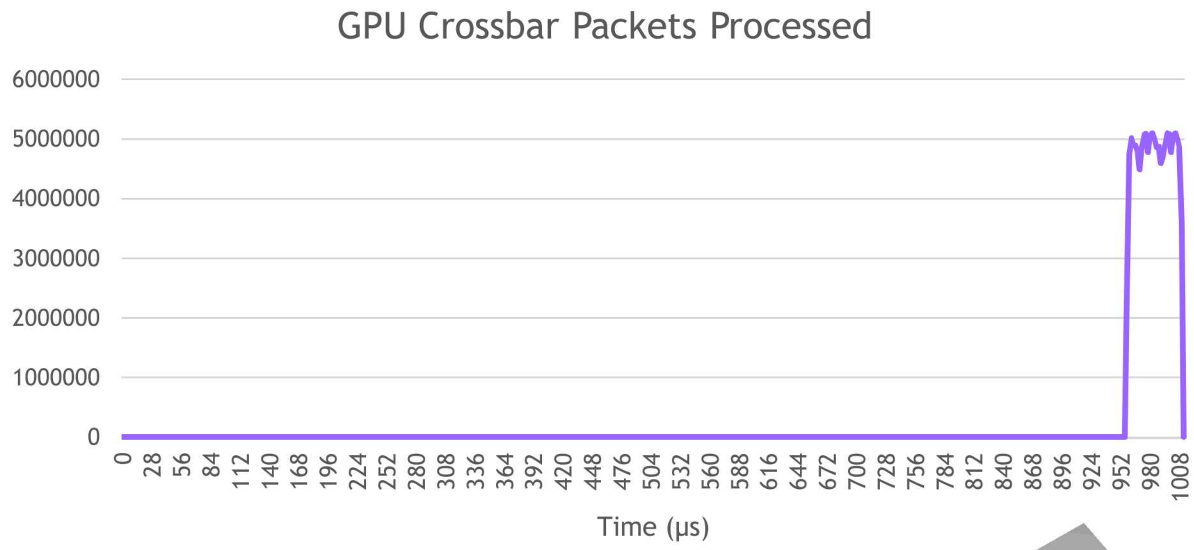
- s=22 (22x22x22 elements)
- Iterations=50

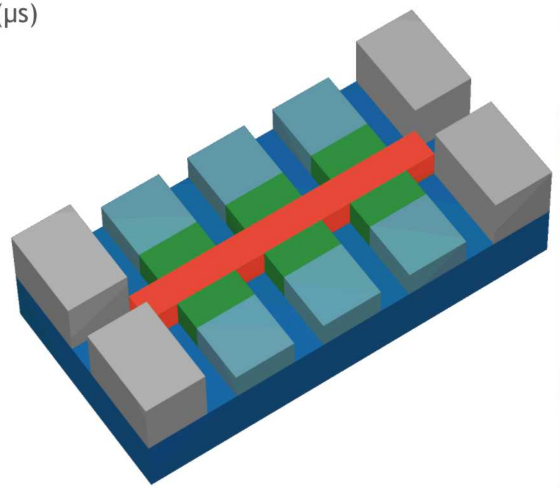
Performance Analysis – Lulesh

SST is capable of providing periodic statistic dumps for all of the currently loaded components

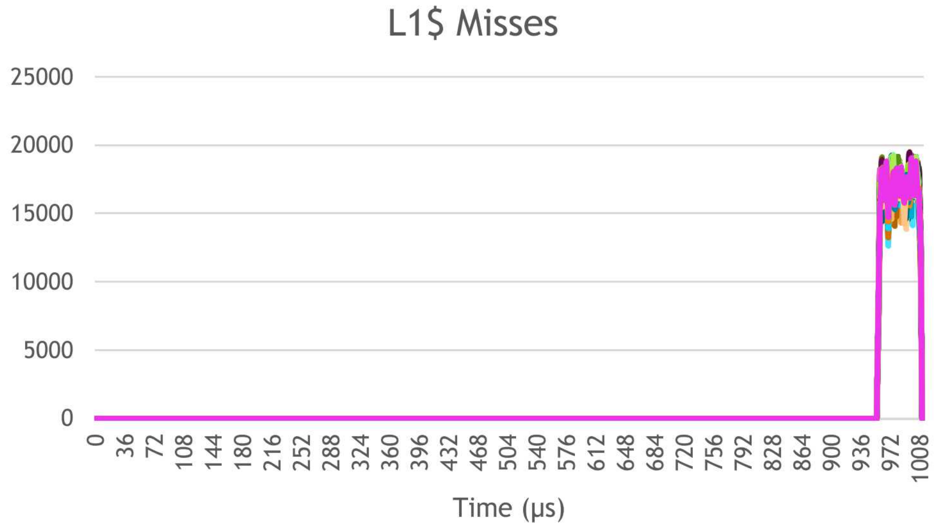
- Valuable for fine-grain performance analysis of applications

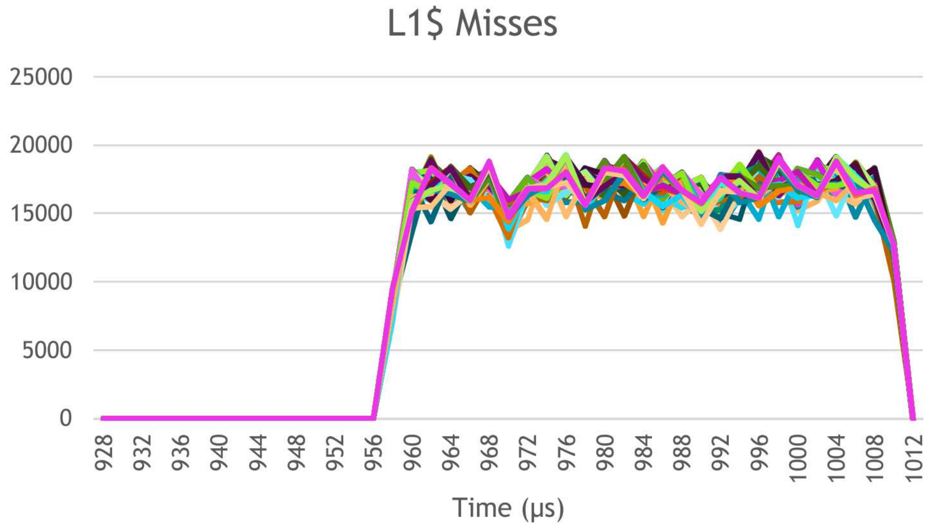


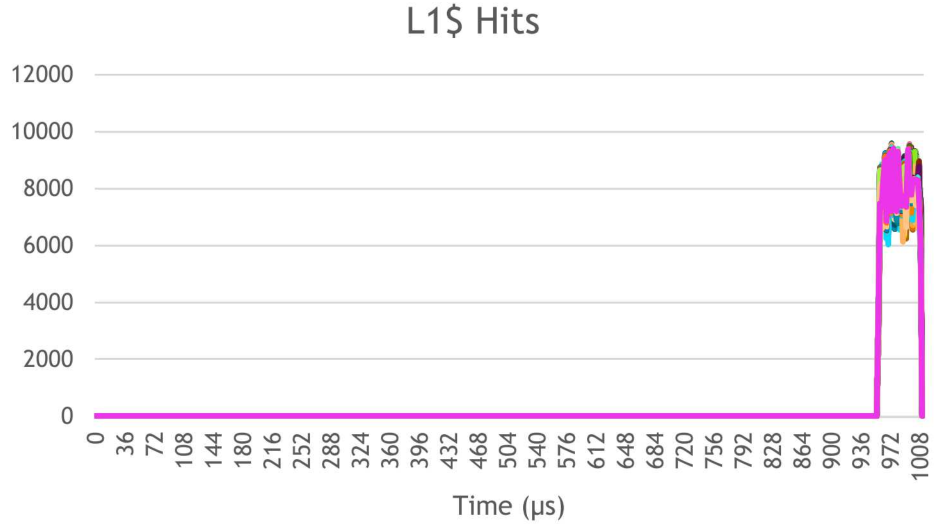


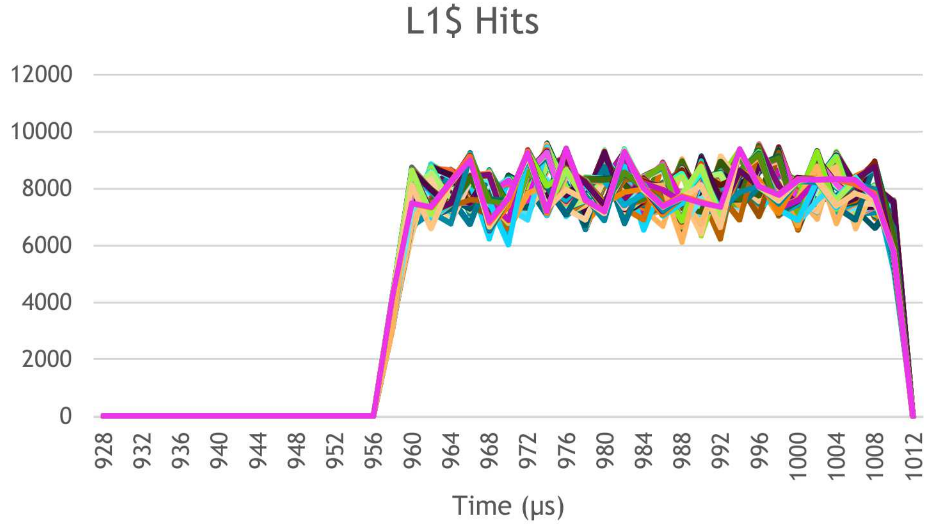


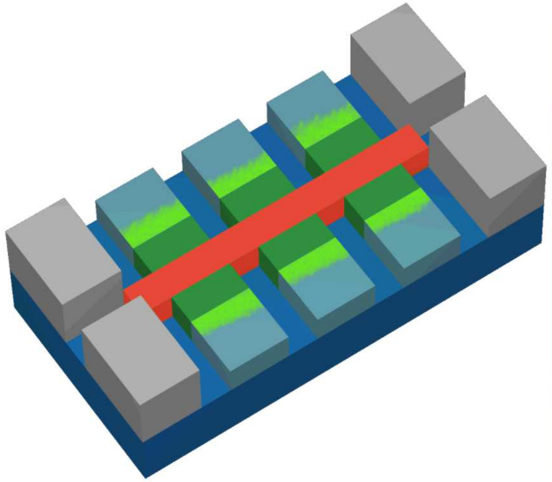
Performance Analysis L1 Cache – Lulesh



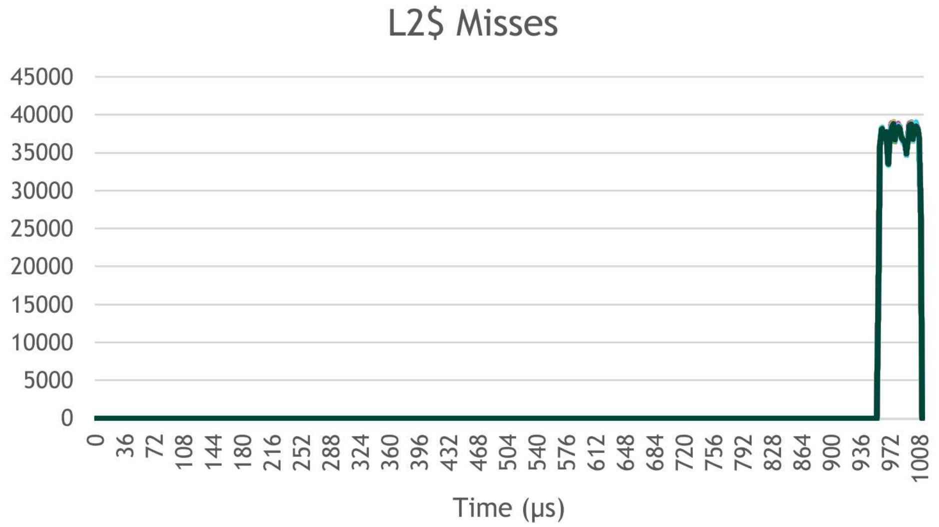


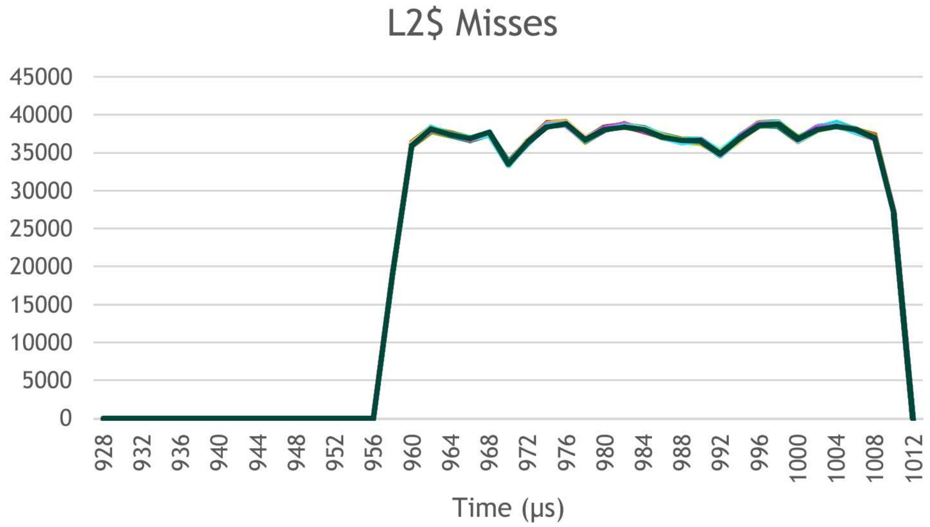


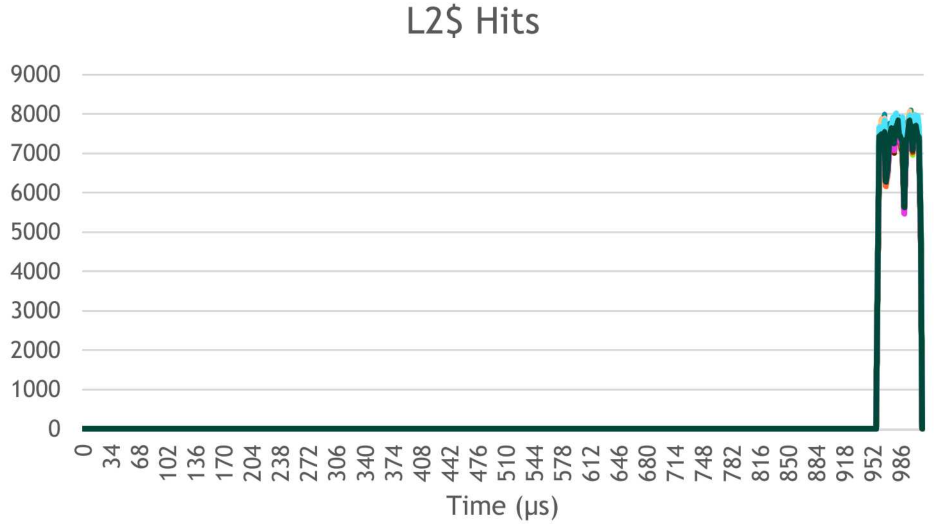


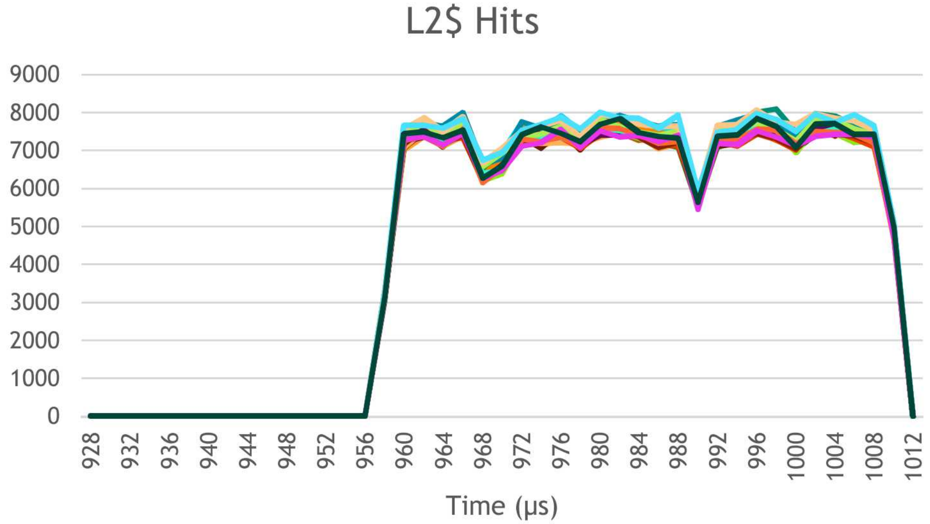


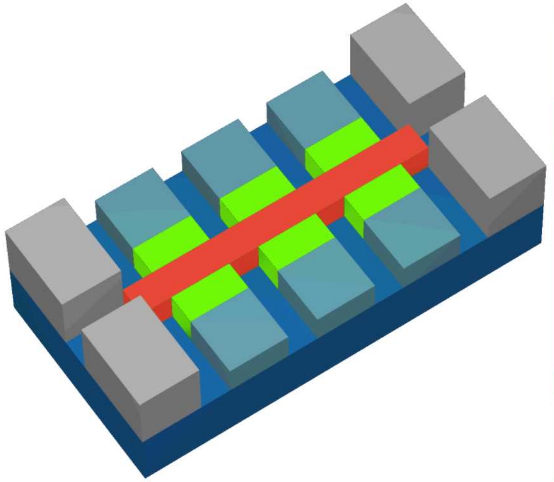
Performance Analysis L2 Cache – Lulesh



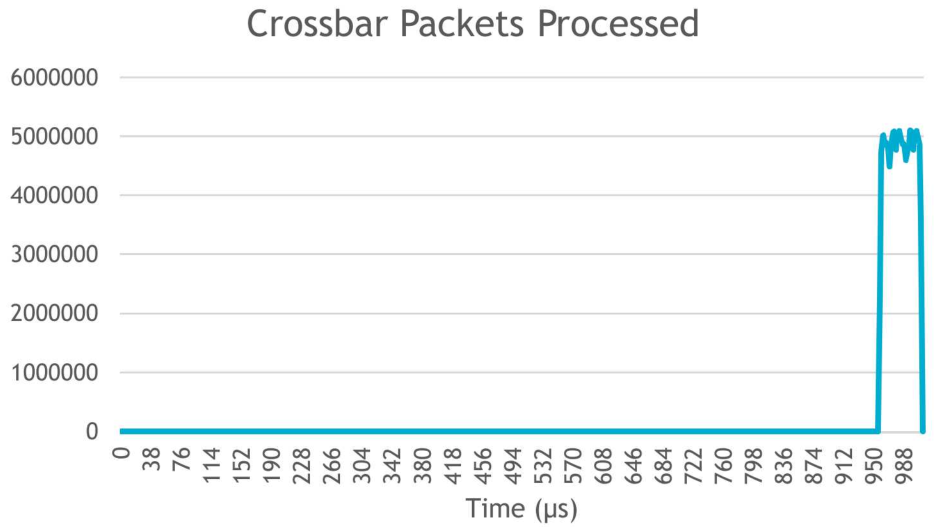


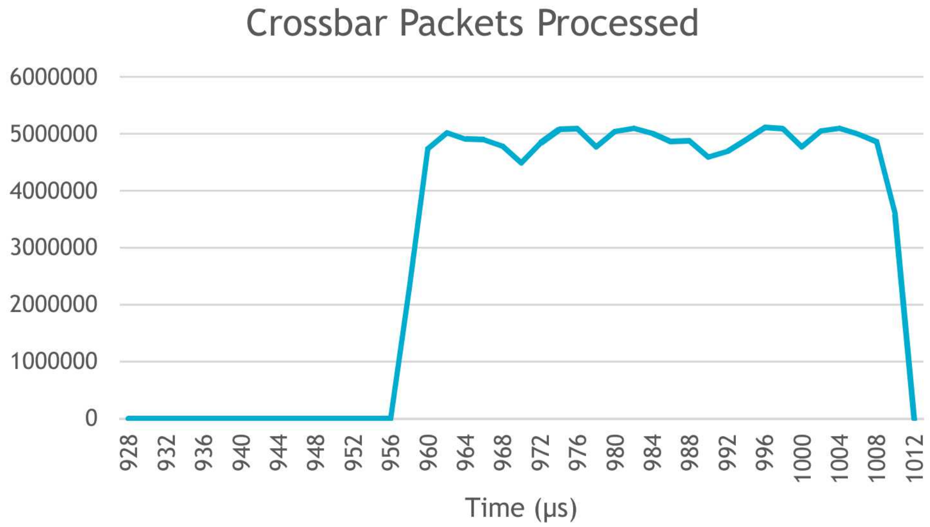


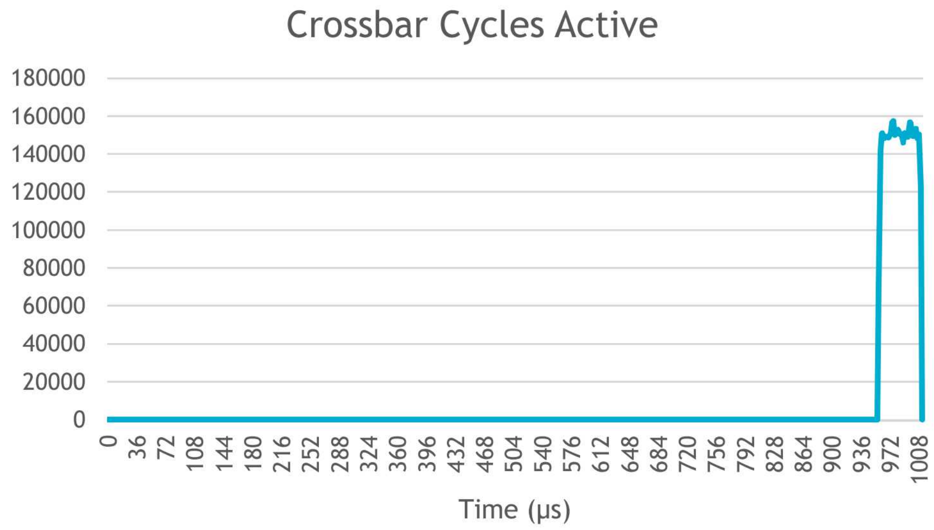


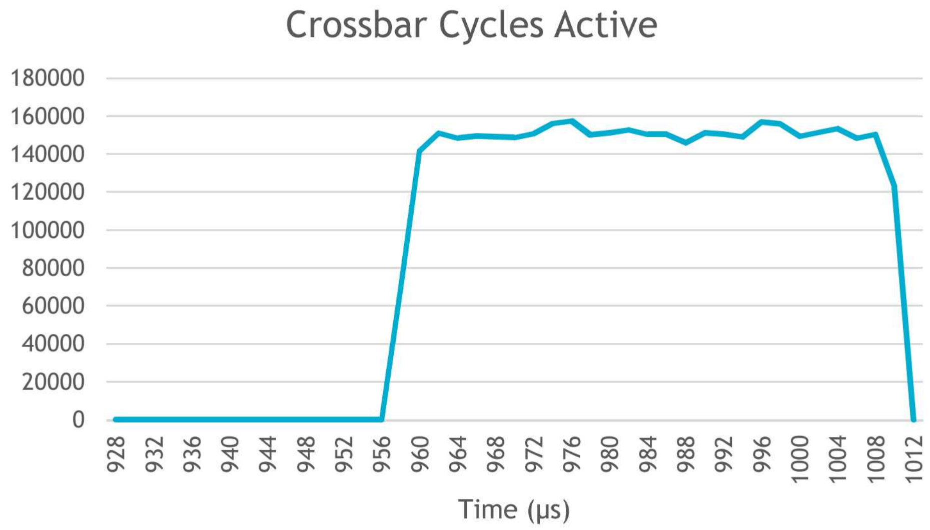


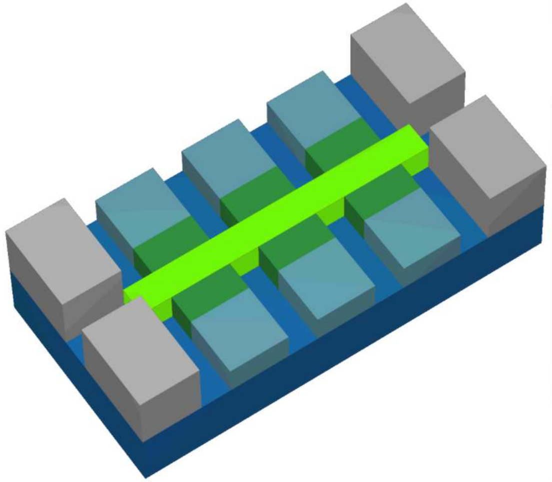
Performance Analysis Crossbar – Lulesh



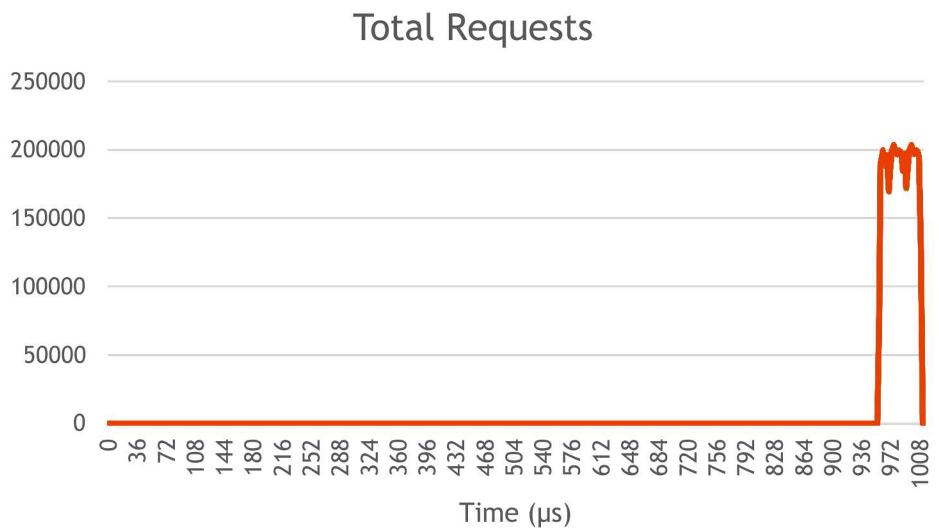


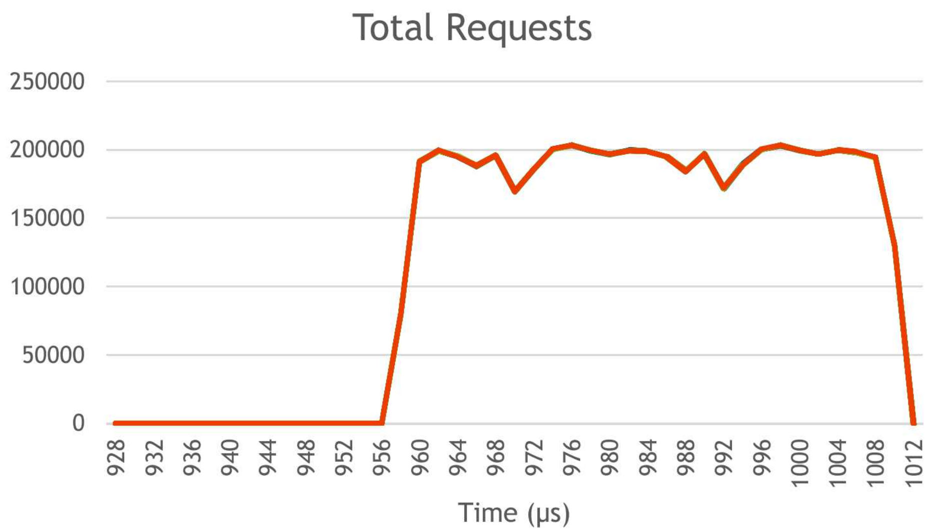


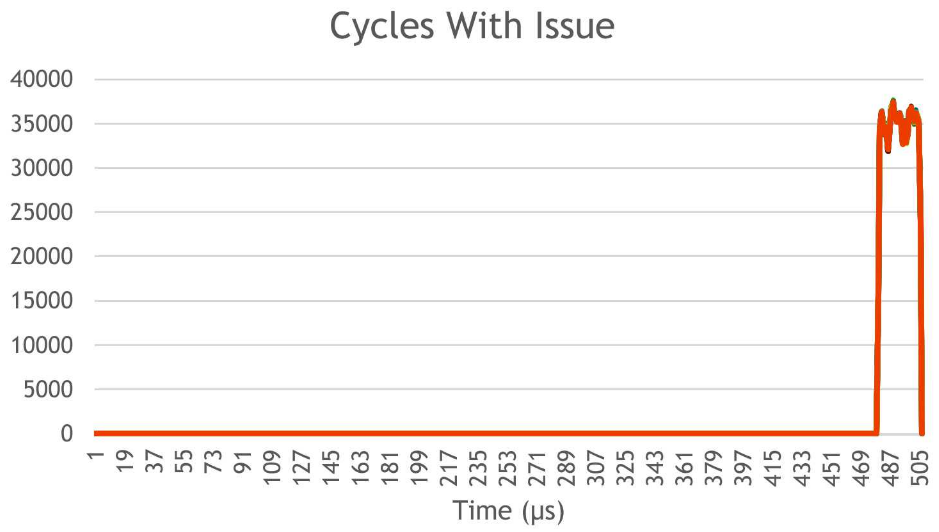


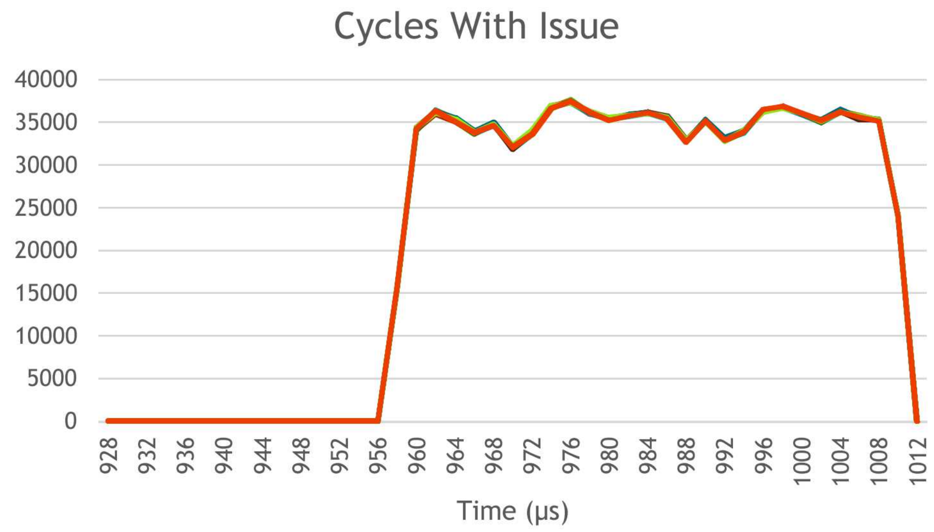


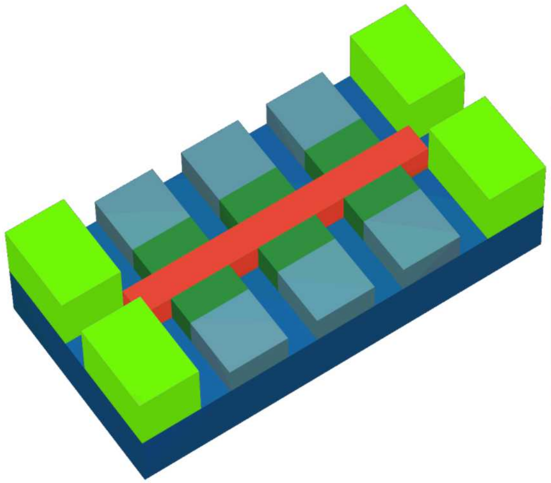
Performance Analysis Memory Controller– Lulesh



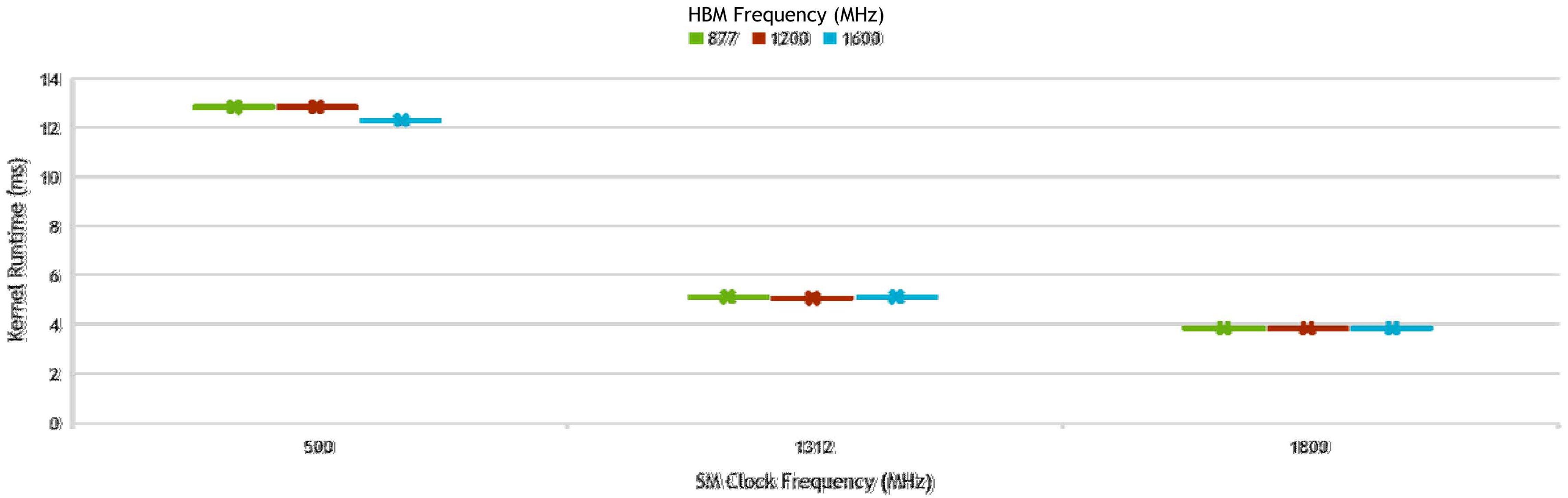








Design Space Exploration – Lulesh



Parameter sweep of SM clock frequency and memory frequency

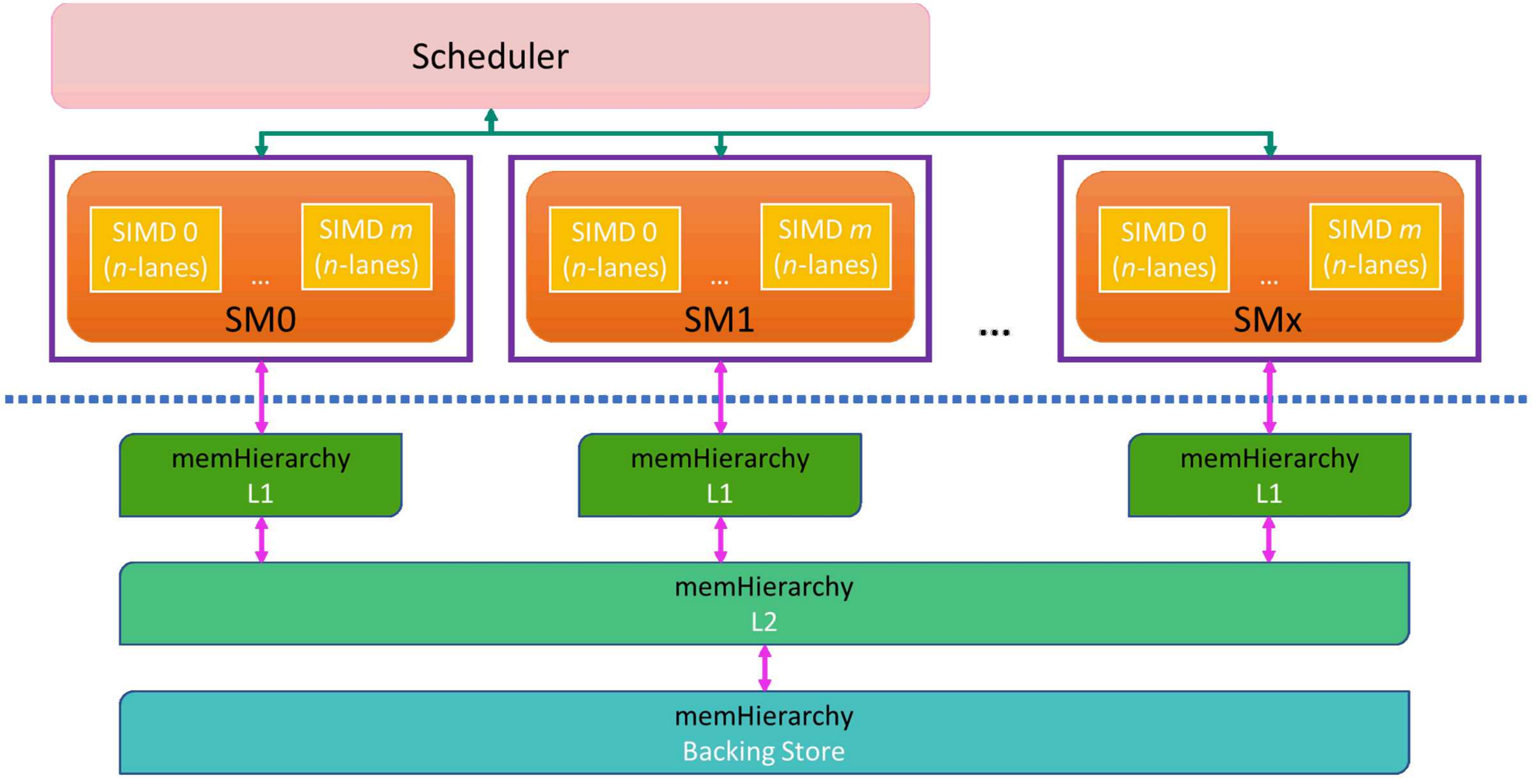
- SM frequency: 500MHz, **1312MHz**, 1800Mhz (2.5x → 1.3x speedup)
- HBM frequency: **877MHz**, 1200MHz, 1600Mhz (0.99x → 1.04x speedup)

Future Work and Summaries

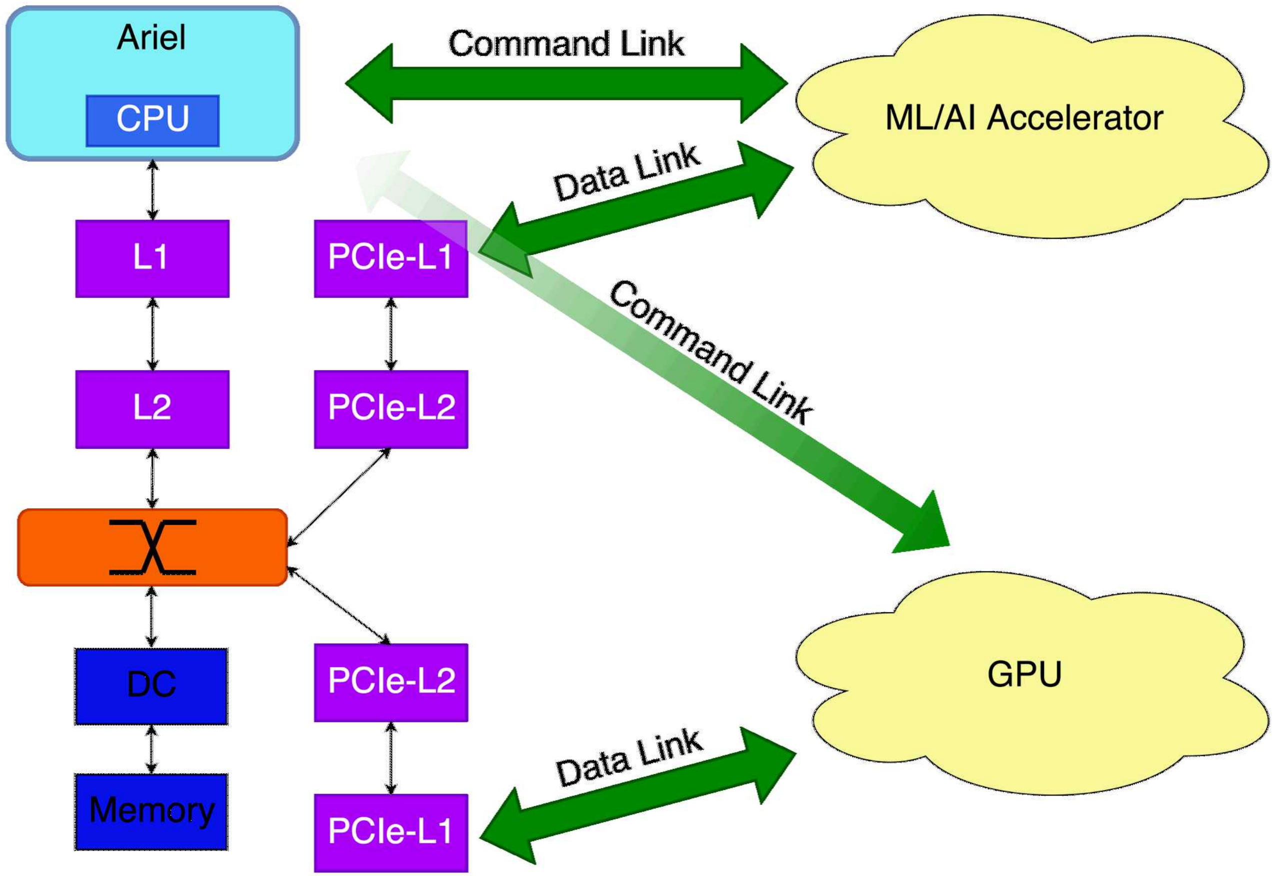
Future Work

Split Monolithic GPGPU Component into SM Components

- Speed up simulations
- Wider range of system options
  - Chiplets
  - Split interconnects



Position SST For the Future



Proves that SST can be integrated with complex external models

- Exploring new ISA plugins for Ariel
  - ARM (SVE?), other alternatives
- HDL simulation components
  - Low-level hardware design including possible path to tape-out
- Neural-inspired & quantum
  - Exploration of interfaces and data movement
- Near memory/storage computing
  - Micron
- Easier integration with vendor-supplied accelerator models

Lessons Learned

The philosophy behind the design of SST has always been modularity

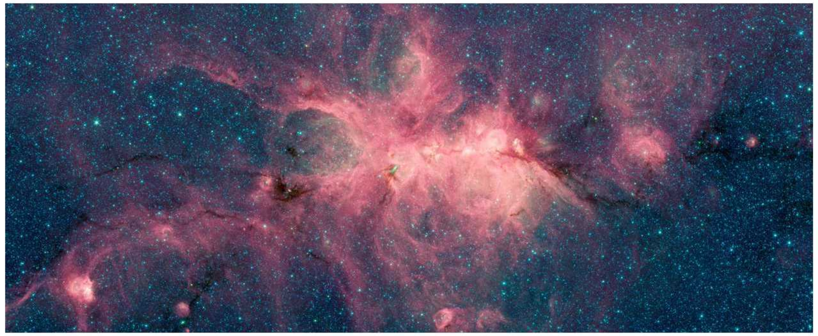
- Any component should be able to communicate with any other component, assuming that the appropriate interface is available, with no other dependencies (mostly)
- The integration of the GPU component has validated this design approach
  - Dropped in an “unknown” component and the memory subsystem…works!

The academic community, particularly in architecture, has little to no exposure to applications of ASC-level code complexity

- Genuinely surprised that so many of the applications had trouble with the parser
- Some instructions not even implemented because they had never encountered them

SST needs an “accelerator” component to hide complexity

- Will probably look like the current GPU model with bus links (PCIe, Gen-Z, CXL) and one or more back channel command/control links



Remaining Issues

Support for UVM

Statistics are still resident in the GPU component and separate from SST

PTX parser has trouble with some instructions when compiling for sm_70

PTX parser is unable to locate a post dominator instruction in some instances

Texture cache is not implemented in functional and timing models

- Presents a problem for some Kokkos-based applications

Some of the Thrust library is not implemented or is implemented incorrectly leading to segfaults

Need write-through support in memHierarchy

Upstream GPGPUSim changes to mainline

Primary SST Components: memHierarchy

Collection of interoperable memory system elements

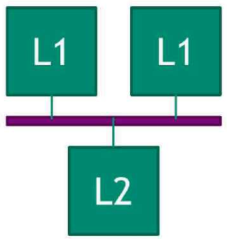
- Caches and directories
- Memory models (DDR, HBM, HMC, NVRAM, etc.)
- Memory controllers & network interfaces for memory (MemNICs)

Inter- and intra-socket coherence

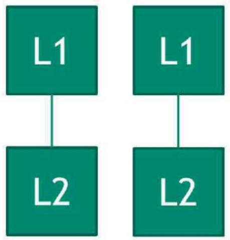
Correlated with modern memory hierarchies

- HBM2/3 Evaluation on Many-core CPU
- Messier: A Detailed NVM-Based DIMM Model for the SST Simulation Framework

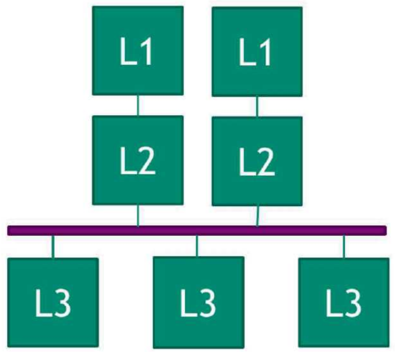
Will allow us to model any sort of memory arrangement that we can imagine!



Private L1
Shared L2



Private L1
Private L2
(would require directory for coherence)



Private L1
Private L2
Distributed shared L3

PTX vs. SASS

PTX

```
$Lt_25_13570:
  ld.global.s32      %r9, [%rd5+0];
  add.s32            %r10, %r9, %r8;
  ld.global.s32      %r11, [%rd5+1024];
  add.s32            %r8, %r11, %r10;
  add.u32            %r5, %r7, %r5;
  add.u64            %rd5, %rd5, %rd6;
  ld.param.u32       %r6, [size];
  setp.lt.u32        %p2, %r5, %r6;
  @%p2 bra           $Lt_25_13570;
...
  mov.u32  %r12, 127;
  setp.gt.u32        %p3, %r3, %r12;
  @%p3 bra           $Lt_25_14082;
  ld.shared.s32      %r13, [%rd10+512];
  add.s32            %r8, %r13, %r8;
  st.shared.s32      [%rd10+0], %r8;
$Lt_25_14082:
  bar.sync  0;
```

SASS (PTXPlus)

```
l0x00000060:
  add.half.u32       $r7, $r4, 0x00000400;
  ld.global.u32      $r8, [$r4];
  ld.global.u32      $r7, [$r7];
  add.half.u32       $r0, $r5, $r0;
  add.half.u32       $r6, $r8, $r6;
  set.gt.u32.u32     $p0/$o127, s[0x0020], $r0;
  add.half.u32       $r6, $r7, $r6;
  add.half.u32       $r4, $r4, $r3;
  @$p0.ne bra        l0x00000060;
...
  set.gt.u32.u32     $p0/$o127, $r2, const [0x0000];
  @$p0.equ add.u32   $ofs2, $ofs1, 0x00000230;
  @$p0.equ add.u32   $r6, s[$ofs2+0x0000], $r6;
  @$p0.equ mov.u32   s[$ofs1+0x0030], $r6;
  bar.sync  0x00000000;
```

Why SST?

Problem: Simulation is slow

- Tradeoff between accuracy and time to simulate
- Many simulators are serial, unable to simulate very large systems

Problem: Lack of simulator flexibility

- Tightly-coupled simulations: Difficult to modify
- Difficult to simulate at different levels of accuracy

The Structural Simulation Toolkit:
A parallel, discrete-event simulation framework
for scalability and flexibility

What is SST?

Goals

- Become the standard architectural simulation framework for HPC
- Be able to evaluate future systems on DOE/DOD workloads
- Use supercomputers to design supercomputers

Status

- Parallel Core, basic components
- Current Release (7.1)
    - Improved components
    - Modular core/elements
    - More Internal documentation

Technical Approach

- Parallel
    - Parallel Discrete Event core with <u>conservative optimization over MPI/Threads</u>
- Multiscale
    - Detailed and simple models for processor, network, & memory
- Interoperability
    - DRAMSim, ,memory models
    - routers, NICs, schedulers
- Open
    - Open Core, non-viral, modular

Consortium

-
-

Key Capabilities

Parallel

- Built from the ground up to be scalable
- Demonstrated scaling to 512+ host processors
- Conservative, Distance-based Optimization
- MPI + Threads

Flexible

- Enables “mix and match” of simulation components
- Multiscale tradeoff between accuracy and simulation time
  - *e.g.*, cycle-accurate network with trace-driven endpoints
- Open API
  - Easily extensible with new models
  - Modular framework
  - Open-source core

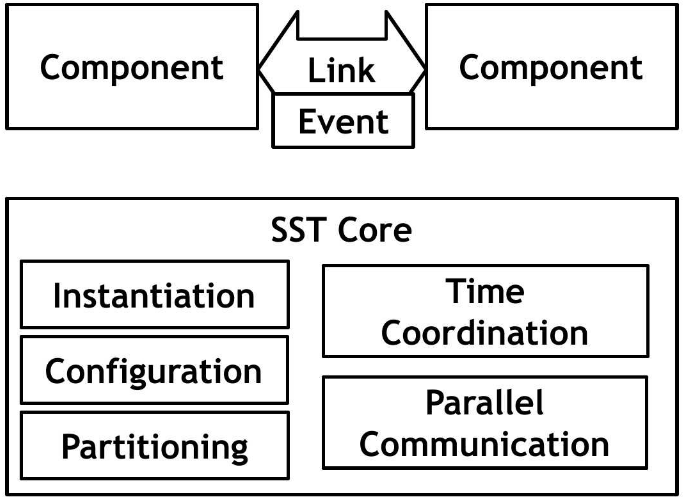
SST's discrete-event algorithm

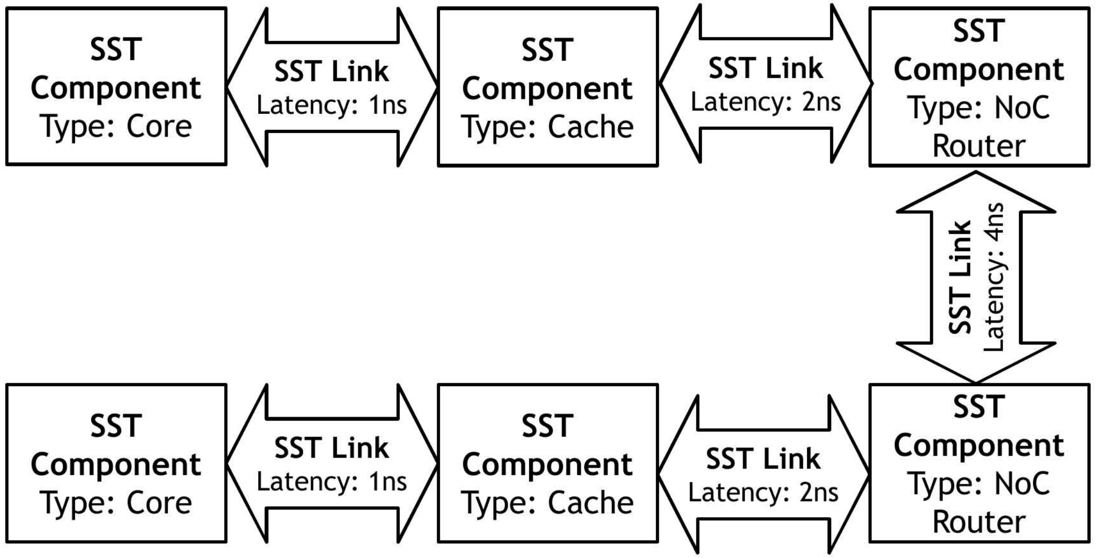
Simulations are comprised of components connected by links

Components interact by sending events over links

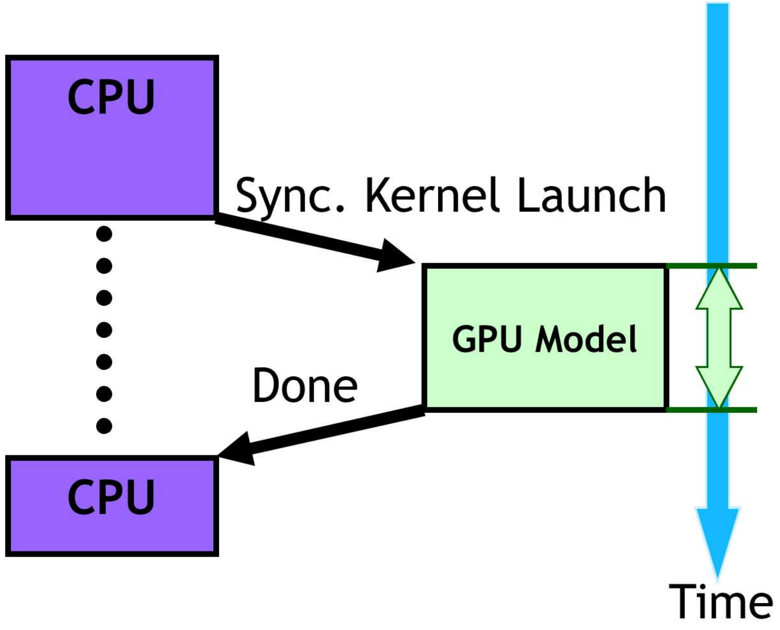
Each link has a minimum latency

Components can load subComponents and modules for additional functionality





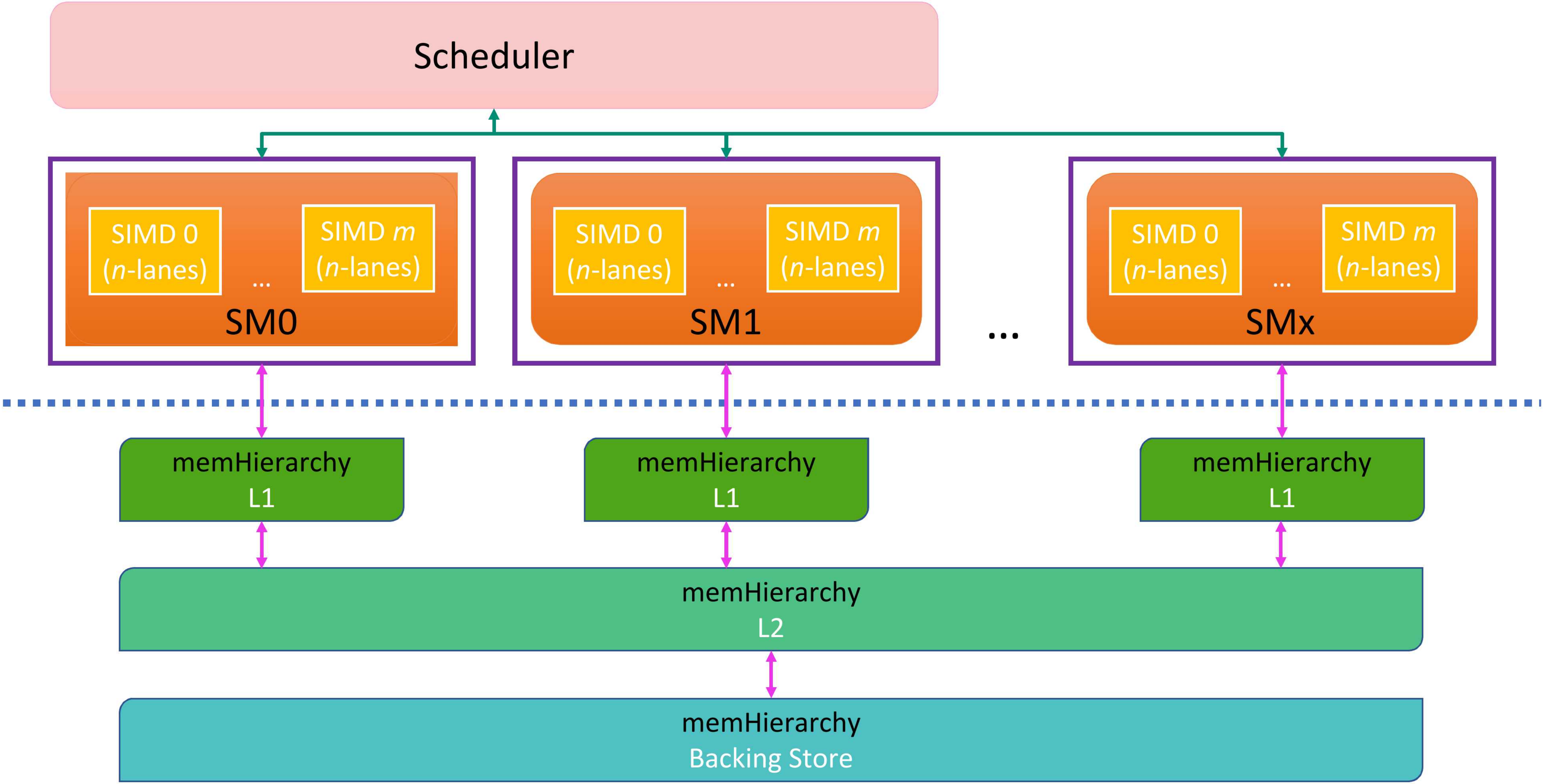
Currently only supports synchronous kernel launches



Kokkos Kernels Unit Test Results

1	abs_double
2	abs_mv_double
3	asum_double
4	axpby_double
5	axpby_mv_double
6	axpy_double
7	axpy_mv_double
8	dot_double
9	dot_mv_double
10	mult_double
11	mult_mv_double
12	nrm1_double
13	nrm1_mv_double
14	nrm2_double
15	nrm2_mv_double
16	nrm2_squared_double
17	nrm2_squared_mv_double
18	nrminf_double
19	nrminf_mv_double
20	reciprocal_double
21	reciprocal_mv_double
22	scal_double
23	scal_mv_double
24	sum_double
25	sum_mv_double
26	update_double
27	update_mv_double
28	gemv_double
29	gemm_double
30	sparse_spgemm_double_int_int_TestExecSpace
31	sparse_spgemm_double_int_size_t_TestExecSpace
32	sparse_spadd_double_int_int_TestExecSpace
33	sparse_spadd_double_int_size_t_TestExecSpace
34	sparse_gauss_seidel_double_int_int_TestExecSpace
35	sparse_gauss_seidel_double_int_size_t_TestExecSpace
36	sparse_block_gauss_seidel_double_int_int_TestExecSpace
37	sparse_block_gauss_seidel_double_int_size_t_TestExecSpace
38	sparse_crsmatrix_double_int_int_TestExecSpace
39	sparse_crsmatrix_double_int_size_t_TestExecSpace
40	sparse_blkcrsmatrix_double_int_int_TestExecSpace
41	sparse_blkcrsmatrix_double_int_size_t_TestExecSpace
42	sparse_replaceSumIntoLonger_double_int_int_TestExecSpace
43	sparse_replaceSumIntoLonger_double_int_size_t_TestExecSpace
44	sparse_replaceSumInto_double_int_int_TestExecSpace
45	sparse_replaceSumInto_double_int_size_t_TestExecSpace
46	graph_graph_color_double_int_int_TestExecSpace
47	graph_graph_color_double_int_size_t_TestExecSpace
48	graph_graph_color_deterministic_double_int_int_TestExecSpace
49	graph_graph_color_deterministic_double_int_size_t_TestExecSpace
50	graph_graph_color_d2_double_int_int_TestExecSpace
51	graph_graph_color_d2_double_int_size_t_TestExecSpace
52	common_ArithTraits
53	common_set_bit_count
54	common_ffs
55	batched_scalar_serial_set_double_double
56	batched_scalar_serial_scale_double_double
57	batched_scalar_serial_gemm_nt_nt_double_double
58	batched_scalar_serial_gemm_t_nt_double_double
59	batched_scalar_serial_gemm_nt_t_double_double
60	batched_scalar_serial_gemm_t_t_double_double
61	batched_scalar_serial_trsm_l_l_nt_u_double_double
62	batched_scalar_serial_trsm_l_l_nt_n_double_double
63	batched_scalar_serial_trsm_l_u_nt_u_double_double
64	batched_scalar_serial_trsm_l_u_nt_n_double_double
65	batched_scalar_serial_trsm_r_u_nt_u_double_double
66	batched_scalar_serial_trsm_r_u_nt_n_double_double
67	batched_scalar_serial_trsm_l_l_t_u_double_double
68	batched_scalar_serial_trsm_l_l_t_n_double_double
69	batched_scalar_serial_trsm_l_u_t_u_double_double
70	batched_scalar_serial_trsm_l_u_t_n_double_double
71	batched_scalar_serial_gemv_nt_double_double
72	batched_scalar_serial_gemv_t_double_double
73	batched_scalar_serial_trsv_l_nt_u_double_double
74	batched_scalar_serial_trsv_l_nt_n_double_double
75	batched_scalar_serial_trsv_u_nt_u_double_double
76	batched_scalar_serial_trsv_u_nt_n_double_double
77	batched_scalar_team_set_double_double
78	batched_scalar_team_scale_double_double
79	batched_scalar_team_gemm_nt_nt_double_double
80	batched_scalar_team_gemm_t_nt_double_double
81	batched_scalar_team_gemm_nt_t_double_double
82	batched_scalar_team_gemm_t_t_double_double
83	batched_scalar_team_trsm_l_l_nt_u_double_double
84	batched_scalar_team_trsm_l_l_nt_n_double_double
85	batched_scalar_team_trsm_l_u_nt_u_double_double
86	batched_scalar_team_trsm_l_u_nt_n_double_double
87	batched_scalar_team_trsm_r_u_nt_u_double_double
88	batched_scalar_team_trsm_r_u_nt_n_double_double
89	batched_scalar_team_trsm_l_l_t_u_double_double
90	batched_scalar_team_trsm_l_l_t_n_double_double
91	batched_scalar_team_trsm_l_u_t_u_double_double
92	batched_scalar_team_trsm_l_u_t_n_double_double
93	batched_scalar_team_gemv_nt_double_double
94	batched_scalar_team_gemv_t_double_double
95	batched_scalar_serial_lu_double
96	batched_scalar_serial_inverselu_double
97	batched_scalar_serial_solvelu_double
98	batched_scalar_team_lu_double
99	batched_scalar_team_inverselu_double
100	batched_scalar_team_solvelu_double

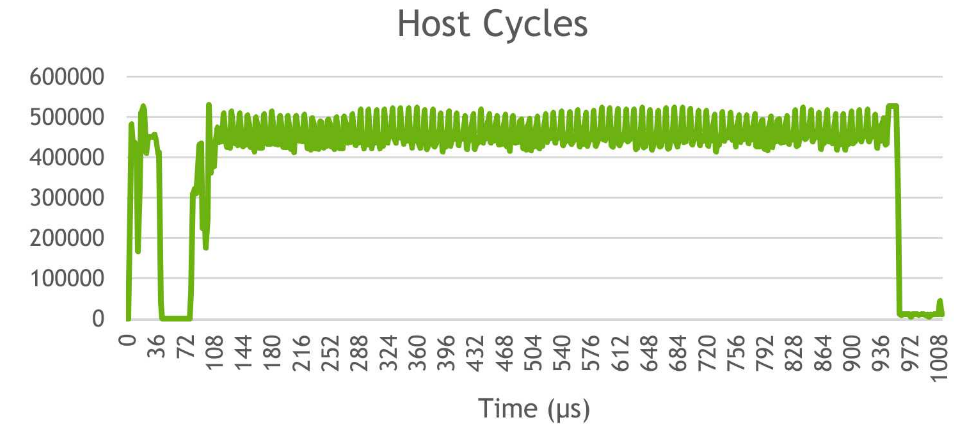
Split Monolithic GPGPU Component into SM Components

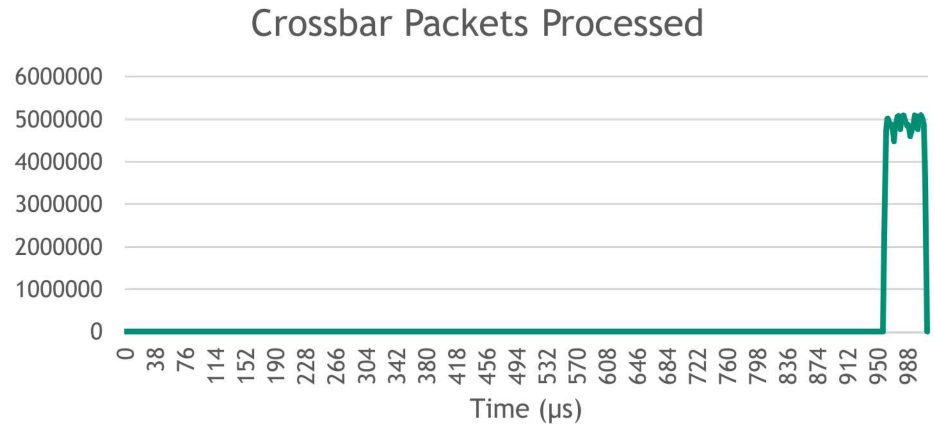


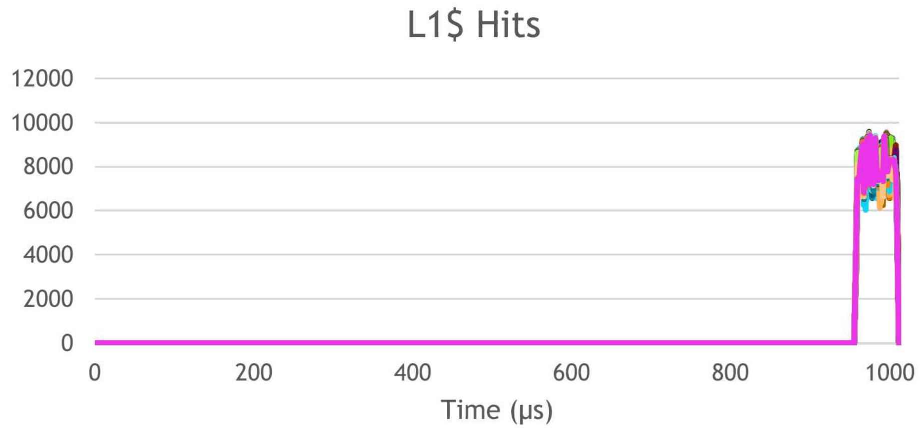
Performance Analysis – Lulesh

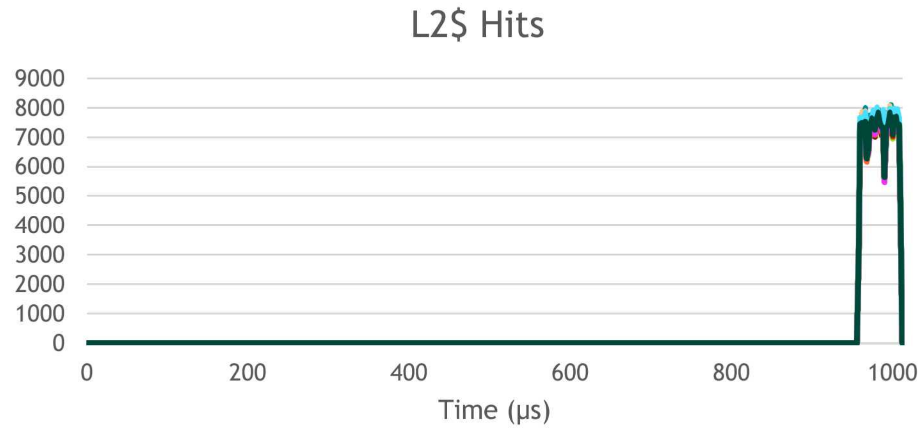
SST is capable of providing periodic statistic dumps for all of the currently loaded components

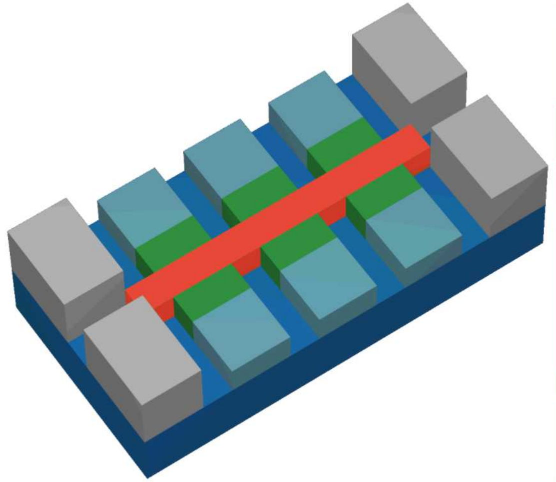
- Valuable for fine-grain performance analysis of applications



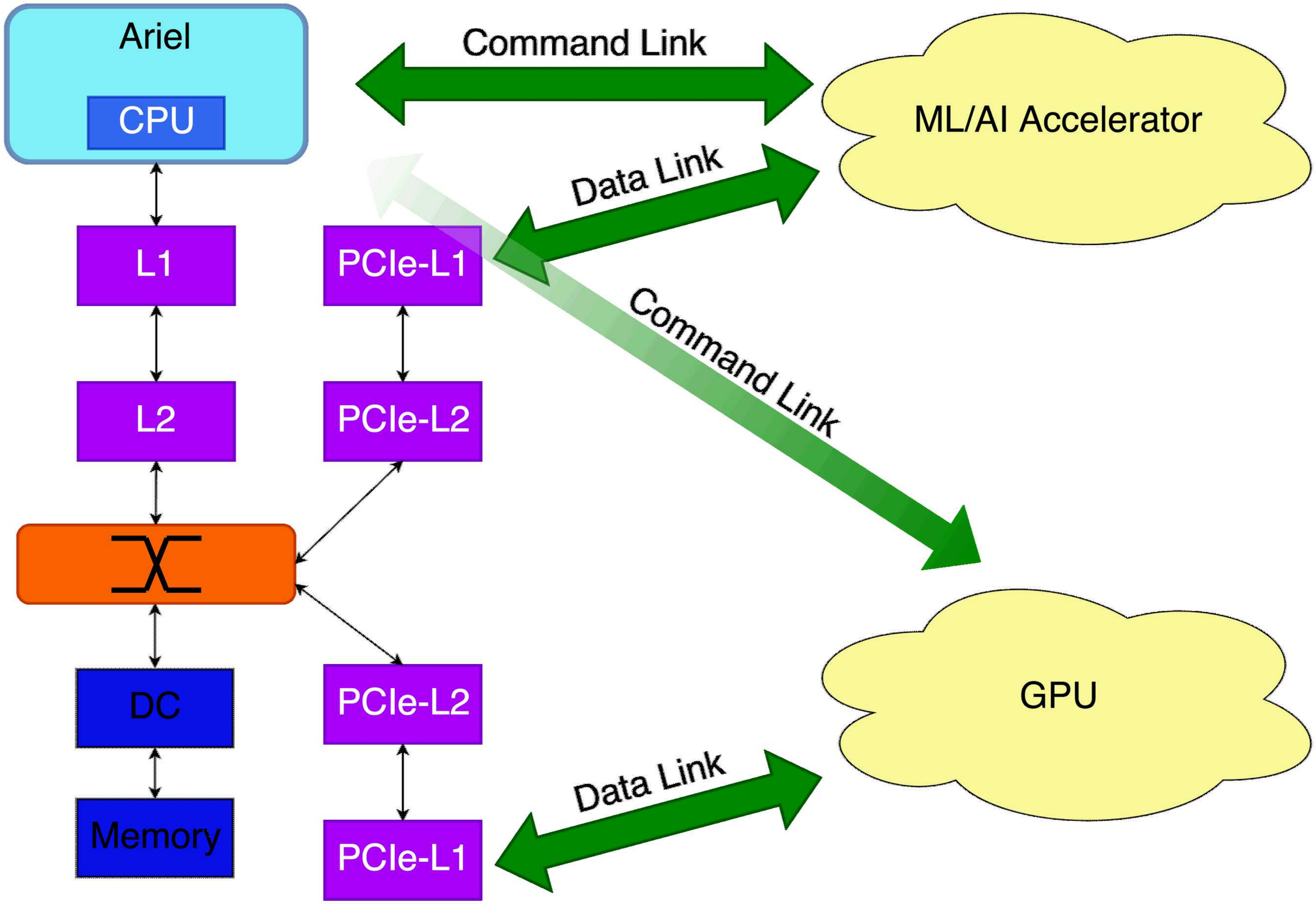








GPGPUSim Integration



CLICK TO EDIT MASTER TITLE STYLE

Slide Left Blank