PRINCETON UNIVERSITY

MuchiSim: A Simulation Framework for Design Exploration of Multi-Chip Manycore Systems

Marcelo Orenes-Vera, Esin Tureci, Margaret Martonosi and Stojche Nakov
{movera, esin.tureci, mrm, sn3332} @princeton.edu

Getting Started! Open the terminal and go to the path where you want to have the simulator:

```
git clone https://github.com/PrincetonUniversity/muchiSim.git
cd muchiSim
source setup.sh
```

If the C++ compiler was detected successfully, then you can run the first basic experiment:

```
exp/run_app.sh
```

Installing the Python environment to use our cool visualization tools ☺

```
python3 -m venv gui/env_viz
source gui/env_viz/bin/activate
pip install --upgrade pip && pip install PyQt5 matplotlib imageio
```

Tutorial Website

github.com/PrincetonUniversity/MuchiSim

PRINCETON UNIVERSITY

Acknowledgements

- PI David Wentzlaff (Princeton) + Co-PIs Margaret Martonosi (Princeton) and Luca Carloni (Columbia University)
- Funding: DARPA MTO Software-Defined Hardware Program
- The Full DECADES Team!
  - https://decades.cs.princeton.edu

PRINCETON UNIVERSITY

Plan for the Day

First Half

Introduction

Target Architectures covered by MuchiSim

Simulator Infrastructure

Downloading and Basic Experiment

Reading a Simulator Report

Parallel Simulation

Simulator Configurations

System Hardware Configurations

Energy, Area & Cost Parameters

Sweeping Through Parameter Values

Case Study: Cerebras Wafer Scale Engine

Application-Hardware Interactions: Programming Model

Application Suite

Task-based Programs

Adding Applications to MuchiSim

Datasets Included in the Repo

Second Half

Visualization Tools

Case Study: Regional Reductions

Visualization Tools

Case Study: Memory-Compute Ratio

End-to-end Experiment

Simulation Speed & Scalability

Conclusions & Future Work

Work that users could do to explore System design and application mapping

Future work on the simulator itself

PRINCETON UNIVERSITY

MuchiSim's Roots: DARPA-funded DECADES Project

Language and Compiler Support

Lead: Margaret Martonosi

- Enhance data locality
- Optimize spatial mapping of threads
- Enable in-memory computing

Very Coarse-Grained Reconfigurable Tile-Based Architecture

Lead: Luca Carloni

- Accelerator rich architecture
- Reconfigurable interconnection network
- Reconfigurable in-memory computing
- Embedded Machine Learning Accelerators

Multi-Tiered Demonstration Strategy

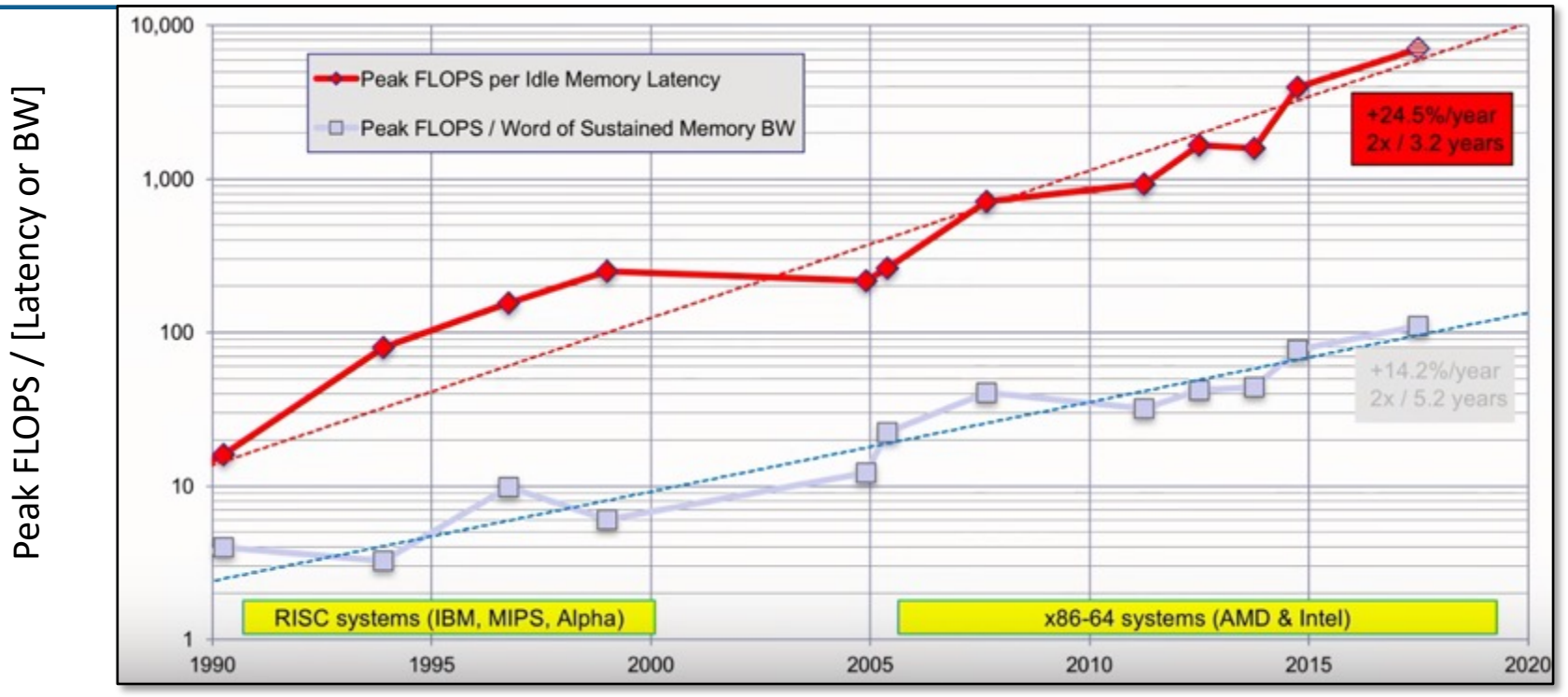
Lead: David Wentzlaff

- DECADES chip prototype
- In-memory Computing Hardware
- Multi-FPGA emulation infrastructure

PRINCETON UNIVERSITY

From Our Original Project Slides:
Data Supply = Fundamental Bottleneck in Accelerator-Oriented Systems

- Amdahl's Law: Accelerating compute makes data supply bottlenecks look relatively bigger!
- Key memory/comm bottlenecks lie in supplying specialized accelerators with data
- Different apps -> different data supply needs



John McCalpin, SC'16 Keynote

Latency and Bandwidth:

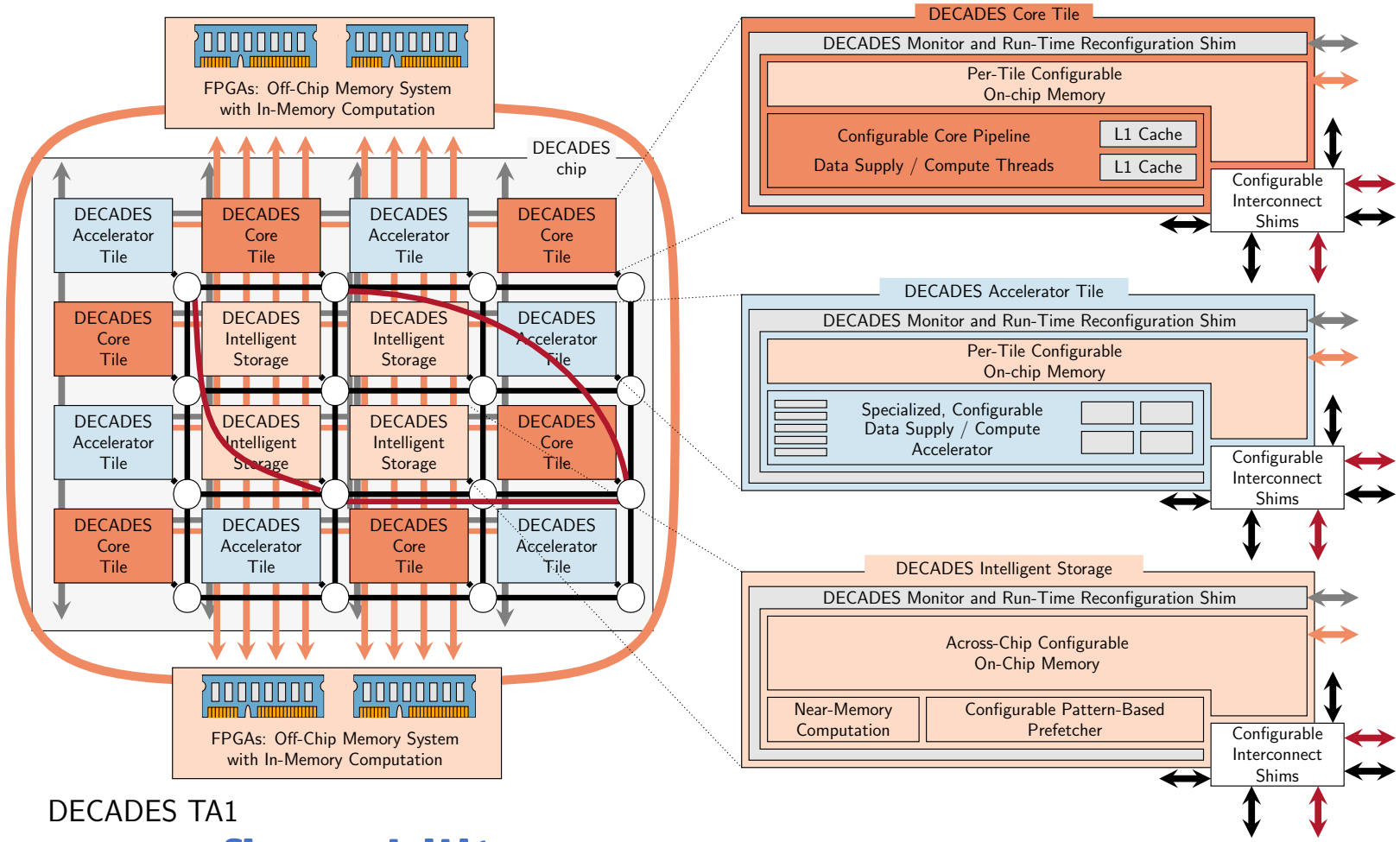
- Accelerator often lacks general-purpose latency-tolerance mechanisms (e.g., OoO execution, Multithreading)
- Improving Accelerator compute throughput increases memory bandwidth pressure

5

DECADES Platform Architecture

Computations mapped onto core tiles or available accelerator tiles

Dynamic reconfiguration of Supply-Compute decoupling, power-performance tradeoffs, and interconnect



DECADES TA1

Heterogeneity meets coarse reconfigurability

Open-Source Tiles, Compiler, Emulators

PRINCETON UNIVERSITY

Our Simulator Needs

- Extreme Parallelism: Thousands or Millions of processing elements
- Chiplet Ecosystem: Interconnect and Memory system focus
- Support for applications with sparse and irregular control and dataflow patterns: Graphs, Sparse, etc.

➔We created MuchiSim to support this future-looking research.

➔Today's tutorial: Encourage other users as well!

PRINCETON UNIVERSITY

Outline First Half

Introduction
Motivation

Target Architectures covered by MuchiSim

Simulator Infrastructure

Downloading and Basic Experiment

Reading a Simulator Report

Parallel Simulation

Simulator Configurations

System Hardware Configurations

Energy, Area & Cost Parameters

Sweeping Through Parameter Values

Case Study: Cerebras Wafer Scale Engine

Application-Hardware Interactions: Programming Model

Application Suite

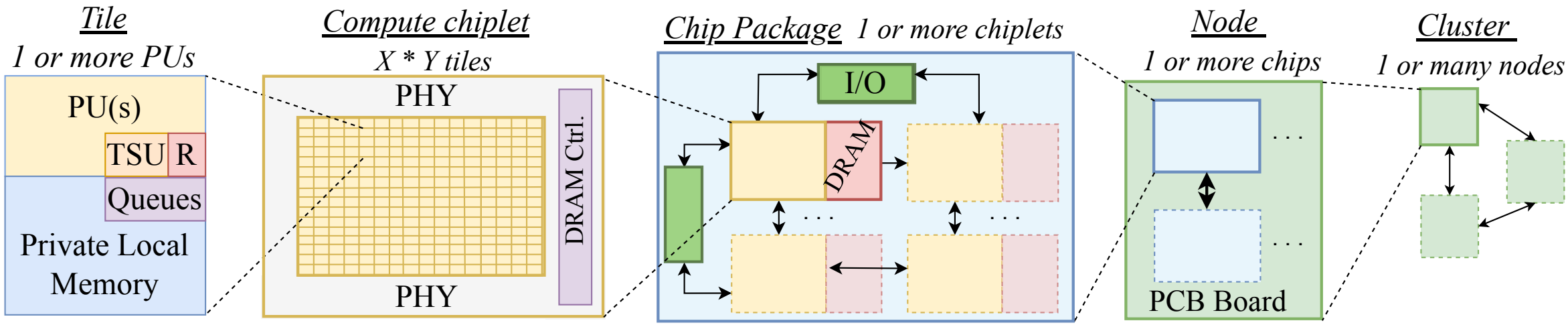
Task-based Programs

Adding Applications to MuchiSim

Datasets Included in the Repo

PRINCETON UNIVERSITY

Hierarchical & Scalable Architectures



- Each **Tile** contains 1+ Processing Units (PUs), a router, and a private local memory (PLM)
  - **TSU**: Manage and schedule tasks
  - **Queues**: Coordinate ordered communication.
- A compute **Chiplet** has X*Y tiles and may include a memory controller (if attaching DRAM)
- A chip **Package** has a grid of compute chiplets (with or without DRAM attached).
- A compute **Node** is a board with 1+ chip Packages.

PRINCETON UNIVERSITY

What is changeable in this?

- Variations on task scheduling via changes to TSU Model
- Interconnect topology, bandwidth and latency
- Variations on memory: SRAM-per-tile, scratchpad vs cache, on-package HBM
- Granularity: number of PUs per tile, tiles per chiplet.
- Communication Scheme: Routing exposed to software applications; Logical mappings of application data and task invocations to tiles.

PRINCETON UNIVERSITY

Outline First Half

Introduction
Motivation

Target Architectures covered by MuchiSim

Simulator Infrastructure

Downloading and Basic Experiment

Reading a Simulator Report

Parallel Simulation

Simulator Configurations

System Hardware Configurations

Energy, Area & Cost Parameters

Sweeping Through Parameter Values

Case Study: Cerebras Wafer Scale Engine

Application-Hardware Interactions: Programming Model

Task-based Programs

Adding Applications to MuchiSim

Application Suite

Datasets Included in the Repo

PRINCETON UNIVERSITY

Downloading and Initial Compilation

If you haven't done it yet, open the terminal and go to the path where you want to have the simulator and download it. You have the commands in www.github.com/PrincetonUniversity/muchiSim/blob/main/tutorial/**1.COMPILATION.md**

```
git clone https://github.com/PrincetonUniversity/muchiSim.git
cd muchiSim
source setup.sh
```

If the C++ compiler was detected successfully, then you can run the first basic experiment:

```
exp/run_app.sh
```

PRINCETON UNIVERSITY

Running the Simulator

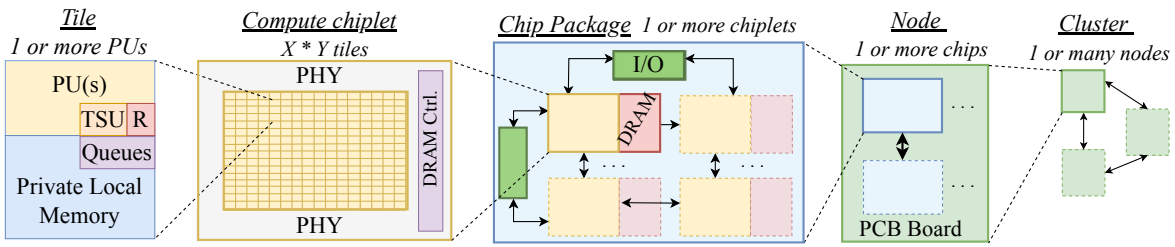
Parallel Simulator written in C/C++ only using standard libraries, no external dependencies

➢ A Bash script configures the system to simulate and launches the simulation

exp/run.sh [-a app | -b barrier | -c chiplet_width | -d datasets | -e proxy_width | -f force_proxy_ratio | -g large_queue | -j write_thru | -k board_width | -l express_link | -m grid_conf | -n name | -o NOC | -p dry_run | -q queue_mode | -r assert_mode | -s machine_to_run | -t max_threads | -u noc_conf | -v verbose | -w SRAM size-per-tile | -x num_phy_nocs | -y dcache | -z proxy_routing]

- **Experiment name.** Scripts that use the run.sh script to setup experiment-specific configurations, e.g. **run_exp_granularity.sh** (*will cover later*)
- Max number of **host threads** to run with
- **Application** (we'll cover later the benchmark and datasets included in the repo)
- **Dataset**

- **Total Grid size,** up to 1024x1024 evaluated.
- **Chiplet** width, E.g., 16x16.
- **Board** width [Default size of Grid]



- **NoC** type, e.g., mesh, torus
- Num Physical **NoCs**
- **NoC width** configuration, including inter-die direct channels for multi-chiplet systems

+ Many other configurations settable through set_conf (More later)

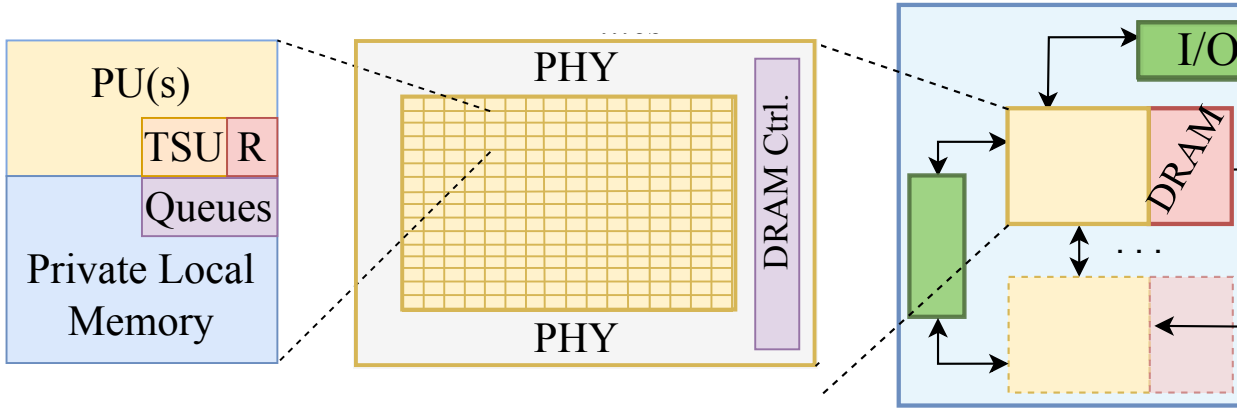
PRINCETON UNIVERSITY

Basic Experiment Configuration

exp/run_app.sh

```
3   # Check if first argument is provided, otherwise set default APP to SPMV (4)
4   if [ -z "$1" ]; then
5       echo "Default APP is SPMV (4)"
6       app=4
7   else
8       app=$1 # [0: sssp, 1: pagerank, 2: bfs, 3: wcc, 4: spmv, 5: histogram, 6: fft, 7: spmv_alternative, 8: spmm]
9   fi
```

```
31  th=2 # Number of threads
32  assert_mode=1 # Enable assertions within the simulator code
33  verbose=2 # Level 1: print basic stats on each sample, Level 2: print full stats on each iteration, etc.
34  dry_run=0 # [0: run the simulator, 1: not running the simulator, only print the configuration block in the logs]
35
36  # Configuration for test_case 0
37  if [ $test_case -eq 0 ]; then
38      datasets="Kron16"
39      grid_x=16
40      chiplet_w=$((grid_x))
41      pack_w=$((grid_x))
42      proxy_w=$((grid_x))
43      sram_memory=$((256 * 512)) # 512 KiB
```



PRINCETON UNIVERSITY

Basic Experiment Configuration

exp/run_app.sh

```
52  if [ $datasets = "Kron16" ]; then
53      local_run=2 # Run in the foreground (wait for the application result to check it with the reference)
54  else
55      local_run=1 # Run in the background (not outputting the application functional result)
56  fi
57
58  # Dcache type (0: no dcache, only spd, 1: dcache if not fit on spd, 2: always dcache, 3: only prefetching and spatial locatity,
    4: spatial locality only, 5: everything is a miss)
59  let dcache=1
60  let noc_type=1
61  let torus=1
62
63  params="-n ${name}${proxy_w} -o $torus -t $th -u $noc_type -v $verbose -s $local_run -e $proxy_w -c $chiplet_w -k $pack_w -r
    $assert_mode -w $sram_memory -y $dcache -p $dry_run -d $datasets -a $app -m $grid_x"
64  echo "Running $params"
65  exp/run.sh $params
```

PRINCETON UNIVERSITY

Reading a Simulator Report

The **simulation logs** will be saved in the sim_logs folder. Particularly, **the basic experiment you ran with run_app.sh** creates a log called DATA-Kron16--16-X-16--BA16-A4.log

- This log contains first the logs of the dataset being loaded, Kron16, aka, RMAT-16 (containing 2^16 vertices or non-zero elems)
- Later, it shows information about the manycore **configuration** (16x16 grid with a single chiplet of 16x16 tiles, 1 PU/tile, etc.

```
Dataset footprint per tile: 32 KiB
Proxys footprint per tile: 0 KiB
Reserved SRAM per tile: 28 KiB

Ideal SRAM per tile: 60 KiB
reserved_words_for_dataset:24756
dataset_words_per_tile:8240
remaining_sram_words:123782
min_dcache_sram_words_required:8240
dataset_words_per_tile:8240
min_pcache_sram_words_required:0
proxys_words_per_tile:0
Dcache footprint ratio: 1
Dataset Cached: 0
DCache: 8240 words, lines: 515, sets: 515
Proxys Cached: 0
Pcache: 0 words, sets: 0
Node per tile: 256
Edge per tile: 3734
GRID_X: 16
GRID_SIZE: 256
smt_per_tile: 1
noc_freq: 1
pu_freq: 1
RUCHE_X: 0
PROXY_W: 16
DIE_W: 16
DIE_H: 16
DIES: 1
BORDER_TILES_PER_DIE: 64
PACK_W: 16
```

```
Inter-Die #links: 512
Inter-Die (Gbits): 16384
Inter-Die (Gbits/mm): 450.658
Bisection BW of Board (Gbits): 4096
Border IO #Dies: 4
Border IO Die BW (Gbits): 2048
Off-Board BW Total (Gbits): 8192
Off-Board BW Density (Gbits/mm): 124
Die side: 9.09 mm
Die border wire density: 450.66 Gbits/mm
HBM wire density: 450.66 Gbits/mm

Tiles per Die: 256 (16x16)
Tile Area: 0.19 mm2, side: 0.44 mm
- Core/Rout Area: 0.05 mm2
- SRAM Area: 0.14 mm2
- SRAM/Logic Ratio: 2.80x

Dies per Package: 1 (1x1)
Die Area: 82.61 mm2, side: 9.09 mm
- Tiles Area: 49.86 mm2
- PHY Area: 23.74 mm2
- MC Area: 9.00 mm2

Packages per Board: 1 (1x1)
Package Area: 272.61 mm2, side: 16.51 mm
- IOdie Area: 80.00 mm2
- Dies Area: 82.61 mm2
- HBM Area: 110.00 mm2

Boards: 1 (1x1)
Board Area: 272.61 mm2
Total Area: 272.61 mm2
```

PRINCETON UNIVERSITY

Reading a Simulator Report

The **simulation logs** will be saved in the sim_logs folder. Particularly, **the basic experiment you ran with run_app.sh** creates a log called DATA-Kron16--16-X-16--BA16-A4.log

- This log contains first the logs of the dataset being loaded, Kron16, aka, RMAT-16 (containing 2^16 vertices or non-zero elems)
- Later, it shows information about the manycore configuration (16x16 grid with a single chiplet of 16x16 tiles, 1 PU/tile, etc.
- And it's **estimated cost**

```
137   ESTIMATED COST OF SILICON AND PACKAGING. ONLY VARIABLE, NO FIXED COSTS
138   Good dies per wafer 3d: 410
139   Good dies per wafer 2d: 667
140   --Dies 3D Cost: 14.75 $
141   --Dies 2.5D Cost: 9.07 $
142   --Dies 2D Cost: 9.07 $
143   --HBM Cost: 60.00 $
144   --IOdie Cost: 8.44 $
145   --Packaging Cost 3d: 3.93 $
146   --Packaging Cost 2.5d: 5.80 $
147   --Packaging Cost 2d: 2.63 $
148   SiP Cost 3d: 87.12 $
149   SiP Cost 2.5d: 83.30 $
150   SiP Cost 2d: 20.13 $
151   --Board Cost: 0.27 $
152   Total SiPs Cost 3d: 87.39 $
153   Total SiPs Cost 2.5d: 83.58 $
154   Total SiPs Cost 2d: 20.40 $
```

PRINCETON UNIVERSITY

Reading a Simulator Report

The **simulation logs** will be saved in the sim_logs folder. Particularly, **the basic experiment you ran with run_app.sh** creates a log called DATA-Kron16--16-X-16--BA16-A4.log

- This log contains first the logs of the dataset being loaded, Kron16, aka, RMAT-16 (containing 2^16 vertices or non-zero elems)
- Later, it shows information about the manycore configuration (16x16 grid with a single chiplet of 16x16 tiles, 1 PU/tile, etc.
- And its estimated cost
- Then, it logs **statistics and counters about the simulation frame by frame** (length configurable through set_conf).

```
ACollision In Avg:43, SD:17, Max:145, Min:12
ACollision Out Avg:14, SD:12, Max:47, Min:2
ACollision End Avg:0, SD:1, Max:16, Min:0
ATask1 Avg:26, SD:2, Max:31, Min:9
ATask2 Avg:43, SD:7, Max:49, Min:0
ATask3 Avg:29, SD:6, Max:80, Min:16
ACore Active Avg:97, SD:6, Max:100, Min:48
ARouter Active Avg:84, SD:3, Max:90, Min:75
T1 invocations: 7351
T2 invocations: 46666
T3 invocations: 731103
---Energy---
Leakage Energy (nJ): 364
PU Dyn E (nJ): 52576
Mem Dyn E (nJ): 180199
Route Dyn E (nJ): 48374
Dynamic Energy (nJ): 281149
Ratio of Dyn/Leak: 772
Total Energy (nJ): 281513
Total Energy (nJ) 3D: 315766
Cores Energy (nJ): 52591
Route Energy (nJ): 48525
Route Energy (nJ) 3D: 82778
Mem Energy   (nJ): 180398
Mem Energy   (nJ) 3D: 180398
Avg. Power (mW): 13576.05
Power Density (mW/mm2): 49.80
```

PRINCETON UNIVERSITY

Reading a Simulator Report

From **exp/run_app.sh**

```
31    th=2 # Number of threads
32    assert_mode=1 # Enable assertions within the simulator code
33    verbose=2 # Level 1: print basic stats on each sample, Level 2: print full stats on each iteration, etc.
```

Verbose>=1

```
8294    ACollision In Avg:43, SD:17, Max:145, Min:12
8295    ACollision Out Avg:14, SD:12, Max:47, Min:2
8296    ACollision End Avg:0, SD:1, Max:16, Min:0
8297    ATask1 Avg:26, SD:2, Max:31, Min:9
8298    ATask2 Avg:43, SD:7, Max:49, Min:0
8299    ATask3 Avg:29, SD:6, Max:80, Min:16
8300    ACore Active Avg:97, SD:6, Max:100, Min:48
8301    ARouter Active Avg:84, SD:3, Max:90, Min:75
8302    T1 invocations: 7351
8303    T2 invocations: 46666
8304    T3 invocations: 731103
```

Verbose>=2

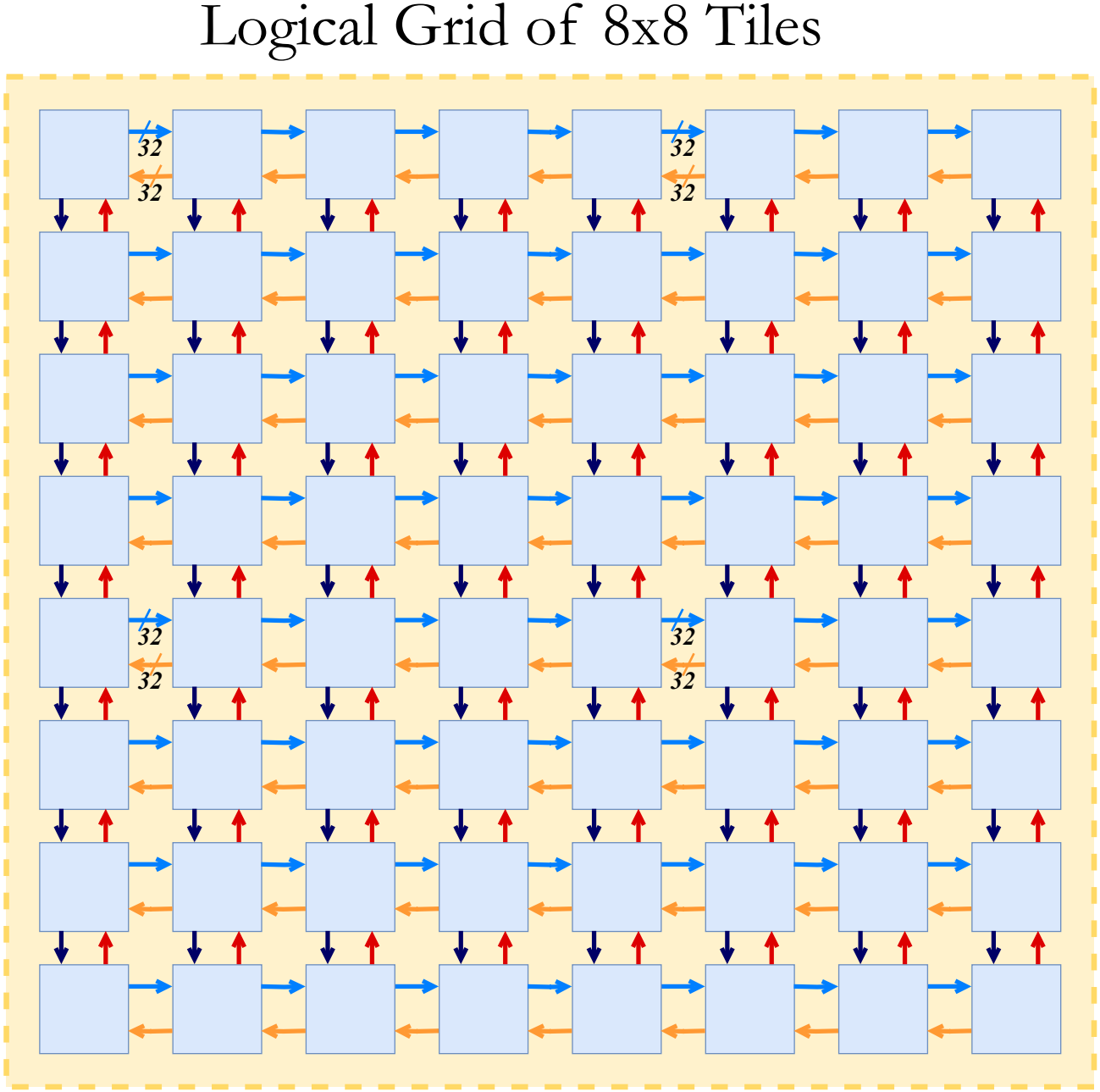
```
8258    AMCore Active
8259    100 100 98  100 100 100 100 93  100 100 96  100 87  95  93  100
8260    100 92  95  100 100 100 93  100 99  100 98  94  100 100 93  96
8261    89  100 88  47  98  100 100 100 100 96  100 100 100 66  83  100
8262    92  100 100 97  100 100 99  95  92  100 96  100 100 100 100 99
8263    100 100 99  100 100 100 92  100 100 100 94  87  100 61  100 100
8264    100 96  100 98  99  100 100 95  88  98  100 94  100 100 100 95
8265    100 100 100 98  97  100 100 93  100 100 100 100 94  100 100 100
8266    100 100 94  91  100 100 100 100 90  100 100 98  100 100 100 97
8267    100 100 100 100 94  99  99  100 100 82  96  100 100 100 100 100
8268    100 100 100 96  100 100 100 100 83  100 100 100 100 100 100 100
8269    100 97  98  96  100 100 100 95  100 91  100 95  88  100 100 96
8270    87  95  100 100 99  100 96  100 94  100 100 96  100 98  100 100
8271    100 90  100 100 100 100 99  100 98  100 94  100 64  100 100 100
8272    100 100 100 82  90  100 100 100 100 100 98  100 93  100 100 100
8273    100 74  100 100 89  84  100 97  100 100 100 100 91  100 96  100
8274    100 100 99  93  100 100 94  100 100 100 93  91  100 100 100 100
```

PRINCETON UNIVERSITY

Simulation Frame

PU and NoC simulation threads keep track of many **performance counters**

- Every X thousand cycles (a **frame**) the PU threads output stats to a Log
- Aggregated and per-frame stats



PRINCETON UNIVERSITY

Simulation

From **exp/run_app.sh**

```
31    th=2 # Number of threads
32    assert_mode=1 # Enable assertions with
33    verbose=2 # Level 1: print basic stats
```

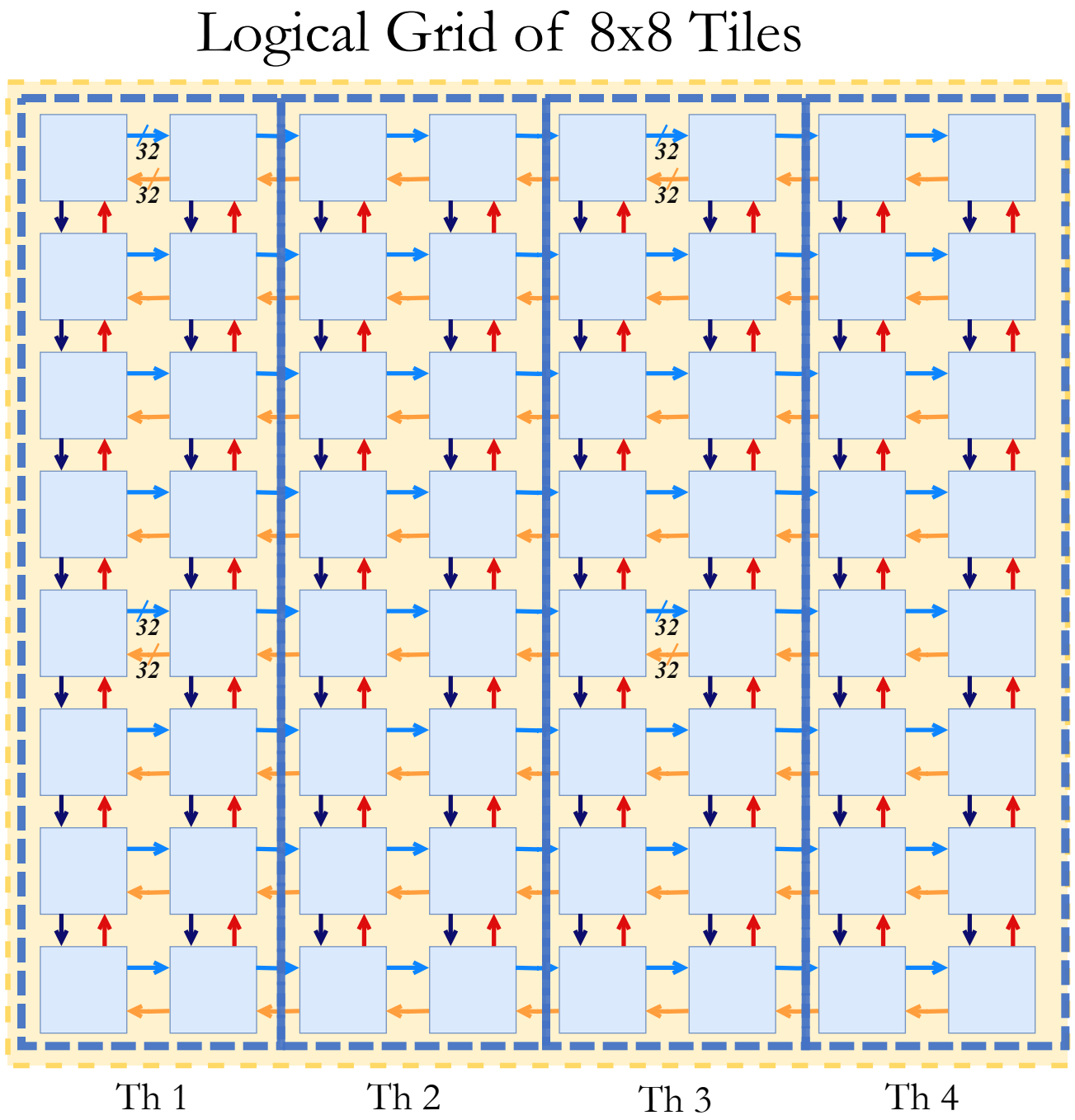
pow2 # threads

Parallelization across columns

- A pair of PU and NoC threads per grid slice
- 4 slices → 8 host threads in total!

PU and NoC threads synchronize based on message timestamps

- NoC and PUs in may use different clock frequency of the DUT



PRINCETON UNIVERSITY

Outline First Half

Introduction
Motivation

Target Architectures covered by MuchiSim

Simulator Infrastructure

Downloading and Basic Experiment

Reading a Simulator Report

Parallel Simulation

Simulator Configurations

System Hardware Configurations

Energy, Area & Cost Parameters

Sweeping Through Parameter Values

Case Study: Cerebras Wafer Scale Engine

Application-Hardware Interactions: Programming Model

Task-based Programs

Adding Applications to MuchiSim

Application Suite

Datasets Included in the Repo

PRINCETON UNIVERSITY

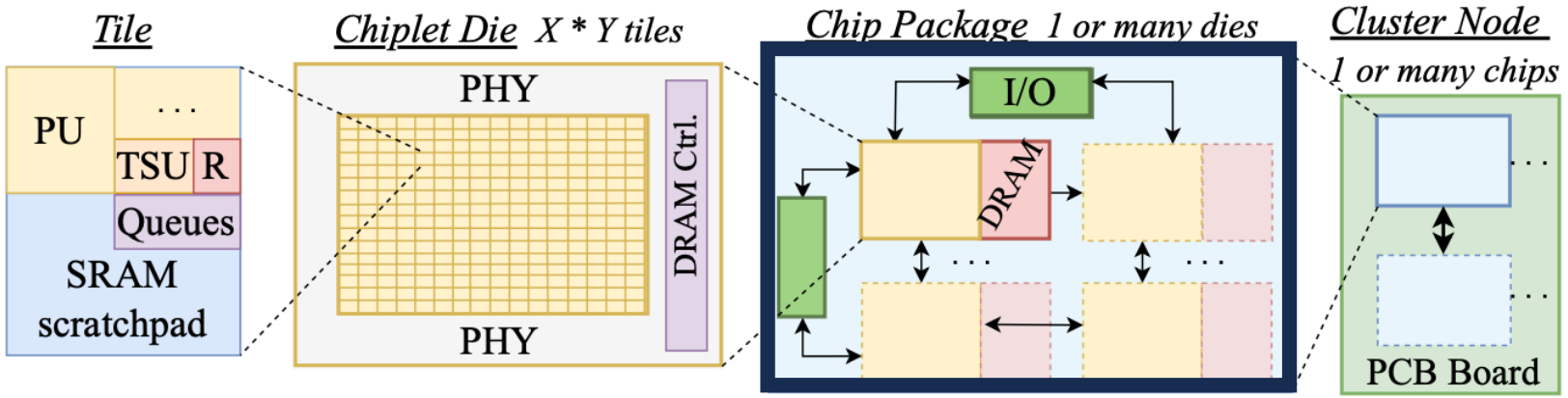
Balancing Memory, Compute, and Network resources

Chiplet-level design choices

- SRAM and PUs-per-tile & Tiles-per-chiplet
- NoC topology and width & extra express channels.

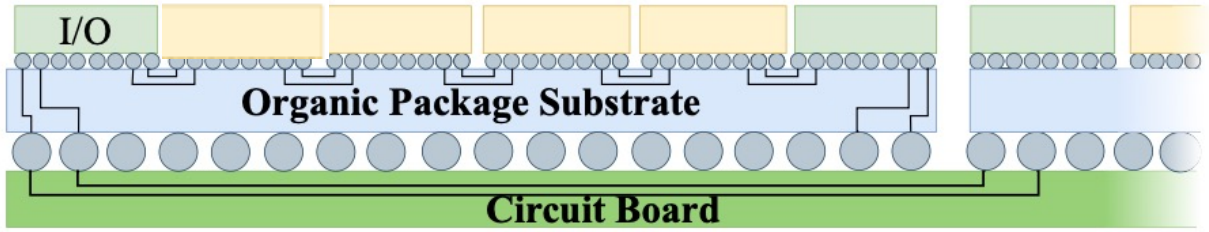
Package-level choices

- Memory hierarchy (levels and capacity)
- Chip package size and off-chip I/O bandwidth

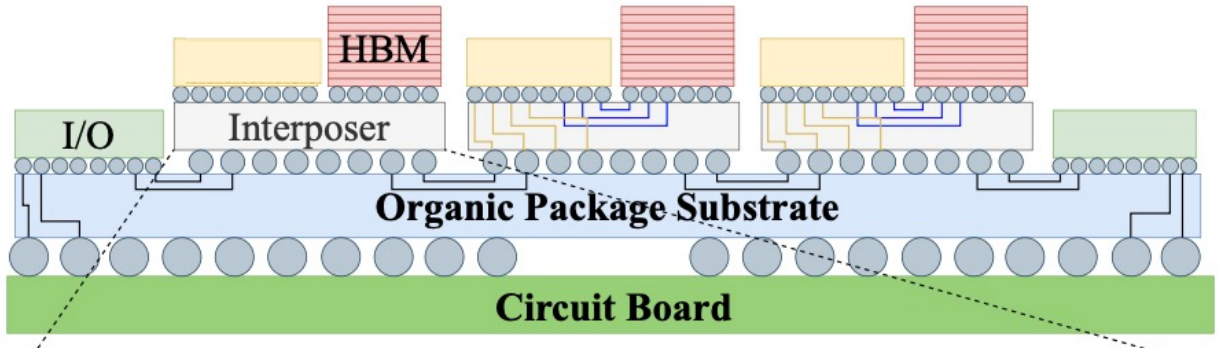


PRINCETON UNIVERSITY

Configurable Memory Hierarchy



a. **Distributed SRAM, as main-memory**
   - For a given dataset, SRAM-only will need more chiplets to store the dataset



b. **L1 cache (SRAM) + main-mem (HBM)**
   - No coherence in our current models

Image from [1]

[1] M Orenes-Vera, E Tureci, D Wentzlaff, and M Martonosi "DCRA: A Distributed Chiplet-based Reconfigurable Architecture for Irregular Applications"

PRINCETON UNIVERSITY

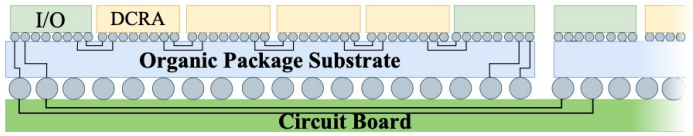
Energy/BW/Latency Parameters for Memory & Links

TABLE I
DEFAULT VALUES OF ENERGY (E), BANDWIDTH, LATENCY, AND AREA PARAMETERS OF LINKS AND MEMORY DEVICES MODELED IN MUCHISIM.

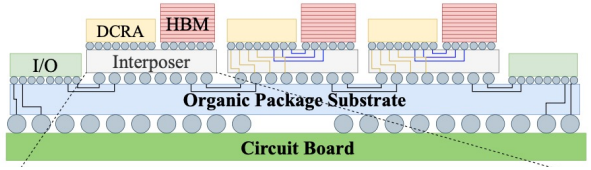
Memory Model Parameters	**Values**
SRAM Density	3.5 MB/mm^2 [99]
SRAM R/W Latency & E.	0.82 ns & 0.18 / 0.28 pJ/bit [99]
Cache Tag Read & cmp. E.	6.3 pJ [99], [101]
HBM2E 4-high Density	8GB on 110mm^2 (75 MB/mm^2) [46]
Mem.Channels & Bandwidth	8 x 64GB/s [46]
Mem.Ctrl-to-HBM Latency & E.	50 ns & 3.7 pJ/bit [40], [74]
Bitline Refresh Period & E.	32 ms & 0.22 pJ/bit [29], [87]
Wire & Link Model Parameters	**Values**
MCM PHY Areal Density	690 Gbits/mm^2 [7]
MCM PHY Beachfront Density	880 Gbits/mm [7]
Si. Interposer PHY Areal Density	1070 Gbits/mm^2 [7]
Si. Interposer PHY Beachfront Density	1780 Gbits/mm [7]
Die-to-Die Link Latency & E.	4 ns & 0.55 pJ/bit (<25 mm) [64]
NoC Wire Latency & E.	50 ps/mm & 0.15 pJ/bit/mm [42]
NoC Router Latency & E.	500 ps & 0.1 pJ/bit
I/O Die RX-TX Latency	20 ns [84]
Off-Package Link E.	1.17 pJ/bit (upto 80mm) [98]

Default memory and communication link parameters

- Multi-Chip-Module (MCM) or Silicon interposers



MCM



Silicon interposer

Other parameters for PU energy in the repo

- Energy scales with voltage → Voltage scales with operating frequency and transistor node (Default parameters with 0.7 V, 1 GHz, 7nm)
- Supports different frequency for NoC and PUs

github.com/PrincetonUniversity/MuchiSim

PRINCETON UNIVERSITY

Area & Cost Parameters

Area parameters for in-order PUs, supported NoC topologies, and memories

- Area scaling with peak design frequency

Cost

- **Die:** Wafer cost by the number of good dies → calculated using parameterizable number of defects per *mm*2 and edge loss, using Murphy's model.
- **Substrate**, **Interposer** and **yield** cost as a % of the die area.
- **HBM** cost based on $/GB.

MuchiSim allows post-processing a given experiment (without re-simulating) to **re-calculate the energy and cost with different model parameters.**

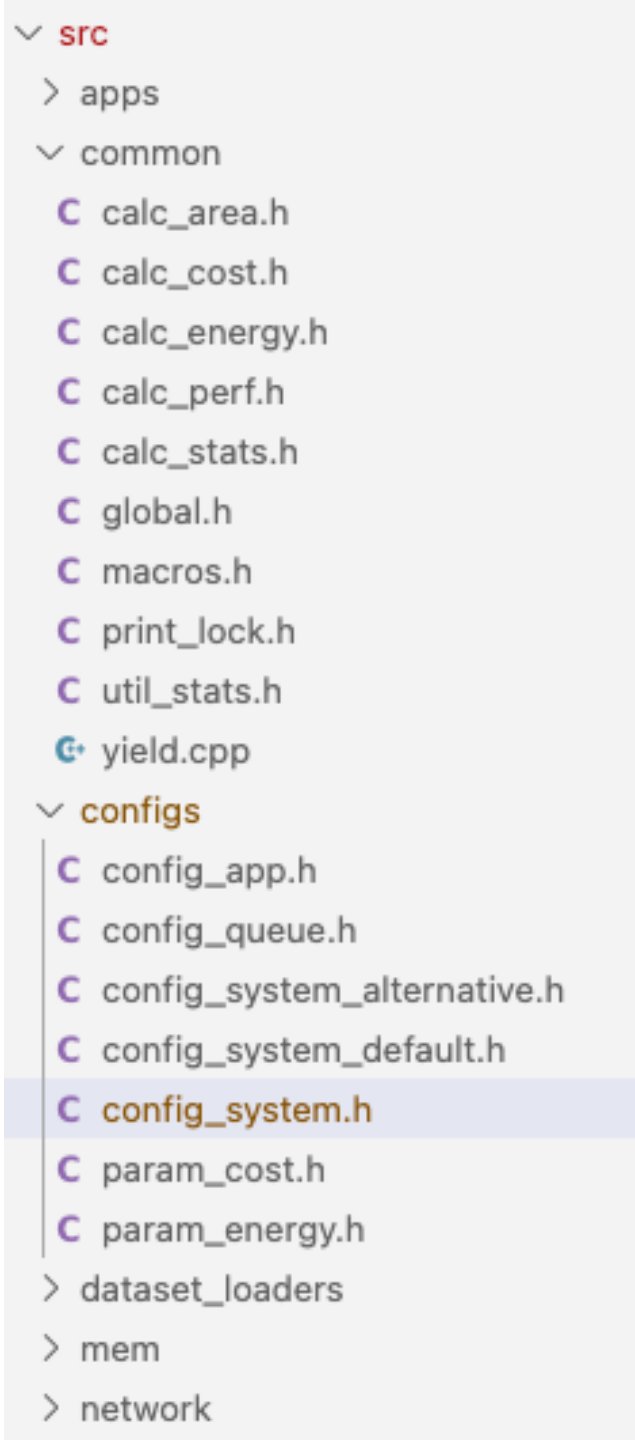
PRINCETON UNIVERSITY

Setting Simulator Configurations

The `set_conf` function allows to programmatically change parameters defined in the config and parameter files (inside `src/config`)

The `reprocess.sh` script re-process a simulation with different energy and cost parameters, to evaluate different scenarios, e.g., calculate different tradeoffs in performance/$ or performance/Watt

```
# Change the cost of HBM memory to 5 USD/GiB
set_conf "param_cost" "hbm_usd_per_gib" 5
exp/reprocess.sh # Default Kron16--16-X-16--BA16-A4


# Change back to the original value
set_conf "param_cost" "hbm_usd_per_gib" 7.5
exp/reprocess.sh
```



PRINCETON UNIVERSITY

Granularity of the Processing Tile

Changing the number of PUs per tile while keeping the total number of PUs and bisection BW constant

```
exp/run_exp_granularity.sh
4 0 2 32 Kron16
```

Run SPMV (app=4), and 3 cases

- Configuration & run script prepares cases 0-2 with different configurations of tiles per chiplet, PUs/per tile (SMT) and network BW
- Later we'll run this and visualize the simulation logs!

```
54  let torus=1
55  #0 Torus-32b, SMT=1
56  let sram_memory=256*512
57  set_conf "config_system" "smt_per_tile" 1
58  let chiplet_w=$grid_w/4
59  options="-n ${exp}0 -t $th $prefix -m $grid_w -u 0 -o $torus -c $chiplet_w -w $sram_memory -k $grid_w"
60  run 0
61
62
63  #1 Torus-64b, SMT=4
64  let sram_memory=$sram_memory*4
65  let grid_w=$grid_w/2
66  let chiplet_w=$grid_w/4
67  set_conf "config_system" "smt_per_tile" 4
68  options="-n ${exp}1 -t $th $prefix -m $grid_w -u 1 -o $torus -c $chiplet_w  -w $sram_memory -k $grid_w"
69  run 1
70
71
72  #2 Torus-64b 2GHZ, SMT=16
73  let sram_memory=$sram_memory*4
74  let grid_w=$grid_w/2
75  let chiplet_w=$grid_w/4
76
77  set_conf "param_energy" "noc_freq" 2.0
78  set_conf "config_system" "smt_per_tile" 16
79  options="-n ${exp}2 -t $th $prefix -m $grid_w -u 1 -o $torus -c $chiplet_w  -w $sram_memory -k $grid_w"
80  run 2
```

PRINCETON UNIVERSITY

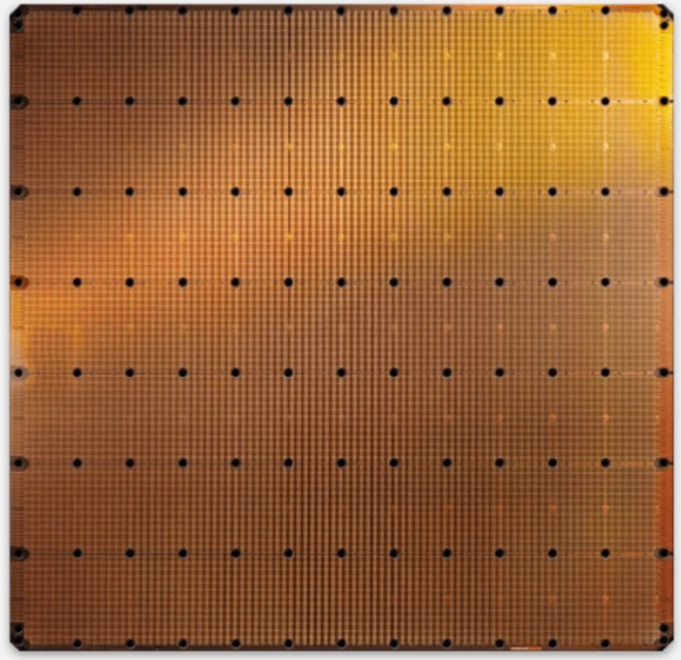
MuchiSim to simulate the Cerebras Wafer-Scale Engine

- Tiles contain a PU, 48KB scratchpad and a router.
- 32-bit 2D mesh NoC
- 3D-FFT of n^3 elements over n^2 PEs for n=32,..,512
  - Using the published implementation[1]

The reported WSE-2 runtimes[1] are ~1.2x of the simulated ones

- Stable across scaling range, 32x32 to 512x512 (1/4 million tiles)

DUT area calculated by MuchiSim is 1.08x of the WSE-2 area.



Cerebras WSE-2
46,225mm² Silicon
2.6 Trillion transistors

	WSE-2
Chip Size	46,225 mm2
Cores	850,000
On-chip memory	40 Gigabytes
Memory bandwidth	20 Petabytes/sec
Fabric bandwidth	220 Petabits/sec

cerebras.net

[1] M. Orenes-Vera, I. Sharapov, R. Schreiber, M. Jacquelin, et al. "Wafer-Scale Fast Fourier Transforms". ICS'23

PRINCETON UNIVERSITY

Outline First Half

Introduction
Motivation

Target Architectures covered by MuchiSim

Simulator Infrastructure

- Downloading and Basic Experiment
- Reading a Simulator Report
- Parallel Simulation

Simulator Configurations

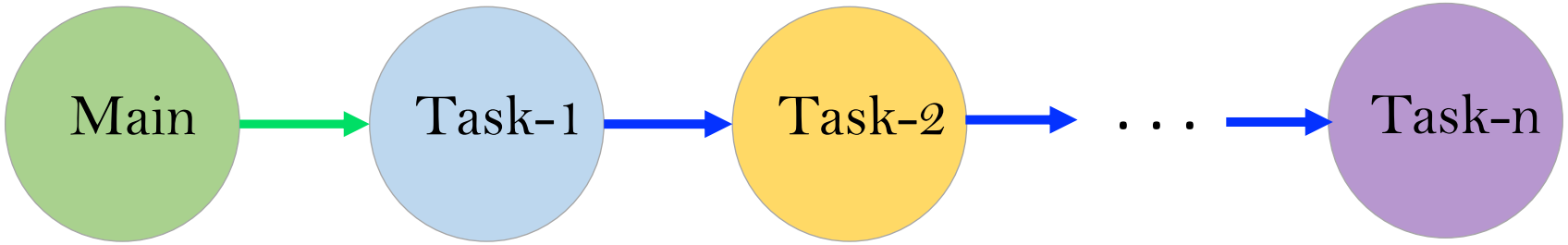
- System Hardware Configurations
- Energy, Area & Cost Parameters
- Sweeping Through Parameter Values
- Case Study: Cerebras Wafer Scale Engine

Application-Hardware Interactions: Programming Model

- Task-based Programs
- Adding Applications to MuchiSim
- Application Suite
- Datasets Included in the Repo

PRINCETON UNIVERSITY

MuchiSim Enables Exploration of Novel Task-based Programming Models



A program can be converted into a series tasks where each task invokes the next preserving program order.

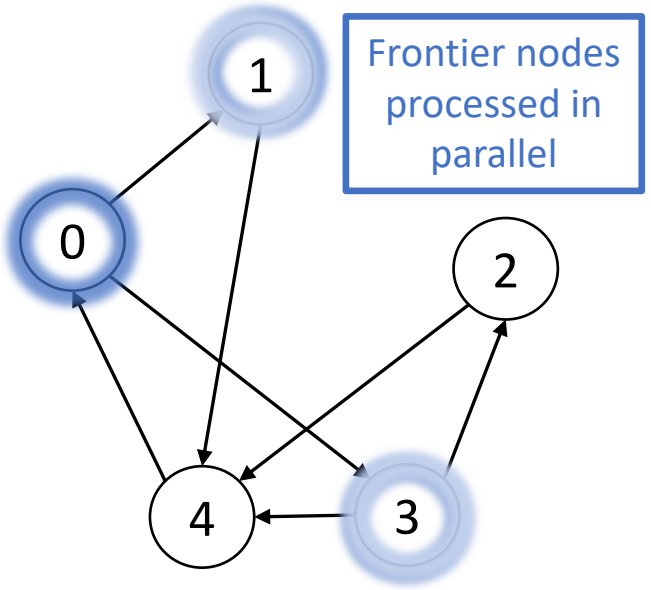
Mapping of the location for the execution of tasks can be done according to the desired parallelization scheme.
e.g. Near Memory Processing parallelization schemes utilize a data-local execution based parallelization.

Since the set of many core architectures MuchiSim targets do not employ cache-coherence, tasks must be written in a way that does not require coherence at cache level.

PRINCETON UNIVERSITY

Transforming a Program to a Task-based Program: Iterative, Frontier-based Graph Applications

```
1 while frontier not empty:
2     for node in frontier:
3       val = process_node(node)
4       for neib in G.neighbors(node):
5         update = update_neib(node_vals,val,neib)
6         if(add_to_frontier(update)):
7           new_frontier.push(neib)
```



PRINCETON UNIVERSITY

Transforming a Program to a Task-based Program: Task Generation

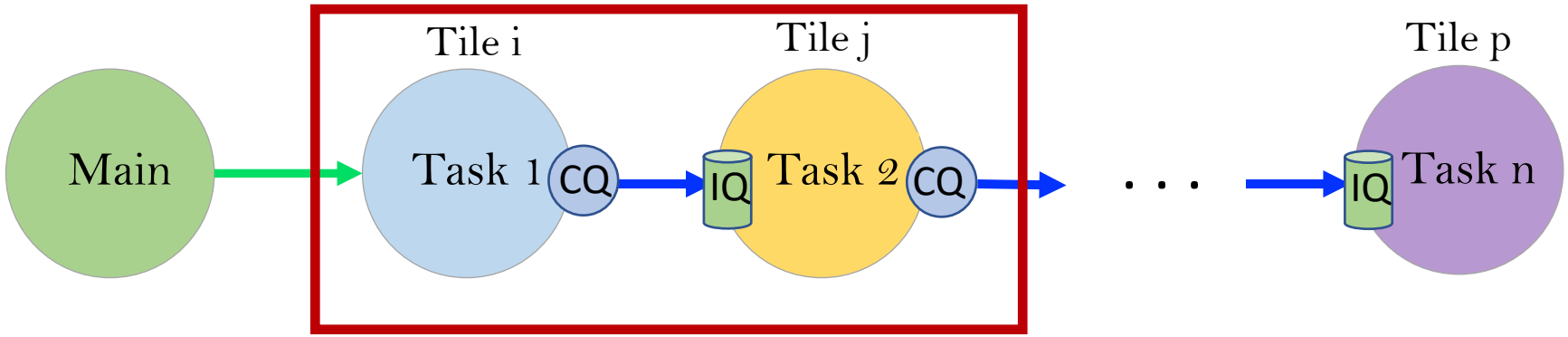
```
while frontier not empty:
  for node in frontier:
    val = process_node(node)
    for neib in G.neighbors(node):
      update = update_neib(node_vals,val,neib)
      if(add_to_frontier(update)):
        new_frontier.push(neib)
```

```
Task T1 (v):
    CQ = PTR[v]
    CQ = PTR[v+1]
    CQ = DIST[v]

Task T2 (neigh_begin, neigh_end, node_dist):
    for i in range(neigh_begin, neigh_end):
      CQ = EDGES[i].neigh_idx
      CQ = node_dist + EDGES[i].val

Task T3 (neighbor, new_dist):
    curr_dist = DIST[neighbor]
    if (new_dist < curr_dist):
      DIST[neighbor] = new_dist
      frontier.push(neighbor)

Task T4:
    CQ = frontier.pop()
```

PRINCETON UNIVERSITY

MuchiSim Enables Exploration of Novel Task-based Programming Models: Variations

1-to-1: One Task invoking a single next Task

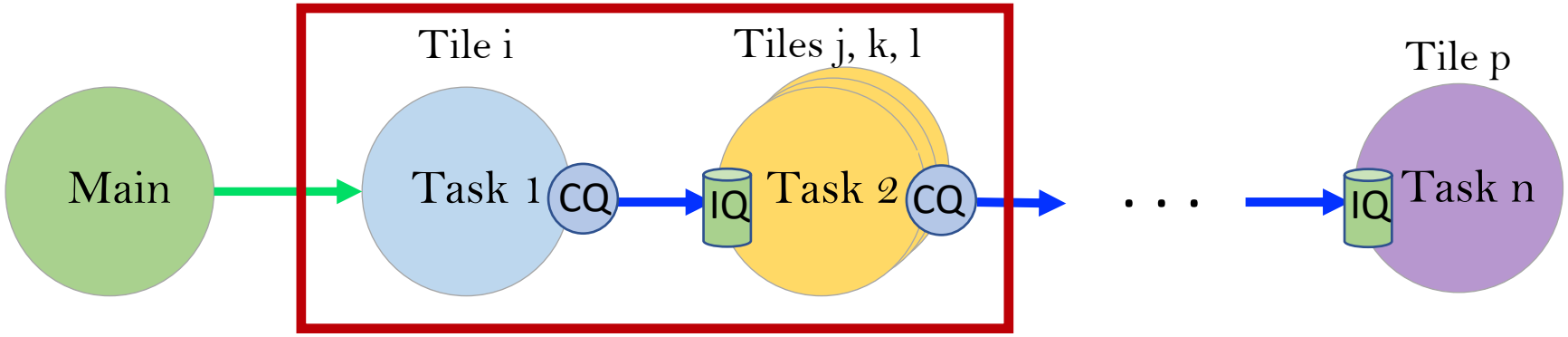



```
while frontier not empty:
 for node in frontier:
   val = process_node(node)
    for neib in G.neighbors(node):
      update = update_neib(node_vals,val,neib)
       if(add_to_frontier(update)):
         new_frontier.push(neib)
```

PRINCETON UNIVERSITY

MuchiSim Enables Exploration of Novel Task-based Programming Models: Variations

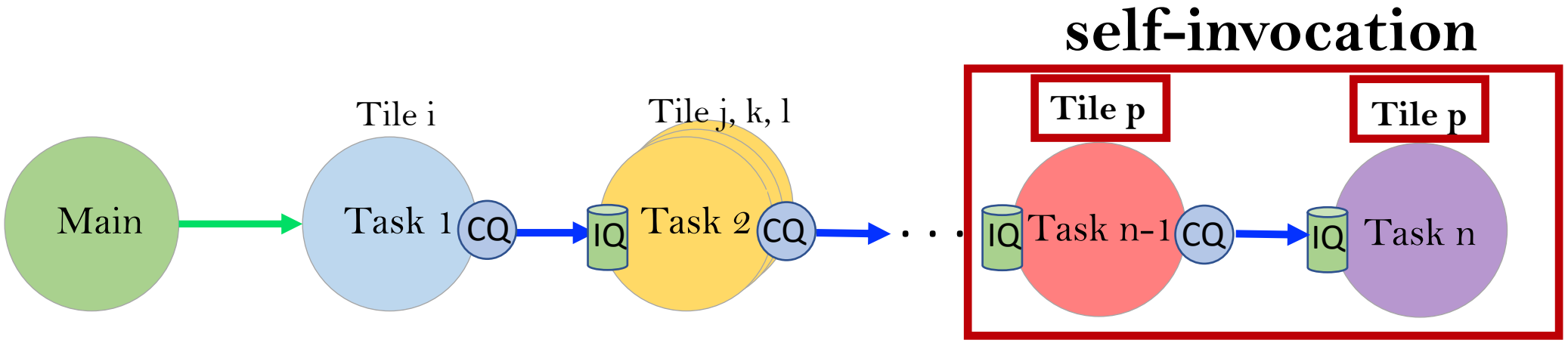
1-to-many: One Task can invoke many of the next Task




```
while frontier not empty:
 for node in frontier:
   val = process_node(node)
    for neib in G.neighbors(node):
      update = update_neib(node_vals,val,neib)
       if(add_to_frontier(update)):
         new_frontier.push(neib)
```

PRINCETON UNIVERSITY

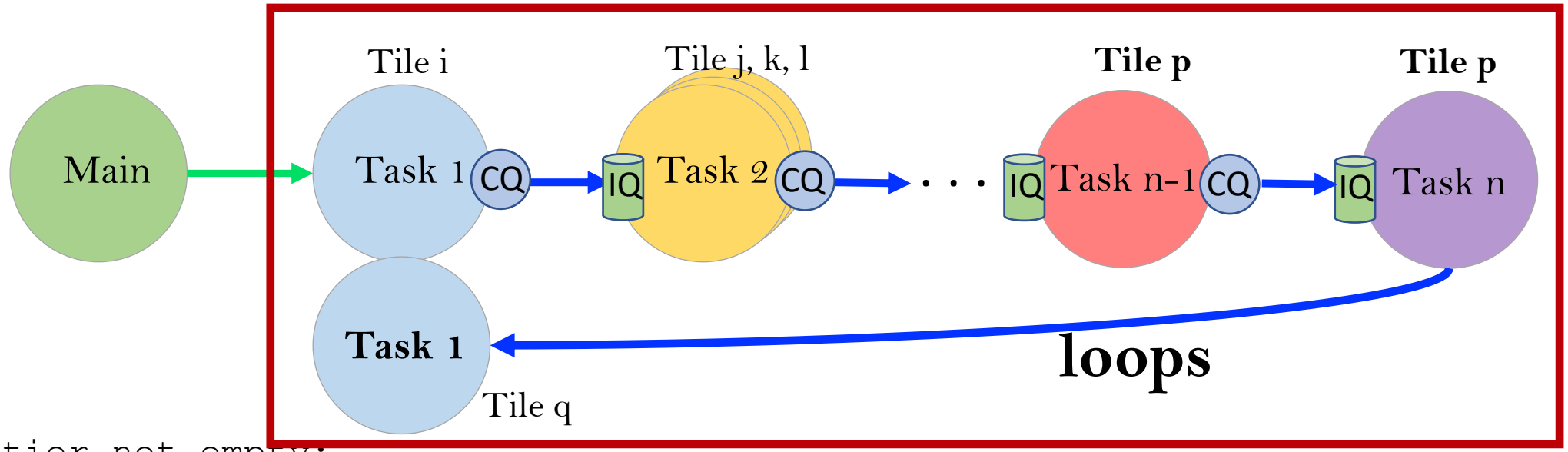
MuchiSim Enables Exploration of Novel Task-based Programming Models: Variations




```
while frontier not empty:
 for node in frontier:
   val = process_node(node)
    for neib in G.neighbors(node):
      update = update_neib(node_vals,val,neib)
       if(add_to_frontier(update)):
         new_frontier.push(neib)
```

PRINCETON UNIVERSITY

MuchiSim Enables Exploration of Novel Task-based Programming Models: Variations




```
while frontier not empty:
 for node in frontier:
   val = process_node(node)
   for neib in G.neighbors(node):
     update = update_neib(node_vals,val,neib)
     if(add_to_frontier(update)):
       new_frontier.push(neib)
```

Application Suite

4 graph algorithms, 2 sparse linear algebra, and 2 HPC kernels

- ***Breadth-First Search (BFS)*** determines the number of hops from a root vertex to all vertices reachable from it.
- ***Single-Source Shortest Path (SSSP)*** finds the shortest path from the root to each reachable vertex.
- ***PageRank*** ranks websites based on the potential flow of users to each page.
- ***Weakly Connected Components (WCC)*** finds and labels each set of vertices reachable from one
- ***Sparse Matrix-Vector Multiplication (SPMV)*** multiplies a square sparse matrix with a dense vector.
- ***Sparse Matrix-Matrix Multiplication (SPMM)*** multiplies a square sparse matrix with a dense matrix and stores the result in a dense matrix.
- ***3D Fast Fourier Transform (FFT)*** computes the Fourier Transformation of a 3D tensor.
- ***Histogram*** counts the values that fall within a series of intervals.

PRINCETON UNIVERSITY

Annotation of Task-based Programs For MuchiSim:

```
Task T3 (neighbor, new_dist):
    curr_dist = DIST[neigh_id]
    if (new_dist < curr_dist):
        return_array[neighbor] = new_dist
        frontier.push(neighbor)
```

- microarchitectural timing for basic blocks
- count stores, loads and ops for energy

Read the inputs of the Task from input queues

```
1  int task3_kernel(int tX, int tY, u_int64_t timer){
2    Msg msg = IQ(2).dequeue();
3    u_int32_t neighbor = task3_dequeue(msg.data);
4    u_int32_t new_dist =  IQ(2).dequeue().data;
5
6    int cycles = check_dcache(tX,tY,ret,neighbor,timer, msg.time);
7    int curr_dist= return_array[neighbor];
8    cycles +=2; //comp + beq
9    bool cond = (new_dist < curr_dist); flop(1);
10   if (cond){
11     return_array[neighbor] = new_dist; //Surely it's a hit in the cache
12     cycles+=1;store(1);
13     cycles+=add_to_frontier(tX, tY, neighbor,timer) + 1; //ADD to Frontier + Store
14   }
15   return cycles;
16 }
```

What else can be mapped to this programming model?

- Multiple PUs **accessing shared data from the DRAM** if it's not modified

Simulator Features: Details and Plans

- No Operating System support yet
  - MuchiSim focuses on manycore systems that are programmed as accelerators. PUs access memory without virtualization

- No hardware cache coherence
  - Currently not needed for the programming models we evaluated
  - Could be implemented via simulator or software extensions

- Few in-core microarchitecture details.
  - Computation modeled at instr-level, via application code annotations

Datasets in Simulator Repo

- Work with the 8 provided benchmarks
- 6 sizes of the synthetic RMAT graphs—standard on the Graph500 list—RMAT-16, 21, 22, 25, 26, and 27.
- Named after their scale. E.g., RMAT-26 contains 2^26, i.e., 67M Vertices (V) and 1.3B Edges (E) ~11 GB

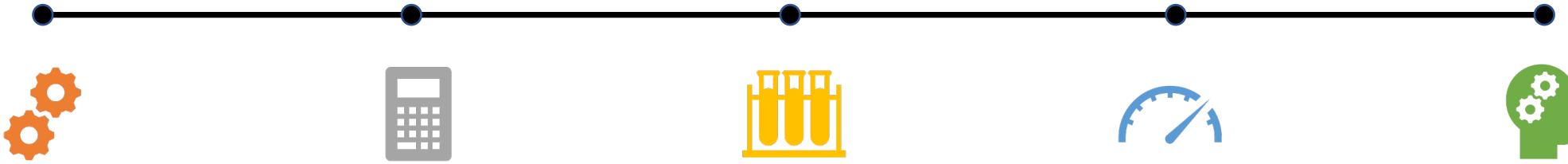
And 4 real-world graphs:

- Wikipedia (V=4.2M, E=101M)
- LiveJournal (V=5.3M, E=79M)
- Amazon (V=262K, E=1.2M)
- Twitter (V=81K, E=2.4M)

For SPMV and SPMM, the graphs are seen as a square sparse matrix with V rows & columns and E non-zero elems.

‖ The graphs (as sparse matrices) are stored in the Compressed Sparse Row (CSR) format

Outline Second Half



Visualization Tools

Case Study: Regional Reductions

Visualization Tools

Case Study: Memory-Compute Ratio

End-to-end Experiment

Simulation Speed & Scalability

Conclusions & Future Work

Work that users could do to explore System design and application mapping

Future work on the simulator itself

Instructions for Visualization Tools

`muchiSim/tutorial/2.VISUALIZATION.md`

PRINCETON UNIVERSITY

Visualization Tools

1. Plotting **per-PE performance counters** or **stats for every execution frame**
   - **Router** port collisions and utilization, end-point contention
   - **PU** utilization
   - **Cache** hit-rate and memory-controller requests and average latency.

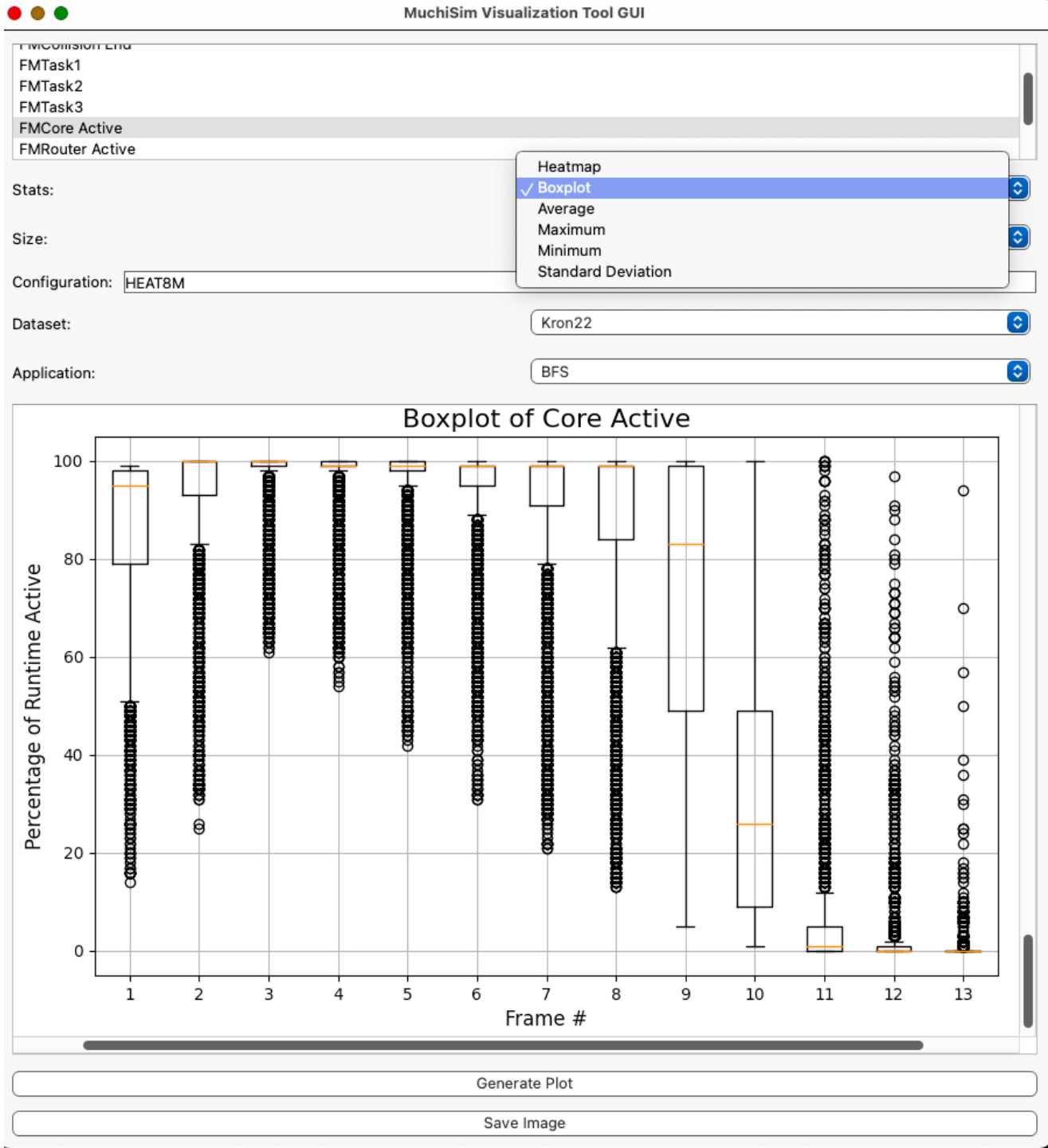
2. Plotting aggregated execution metrics for combinations of HW **configurations**, and **applications** and **datasets.**
   - Runtime, throughput (FLOPS), energy, cost.
   - Network traffic, cache hit-rate, arithmetic intensity, etc.

PRINCETON UNIVERSITY

GUI

visualization.py

1. Performance metric, e.g., boxplot, mean
2. DUT grid size, e.g., 64x64
3. Configuration experiment name
4. Dataset, e.g., RMAT-22 (Kronecker)
5. Application, e.g., BFS



PRINCETON UNIVERSITY

1.2. Creating the Python Environment

```
python3 --version
```

The visualization tools require Python3.6 or higher! If you don't have Python installed, you can use the Docker version (see setting up Docker). If you run within the Docker, do not run this section that creates a `venv` and installs the dependencies.

RUN THE FOLLOWING COMMANDS **ONLY IF RUNNING NATIVELY, NOT WITHIN DOCKER** AS DEPENDENCIES ARE ALREADY INSTALLED

```
python3 -m venv gui/env_viz
source gui/env_viz/bin/activate
pip install --upgrade pip && pip install PyQt5 matplotlib imageio
```

To test if the Python environment is correctly set up, run the following command:

```
python3 gui/visualization.py --h # if you see the help message, then the environment is correctly set up
```

www.github.com/PrincetonUniversity/muchiSim/blob/main/tutorial/**2.VISUALIZATION.md**

PRINCETON UNIVERSITY

Visualizing Execution Frame by Frame

```
visualization.py [-h] [--nogui] [-m METRIC] [-p PLOT] [-s SIZE] [-a APP] [-n NAME] [-d DATASET]

--nogui        Process directly without GUI

  -m METRIC       Metric name (default: FMCore Active)

  -p PLOT         Plot statistical metric (default: Average. Other options Boxplot, Heatmap)

  -s SIZE         Size parameter (default: 64)

  -a APP          App number (default: 2)

  -n NAME         Experiment name (default: )

  -d DATASET      Dataset name (default: Kron22)
```

PRINCETON UNIVERSITY

BFS on RMAT-22, 64x64 tiles, Mesh NoC

With atomic operations vs regional reductions[1]

Whether read-modify-write updates happen at a designated tile across the entire grid, or within a region, which later updates the globally designated owner

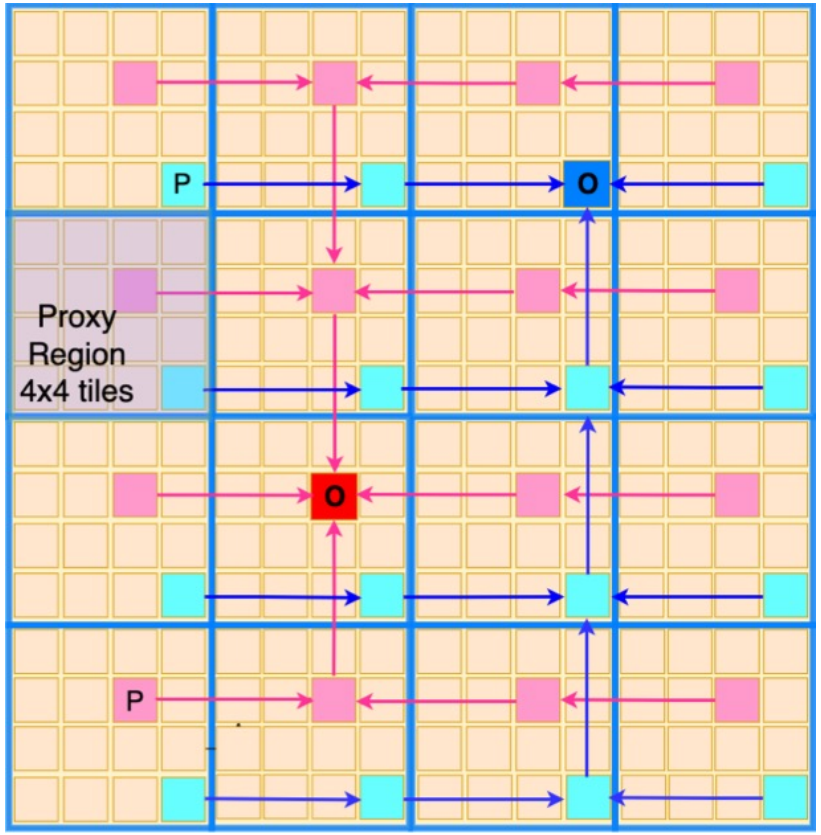
"Atomic" (every update has an owner)

```
python3 gui/visualization.py --nogui -e 1
-p Boxplot -n HEAT64M -m 'FMCore Active'
```

Regional reduction operations:

```
python3 gui/visualization.py --nogui -e 1
-p Boxplot -n HEAT8M -m 'FMCore Active'
```

Boxplots are generated at **muchisim/plots/images**

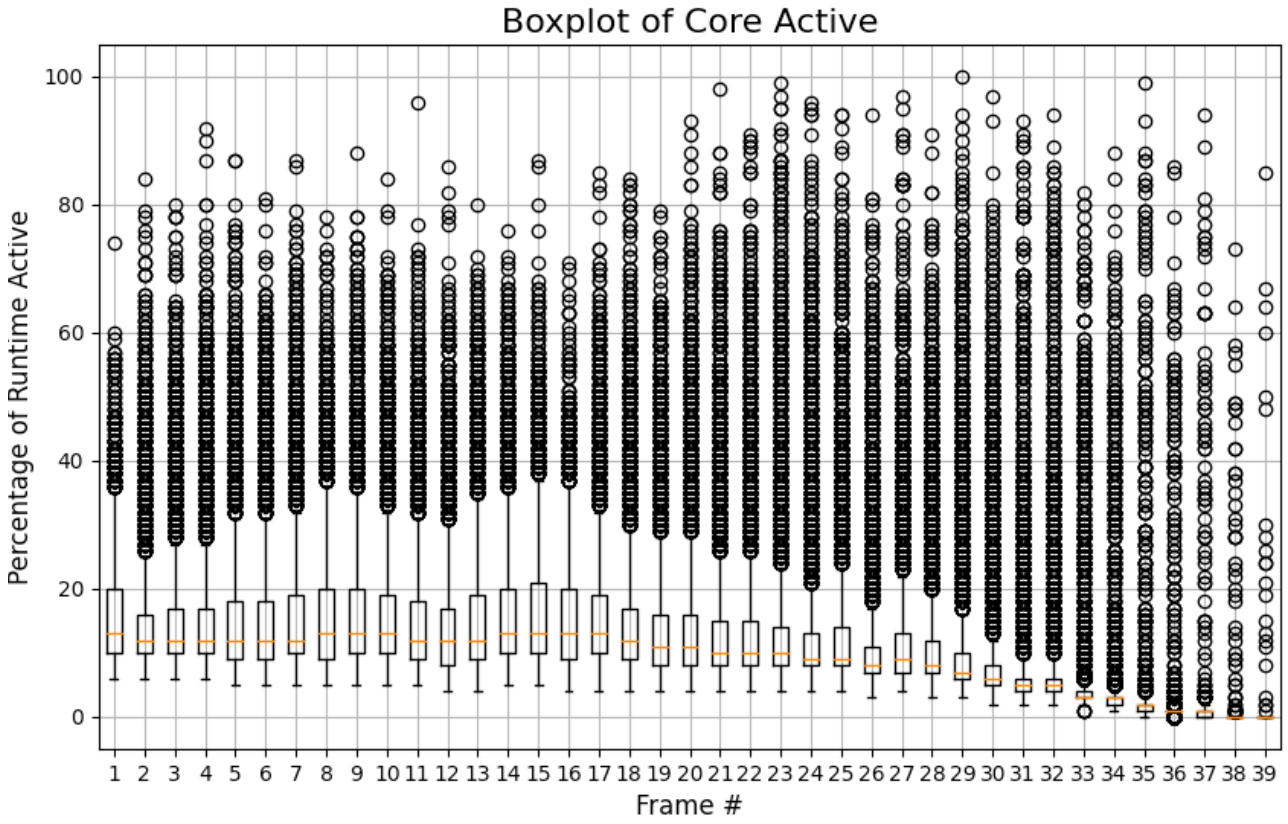


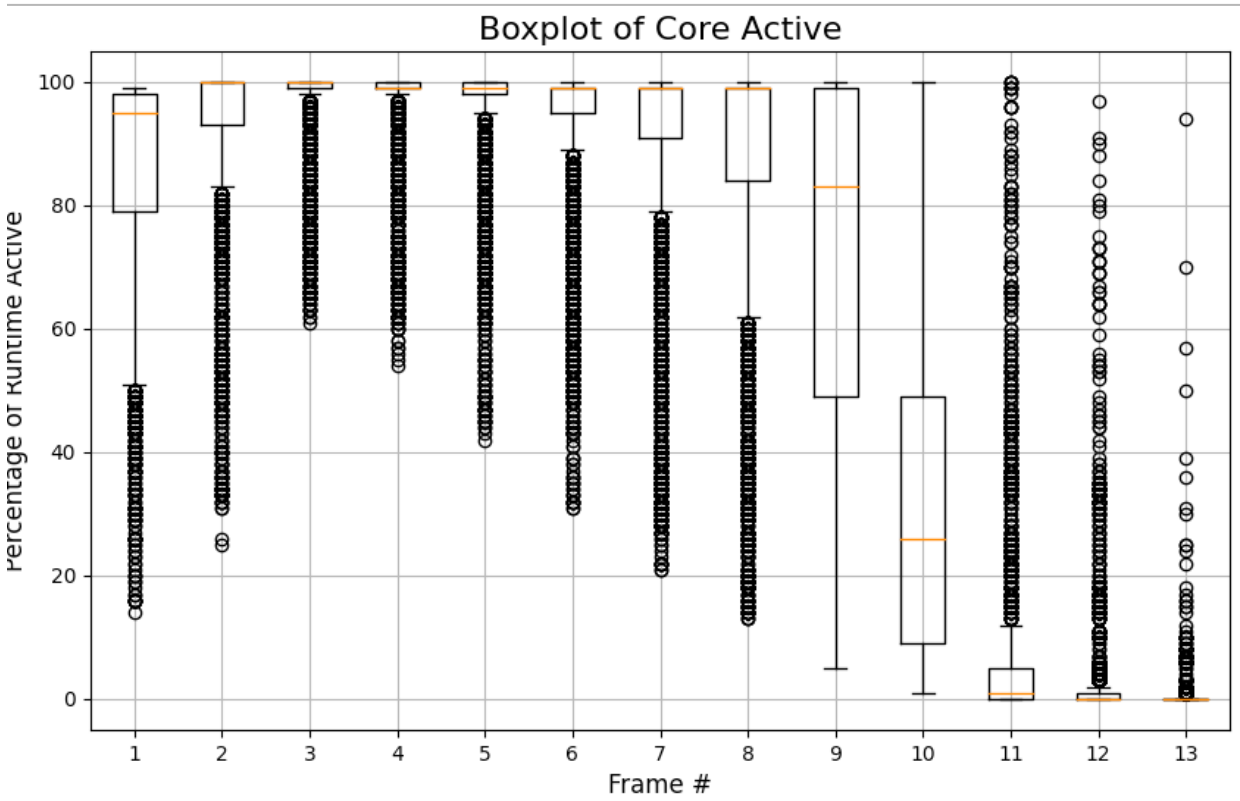
[1] M Orenes-Vera, E Tureci, D Wentzlaff, and M Martonosi "Tascade: Hardware Support for Atomic-free, Asynchronous and Efficient Reduction Trees"

PRINCETON UNIVERSITY

BFS on RMAT-22, 64x64 tiles, Mesh NoC

With atomic operations vs regional reductions[1]





Decrease in Frame # reflects the 3x runtime difference, since frame-rate is the same, 40 µs

[1] M Orenes-Vera, E Tureci, D Wentzlaff, and M Martonosi "Tascade: Hardware Support for Atomic-free, Asynchronous and Efficient Reduction Trees"

PRINCETON UNIVERSITY

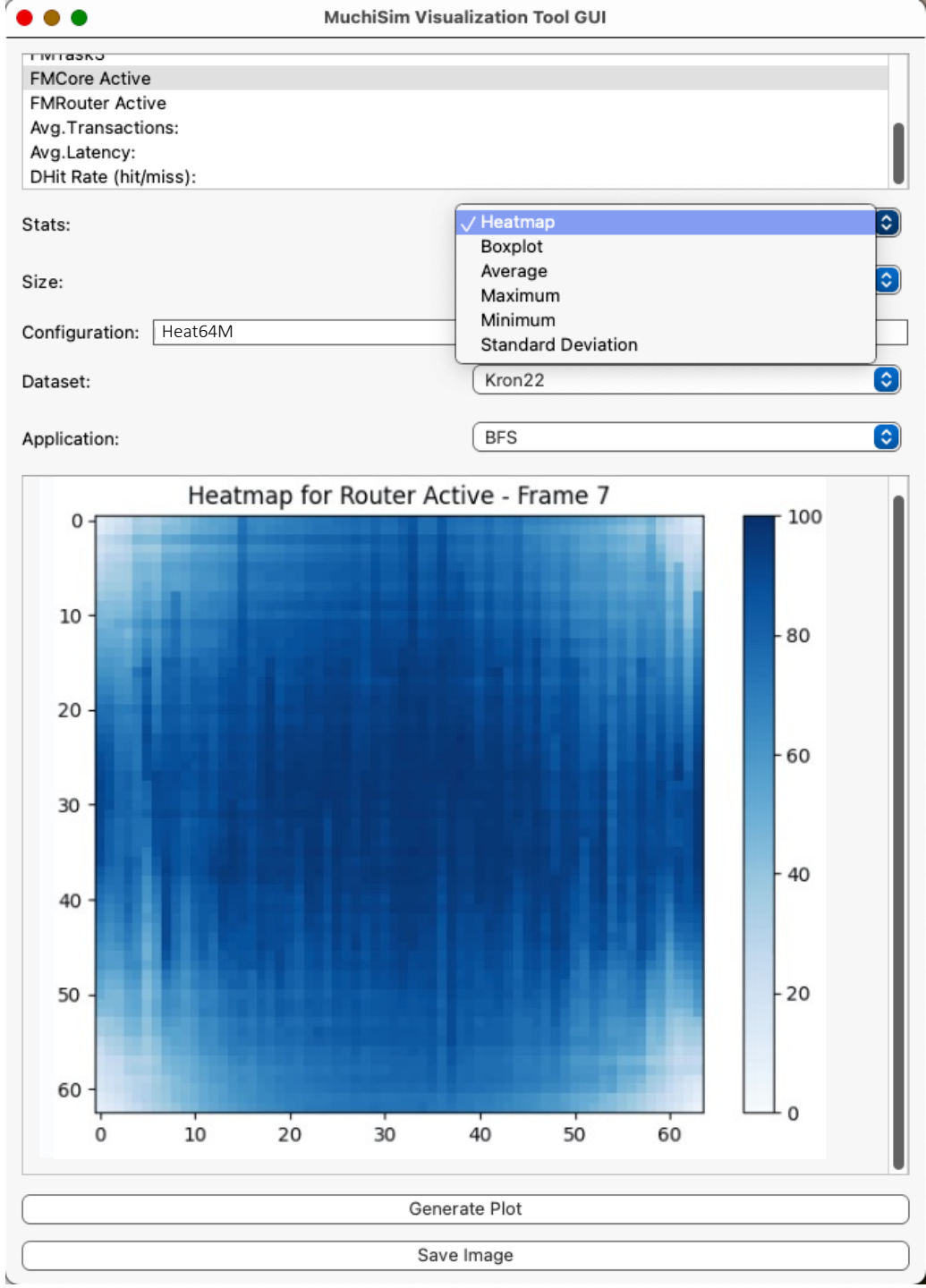
Animated Heatmaps

Creates a GIF with a heatmap image for every frame

- Each image shows % of frame time that the performance counter was active, e.g., PU utilization

```
python3 gui/visualization.py --nogui -e 1
-p Heatmap -n HEAT64M -m 'FMRouter Active'
```

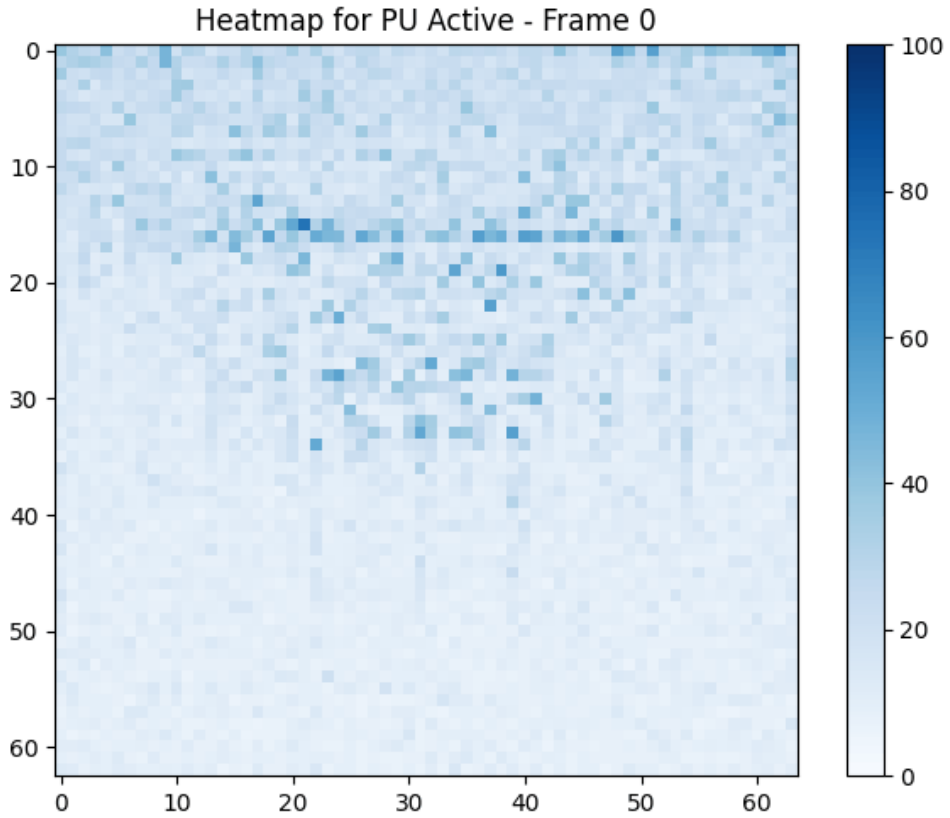
Generated at **muchisim/plots/animated_heatmaps** as subfolders starting with a timestamp

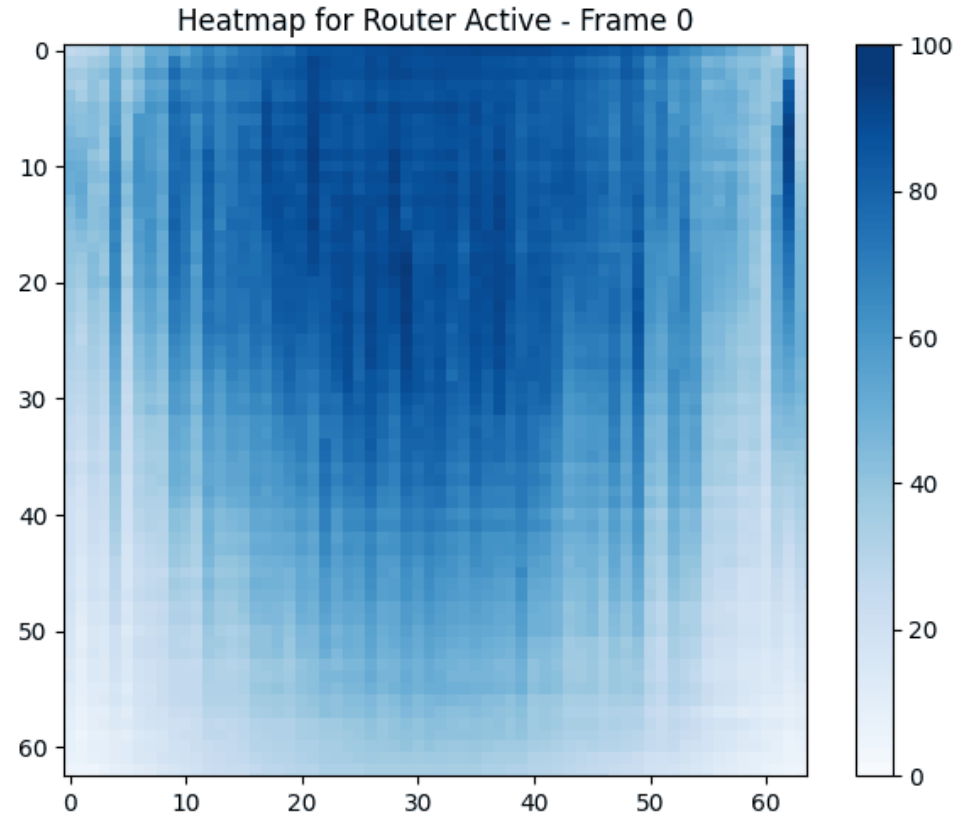


PRINCETON UNIVERSITY

BFS on RMAT-22, 64x64 tiles, Mesh NoC

With atomic operations





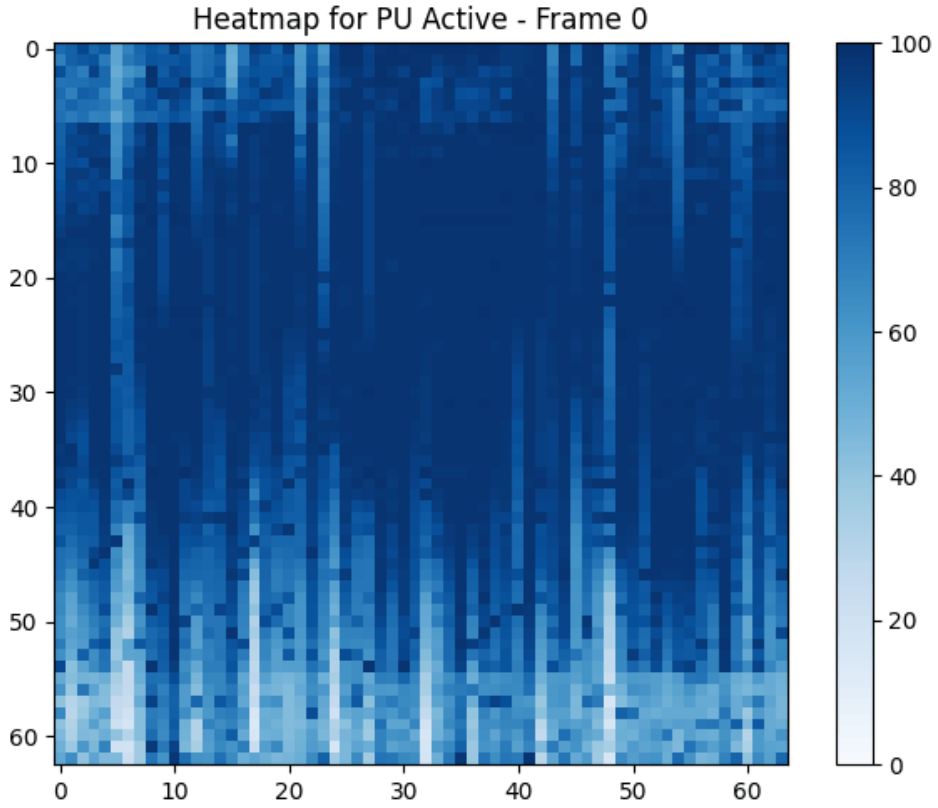

```
python3 gui/visualization.py --nogui -e 1
-p Heatmap -n HEAT64M -m 'FMCore Active'
```

```
python3 gui/visualization.py --nogui -e 1
-p Heatmap -n HEAT64M -m 'FMRouter Active'
```

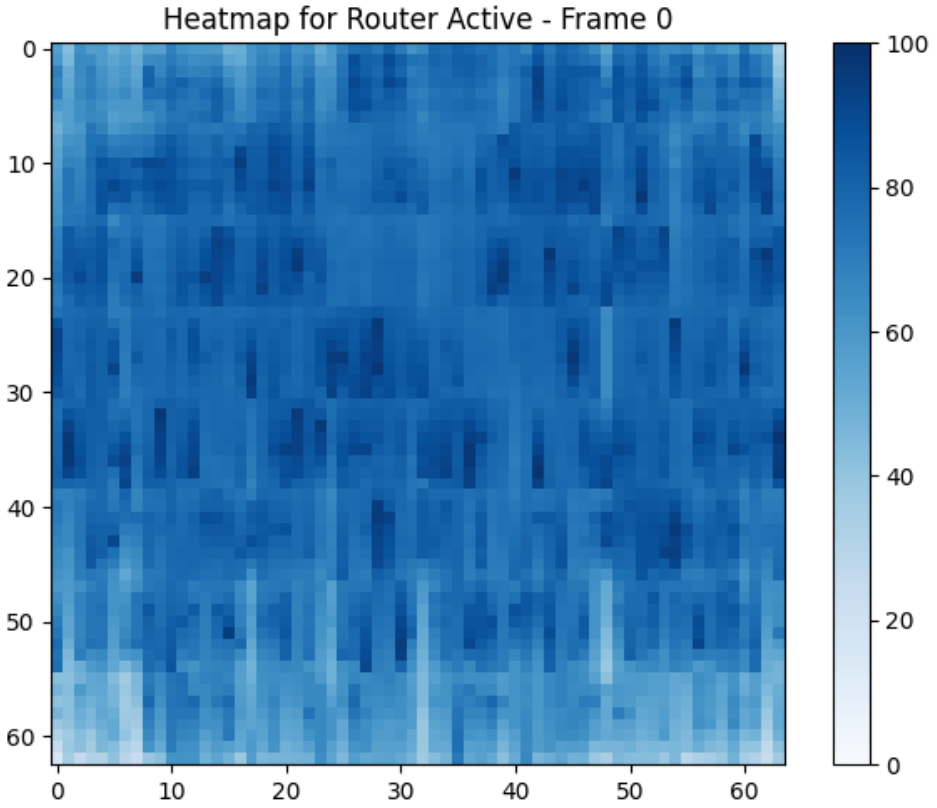
PRINCETON UNIVERSITY

BFS on RMAT-22, 64x64 tiles, Mesh NoC

With regional reductions




```
python3 gui/visualization.py --nogui -e 1
-p Heatmap -n HEAT8M -m 'FMCore Active'
```




```
python3 gui/visualization.py --nogui -e 1
-p Heatmap -n HEAT8M -m 'FMRouter Active'
```

PRINCETON UNIVERSITY

Visualization Tools

Case Study: Regional Reductions

Visualization Tools

Case Study: Memory-Compute Ratio

End-to-end Experiment

Simulation Speed & Scalability

Conclusions & Future Work

Work that users could do to explore System design and application mapping

Future work on the simulator itself

Instructions for Visualization Tools

`muchiSim/tutorial/2.VISUALIZATION.md`

PRINCETON UNIVERSITY

Visualization Tools

1. Plotting **per-PE performance counters** or **stats** for every execution frame
   - **Router** port collisions and utilization, end-point contention
   - **PU** utilization
   - **Cache** hit-rate and memory-controller requests and average latency.

2. Plotting aggregated execution metrics for combinations of **HW configurations**, and **applications** and **datasets.**
   - Runtime, throughput (FLOPS), energy, cost.
   - Network traffic, cache hit-rate, arithmetic intensity, etc.

PRINCETON UNIVERSITY

Plotting Experiments Across Several Benchmarks and Datasets

```
python3 plots/characterization.py -p 23 -m 0 -e 3
```

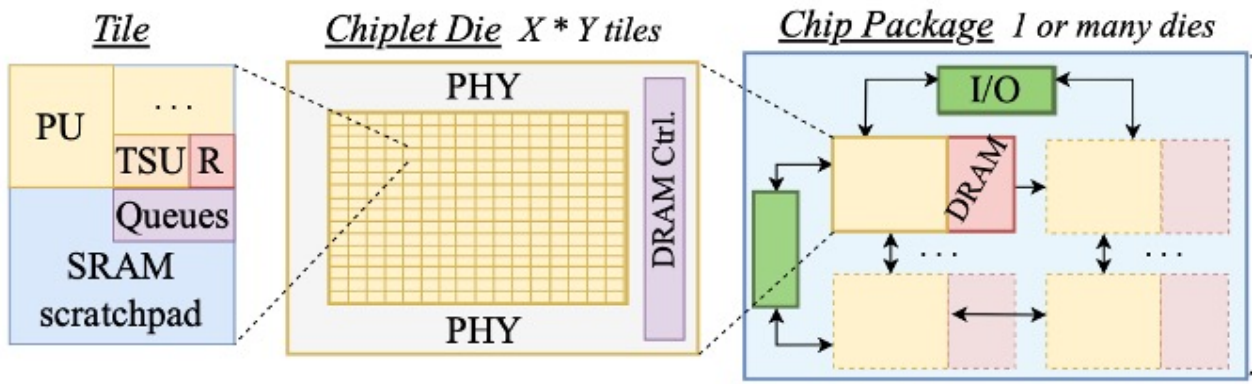
```
-p <plot_type> (based on different experiments performed in our studies
-m <metric> (0:time, 1:utilization, 2:noc_energy, 3:sim_time,
4:arith_intensity_msgs, 5:arith_intensity_loads, 6:flops, 7:energy, 8:total_msg,
9:dhit_rate, 10:perf/$)
-e <artifact_evaluation> (0:default, 1:muchisim_eval, 2:dcra_eval[1], 3:tutorial)
```

Characterizations are generated at **muchisim/plots/characterization/**

[1] M Orenes-Vera, E Tureci, D Wentzlaff, and M Martonosi "DCRA: A Distributed Chiplet-based Reconfigurable Architecture for Irregular Applications" 2023

PRINCETON UNIVERSITY

Case Study: Memory-Compute Ratio



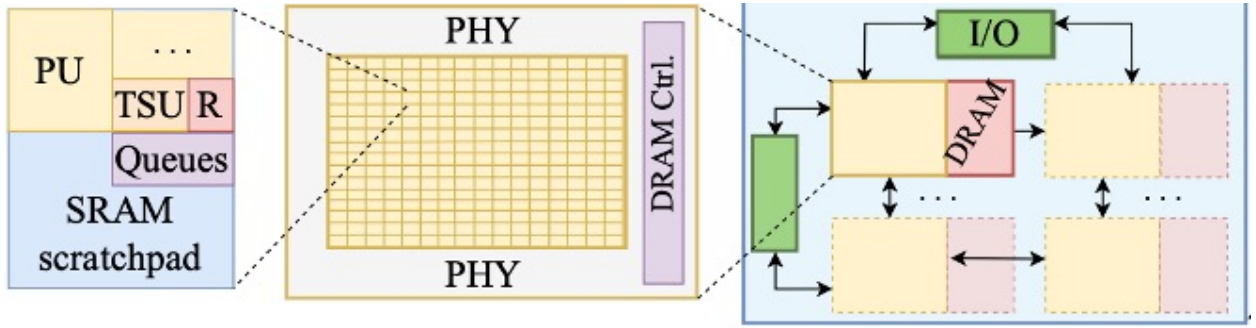
PRINCETON UNIVERSITY

Case Study

SRAM/tile and tiles/chiplet. RMAT-25

A chiplet is attached to an 8-channel HBM

- The number of tiles per chiplet (16x16 or 32x32) determines the ratio of tiles per HBM channel.
- 1024 tiles -> dataset footprint per tile 4-8 MiB



Plotting Memory Integration Case Study

These commands plot the memory integration case study. It compares all the configurations across the application suite, for the metrics of runtime (0), energy (7), and performance/$ (10).

```
python3 plots/characterization.py -e 1 -p 9 -m 0
python3 plots/characterization.py -e 1 -p 9 -m 7
python3 plots/characterization.py -e 1 -p 9 -m 10
```

Characterizations are generated at **muchisim/plots/characterization/**

PRINCETON UNIVERSITY

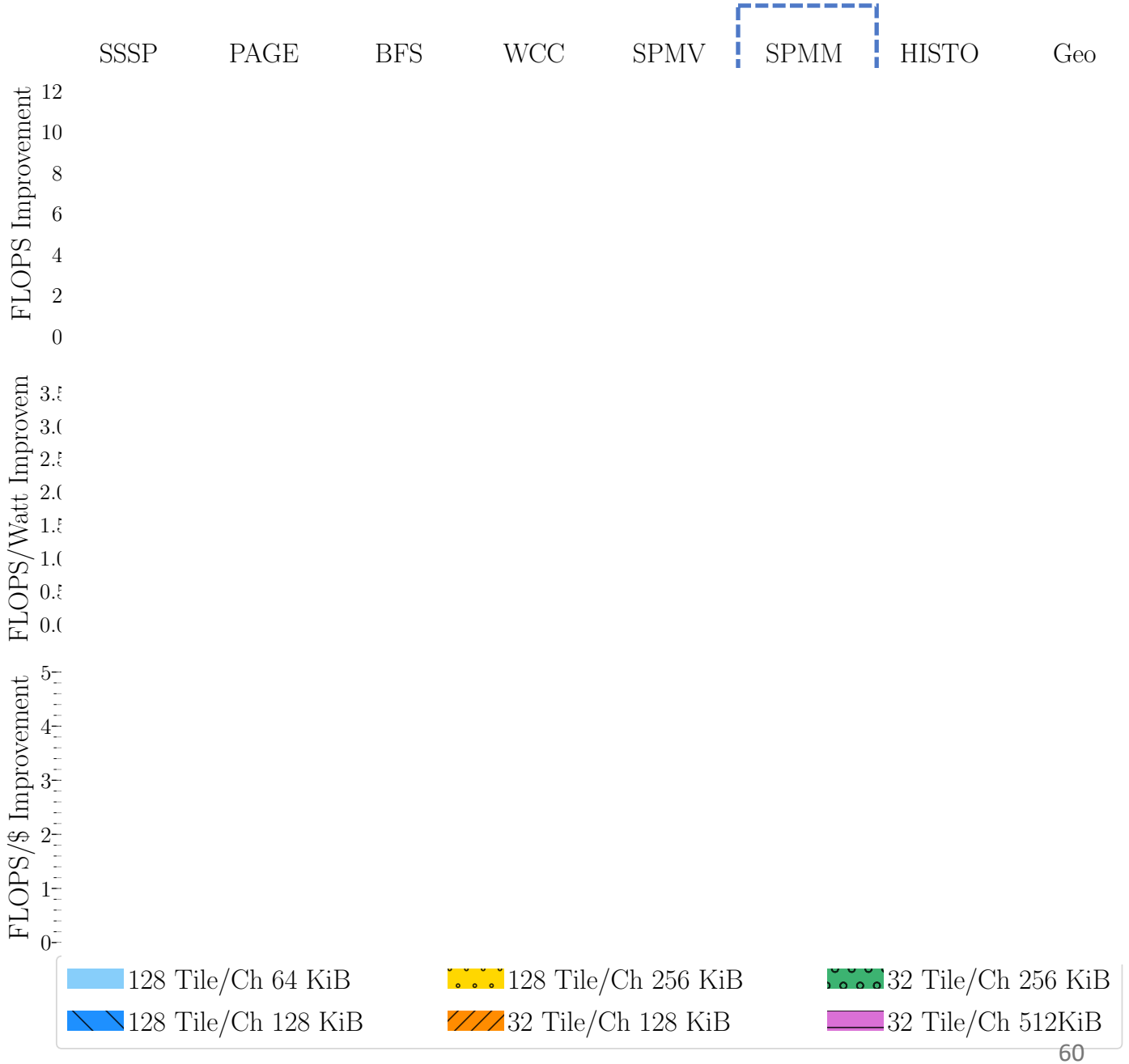
Case Study

SRAM/tile and tiles/chiplet. RMAT-25

SPMM has higher arithmetic intensity

- Lower memory bandwidth demands → Lower **FLOPS** gains with beefier mem.

32x32 chiplets (32 Tiles/Ch) higher **FLOPS/$** than 16x16 chiplets, despite total FLOPS gains



PRINCETON UNIVERSITY

Visualization Tools

Case Study: Regional Reductions

Visualization Tools

Case Study: Memory-Compute Ratio

End-to-end Experiment

Granularity of Processing Tiles

Simulation Speed & Scalability

Conclusions & Future Work

Work that users could do to explore System design and application mapping

Future work on the simulator itself

To run the end-to-end experiment & visualizations

```
source tutorial/3.EXPERIMENTS.md
```

PRINCETON UNIVERSITY

Granularity of the Processing Tile

Changing the number of PUs per tile while keeping the total number of PUs and bisection BW constant

```
exp/run_exp_granularity.sh
4 0 2 32 Kron16
```

Run SPMV (app=4), and 3 cases

- Configuration & run script prepares cases 0-2 with different configurations of tiles per chiplet, PUs/per tile (SMT) and network BW

```
python3 plots/characterization.py
-e 3 -p 23 -m 0 # Plot time

python3 plots/characterization.py
-e 3 -p 23 -m 7  # plot energy
```

To run the end-to-end experiment & visualizations

```
source tutorial/3.EXPERIMENTS.md
```

PRINCETON UNIVERSITY

Granularity of the Processing Tile

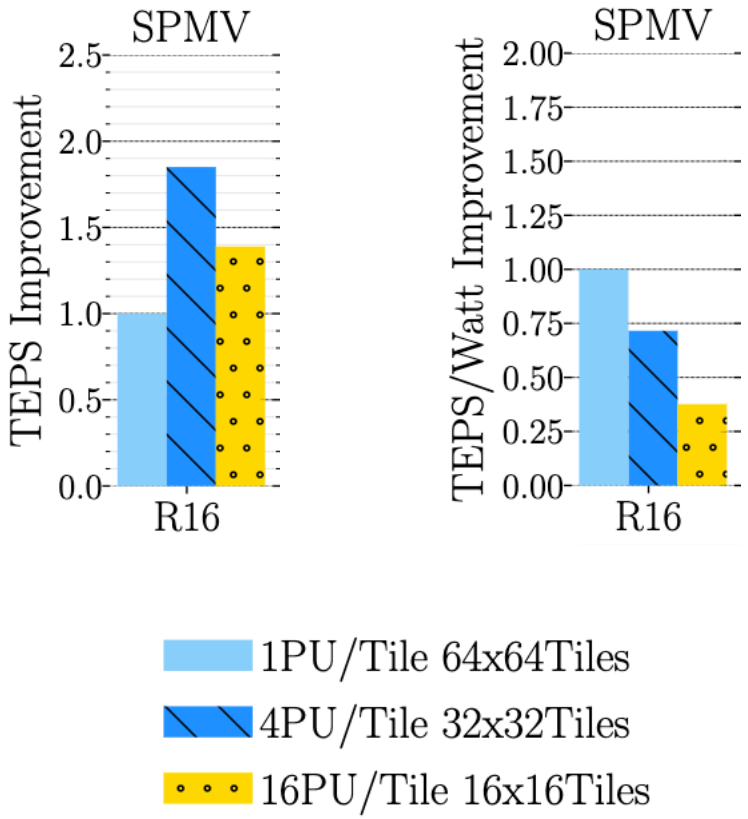
```
python3 plots/characterization.py
-e 3 -p 23 -m 0 # Plot time

python3 plots/characterization.py
-e 3 -p 23 -m 7  # plot energy
```

Inside `plots/characterization.py`

```
335         elif plot_type==23: #GRANULARITY
336             y_lim = 2.5
337             if artifact_type==3: # Tutorial
338                 apps = ['bfs'];
339                 inputs = ['Kron16'];
340                 binaries = ["GRANU0--32","GRANU1--16","GRANU2--8"]
341             else:
342                 apps = ['sssp','pagerank','bfs','wcc','spmv','histo']
343                 inputs = ['wikipedia','Kron22']
344                 binaries = ["GRANU0--64","GRANU1--32","GRANU2--16"]
345             comp = ("1PU/Tile 64x64Tiles",  "4PU/Tile 32x32Tiles",  "16PU/Tile 16x16Tiles")
```

PRINCETON UNIVERSITY

Granularity of the Processing Tile



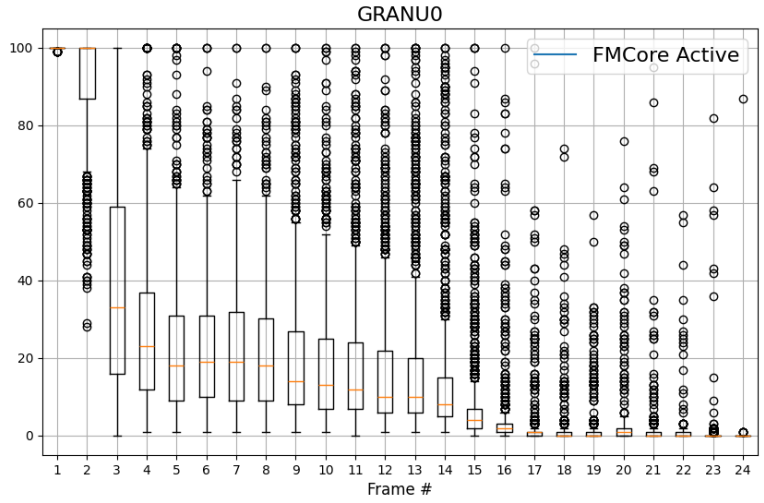

```
python3 plots/characterization.py
-e 3 -p 23 -m 0 # Plot time

python3 plots/characterization.py
-e 3 -p 23 -m 7  # plot energy
```

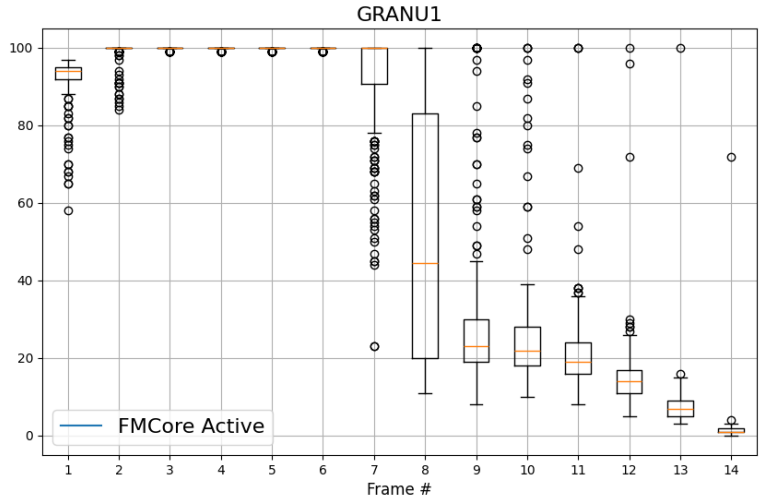
Characterizations are generated at **muchisim/plots/characterization/**

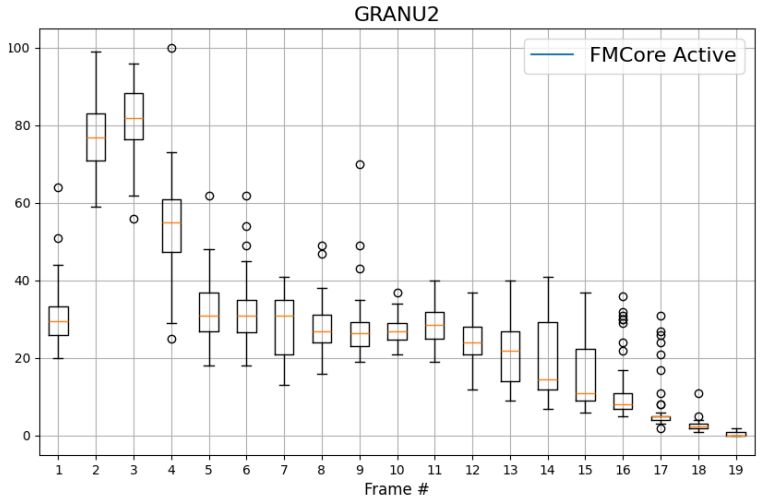
PRINCETON UNIVERSITY

Granularity of the Processing Tile

```
python3 gui/visualization.py --nogui -n GRANU0 -a 4 -s 32 -d Kron16
python3 gui/visualization.py --nogui -n GRANU1 -a 4 -s 16 -d Kron16
python3 gui/visualization.py --nogui -n GRANU2 -a 4 -s 8 -d Kron16
```







Boxplots are generated at **muchisim/plots/images**

PRINCETON UNIVERSITY

Visualization Tools

Case Study: Regional Reductions

Visualization Tools

Case Study: Memory-Compute Ratio

End-to-end Experiment

Simulation Speed & Scalability

Conclusions & Future Work

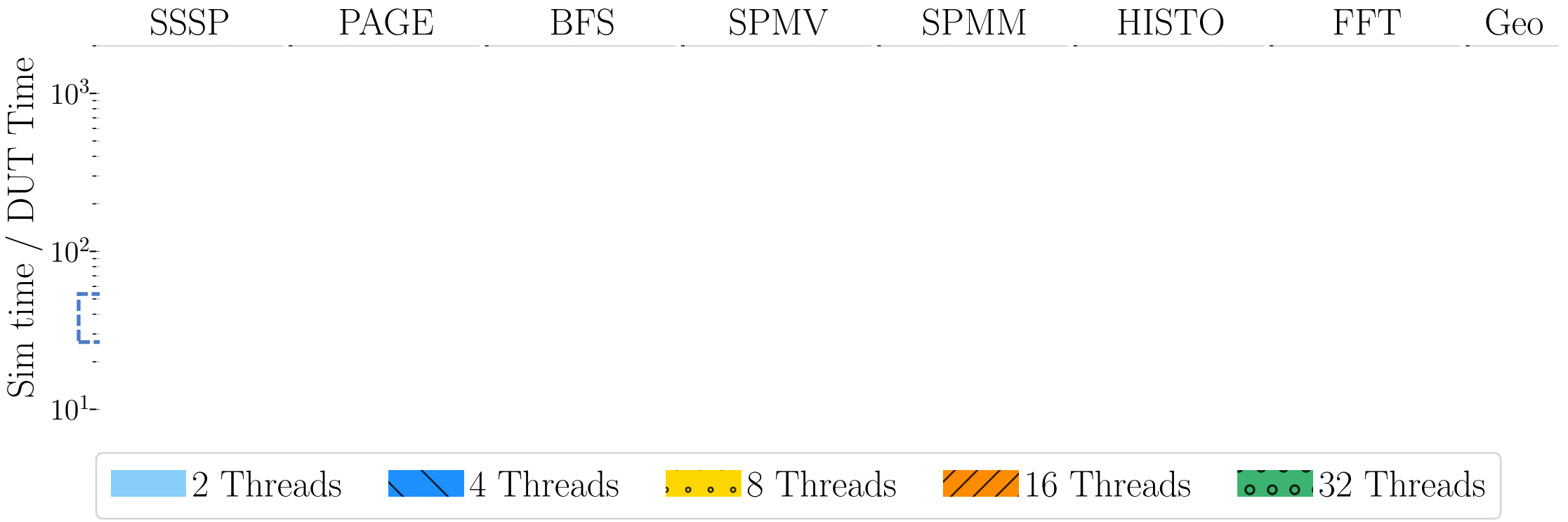
Work that users could do to explore System design and application mapping

Future work on the simulator itself

PRINCETON UNIVERSITY

Parallel Simulation

Scaling host threads from 2 to 32 (2-socket, 24 CPU/socket, ~100GB DRAM). RMAT-22 dataset.

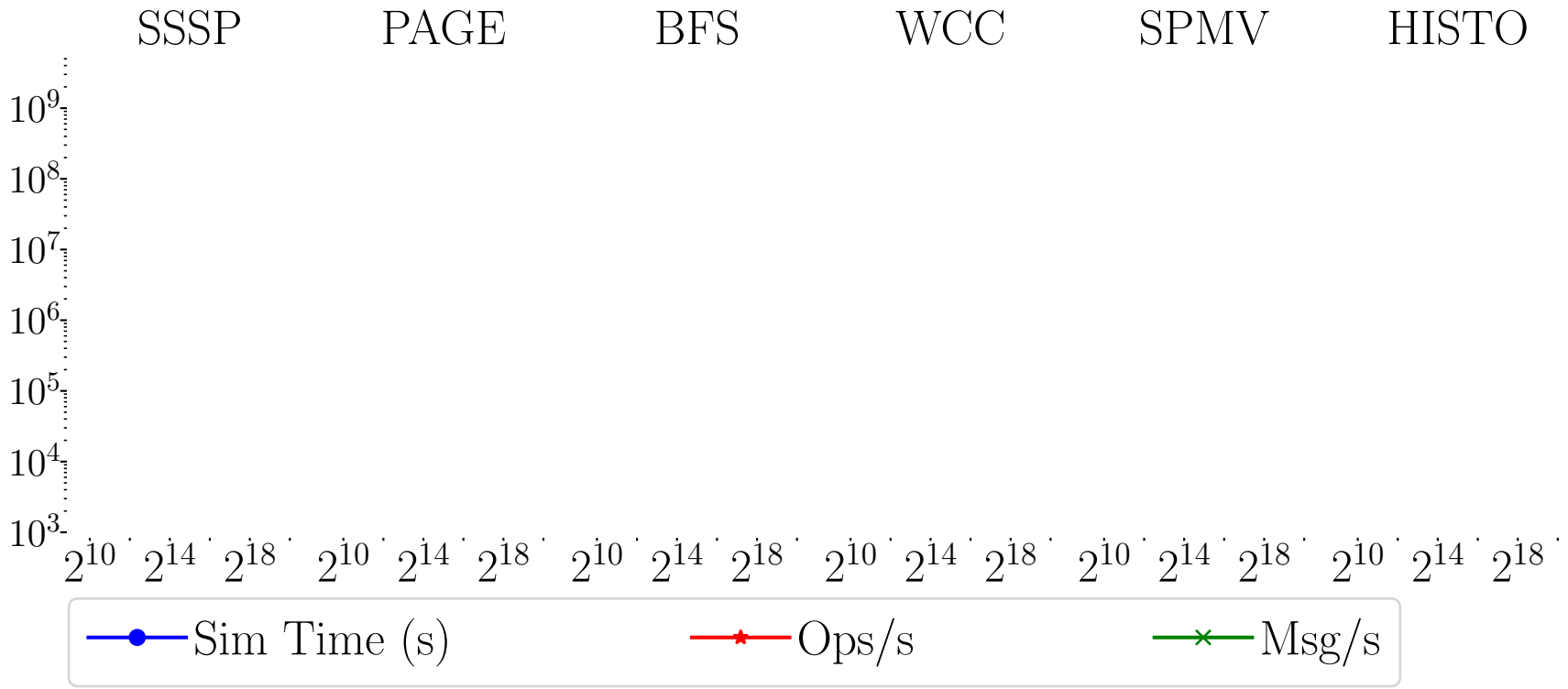


- Ratio between the simulator and the DUT runtime (aggregated for all tiles)
- The **simulator runtime decreases** close **to linearly with the number of host threads.**
  - For 32 threads, the simulator takes ~40x more than the DUT runtime.

PRINCETON UNIVERSITY

Simulation Speed

Scaling host threads from 16 to 128 (4-socket, 20 CPU/socket, 1TB DRAM). RMAT-26 dataset



- Evaluating 1M processing tiles for billion-element datasets in ~12-48 hours (10^5 seconds)
- Simulating up to 40 million routed messages/second
- Up to billions of operations/seconds

PRINCETON UNIVERSITY

Visualization Tools

Case Study: Regional Reductions

Visualization Tools

Case Study: Memory-Compute Ratio

End-to-end Experiment

Simulation Speed & Scalability

Conclusions & Future Work

Work that users could do to explore System design and application mapping

Future work on the simulator itself

Instructions for Visualization Tools

`muchiSim/tutorial/2.VISUALIZATION.md`

PRINCETON UNIVERSITY

Summary & Takeaways

MuchiSim is a simulation framework developed to explore the increasingly relevant design space of scale-out architectures.

- Particularly effective for **comm. intensive applications** such as graph analytics and sparse linear algebra.
- Scales up to simulate **millions of PUs** interconnected hierarchically, with options for different network topologies, and memory integrations.

Fully **open-source** and available for use and modification. Includes:

- Parametrizable simulator with performance, energy, area and cost model.
- 10+ ready-made scripts for experiments of different configurations
- Applications, datasets, and visualization tools.

github.com/PrincetonUniversity/MuchiSim

PRINCETON UNIVERSITY

Future work that Users (aka, you) can do

Things that **can be evaluated** now **only changing script parameters**:
- Case studies from the DCRA paper with different configurations or parameters

Things that can be evaluated **changing the application code:**
- Change dataset mapping
- Different PU models: e.g., ASIC, CGRA…

Further development on the simulator:
- Cluster-level network (with higher dimensionality)
- Multi-node simulation via MPI
- Some degree of hardware coherence

github.com/PrincetonUniversity/MuchiSim

PRINCETON UNIVERSITY

MuchiSim: A Simulation Framework for Design Exploration of Multi-Chip Manycore Systems

Marcelo Orenes-Vera, Esin Tureci, Margaret Martonosi and Stojche Nakov
{movera, esin.tureci, mrm, sn3332} @princeton.edu
Princeton University

Full paper!

Tutorial Website

github.com/PrincetonUniversity/MuchiSim