IXPUG Annual Conference 2023

Kairos

Innovation in Advancing HPC and AI Application Performance Analysis

Antonio Valles, Rebecca David

intel.

Notices & Disclaimers

- Performance varies by use, configuration and other factors. Learn more on the Performance Index site.
- Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
- Your costs and results may vary.
- Intel technologies may require enabled hardware, software or service activation.
-

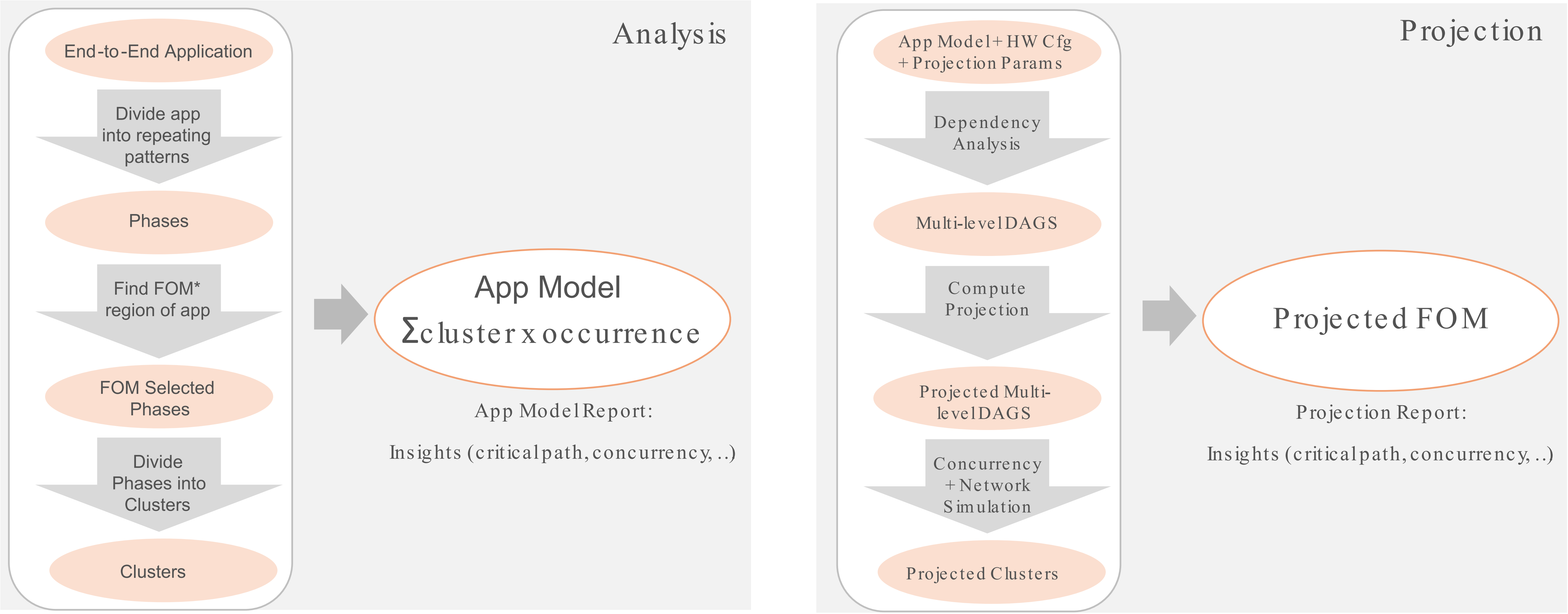
Executive Summary

Kairos is a unique capability not available inside or outside of Intel

- Methodology to analyze and project performance at system level
  - Top-down view of application, enables zoom -in to region of interest
  - Predicts impact of component performance on end -to-end application performance
  - Provides insights: critical path analysis, bottleneck analysis, concurrency behavior
- Key features
  - Quick overall understanding of application behavior
  - Visualize interactions between concurrent execution regions
  - Pathfinding/what -if capability

Kairos Flow

Methodology to analyze and project performance at system level



*FOM = Figure of Merit

Kairos Components

Tracing/Profiling

- MPI Shim
- CPU Shim
- GPU Shims:

  OpenMP, SYCL, CUDA*, OpenCL
- AI Shims: OneCCL , NCCL*,

  Pytorch / Tensorflow profilers

Simulation

- Concurrency Model
- Network/communication simulation

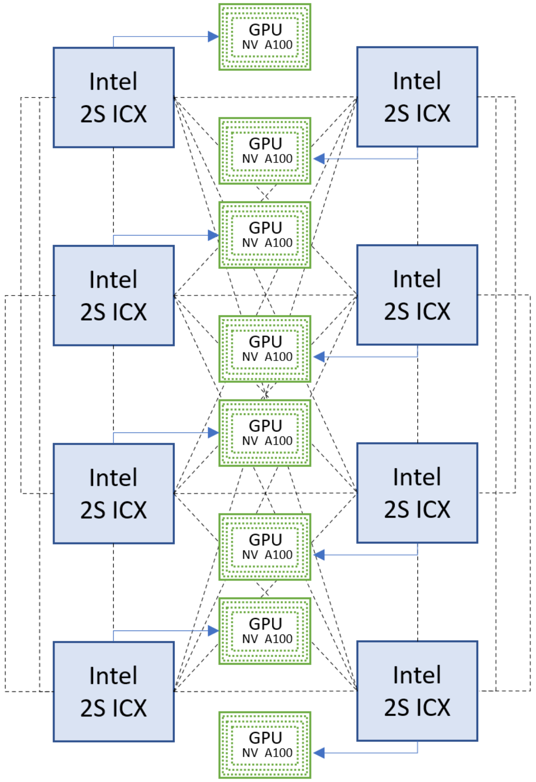
Reports/Profiles

- System trace (CPU, GPU, MPI, Mem, …)
- Visualize/Summarize/Compare data/Insights
- Break up app/zoom into areas of interest

Kairos Methodology With an Example

- Picked an example to show case methodology
- Example:
  - LAMMPS on 8 Nodes
    - 2 socket ICX and a single A100 GPU per node
    - 32 ranks per node, 2 OpenMP threads per rank; total of 256 ranks, 512 threads
    - MPS disabled on A100 GPUs
    - Figure of Merit (FOM) from run log:

```
Loop time of 35.7078 on 512 procs
```



Cluster of 8 ICX Nodes with 1 A100 GPU each
Fat-Tree Interconnect (indicative only)

Run is not meant to be performant run, solely meant to showcase Kairos methodology
Thank you to Mike Brown for his guidance on LAMMPS runs

Application Model:
Kairos Phases

Capture run behavior with Kairos shims

- Divide app into repeating patterns
- For MPI Apps → MPI Phases
- For AI App → AI Distributed Phases

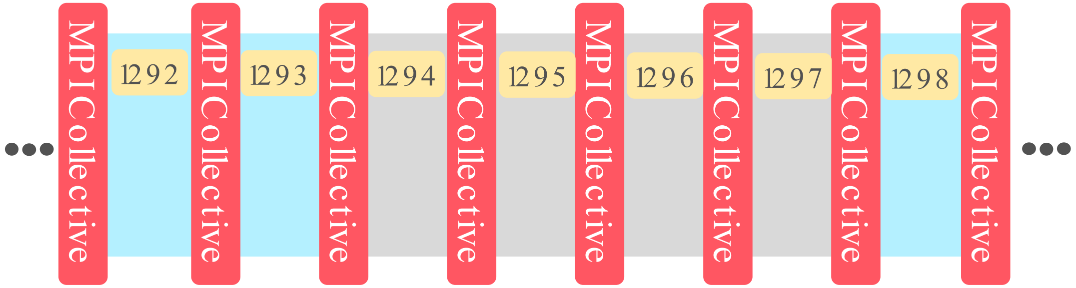
Phases captured across ranks using run behavior heuristics + collective APIs

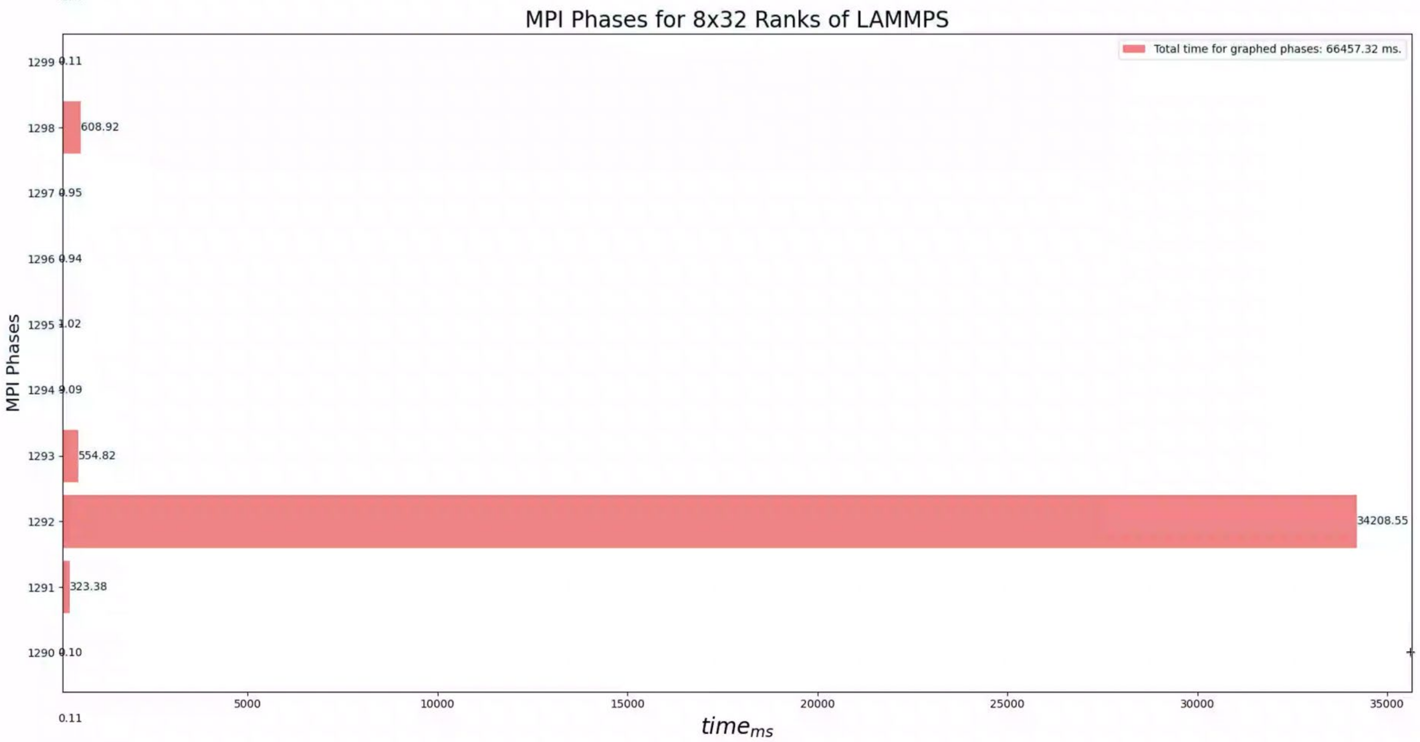
Methodology scales

- Phase numbers/behavior similar as user changes num ranks, num threads, SW configs, HW configs

Phases repeat, e.g. Phase 1292 occurred 624 times

Breaking app into phases using collectives and runtime behavior





Phase 1292 + 1293 + 1298 take significant amount of time

Application Model:

Kairos Full App Trace

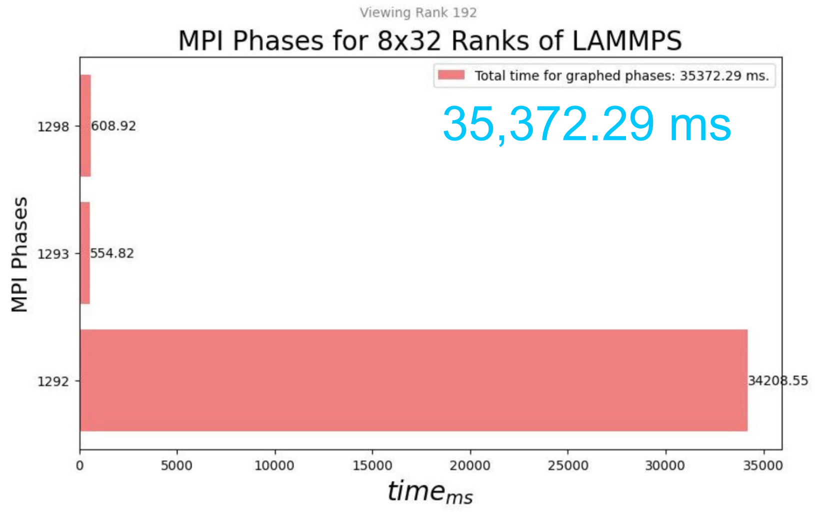
Find phases that make up Figure of Merit (FOM)

From Run Log: `Loop time of 35.7078 on 512 procs`

Application Model:
Kairos Selected Phases App Trace

- Select phases that best represent FOM
  - Can filter out insignificant phases
  - Or can keep all phases in FOM

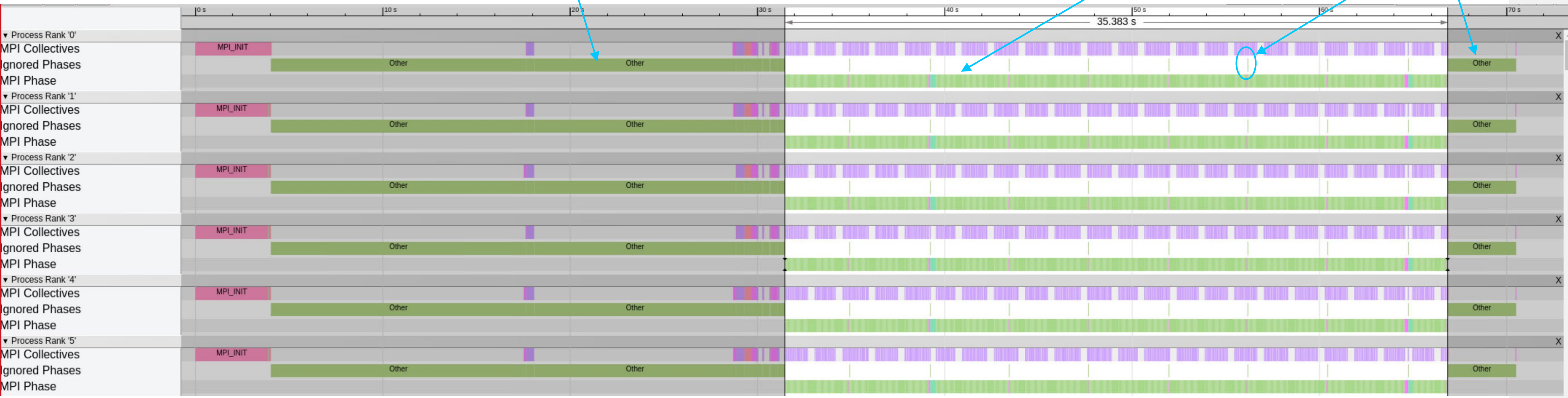
Phase 1292 + 1293 + 1298 represent 99% of app’s FOM



Skipped phases

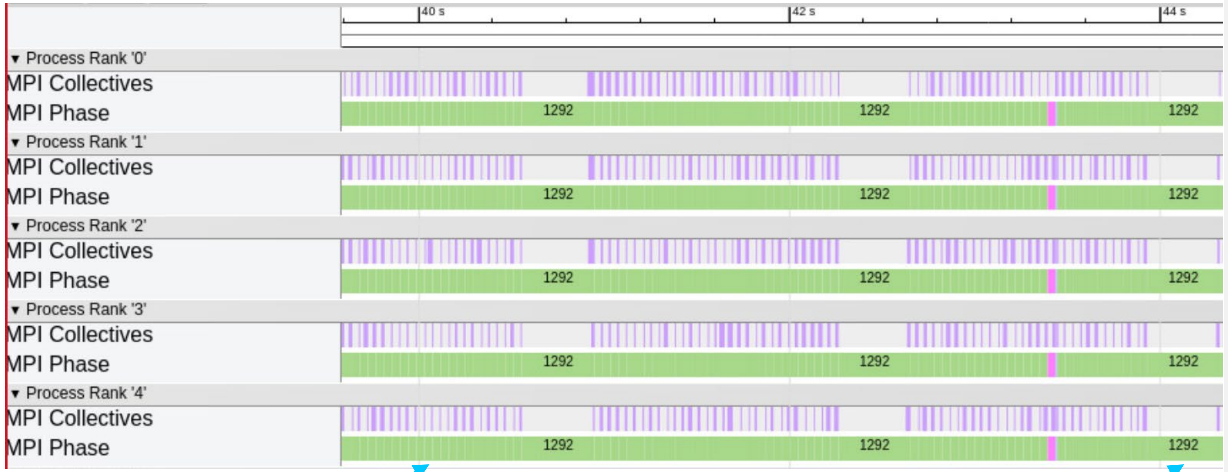
Selected phases

Skipped phases



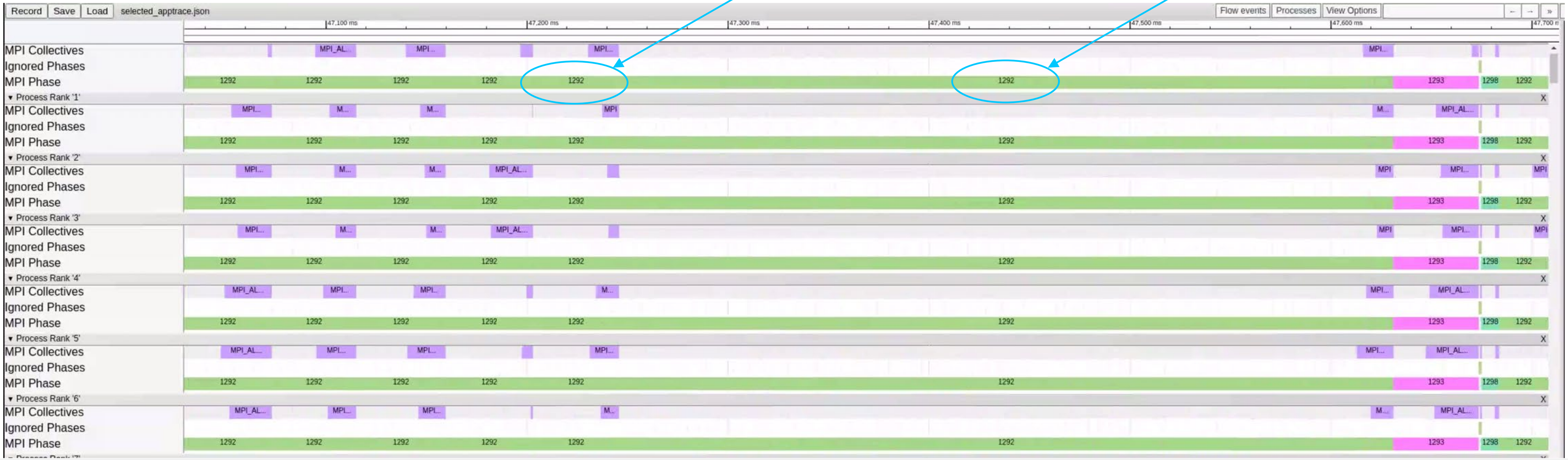
Application Model:
Kairos Clusters

- Same phase, different run behavior
  - E.g. 1292 short and long run duration
- Behaviors divide phases into clusters

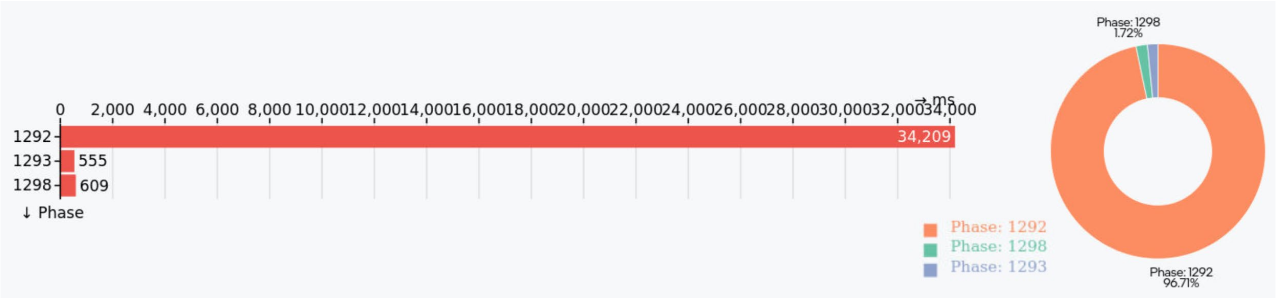


Short duration

Long duration



Application Model



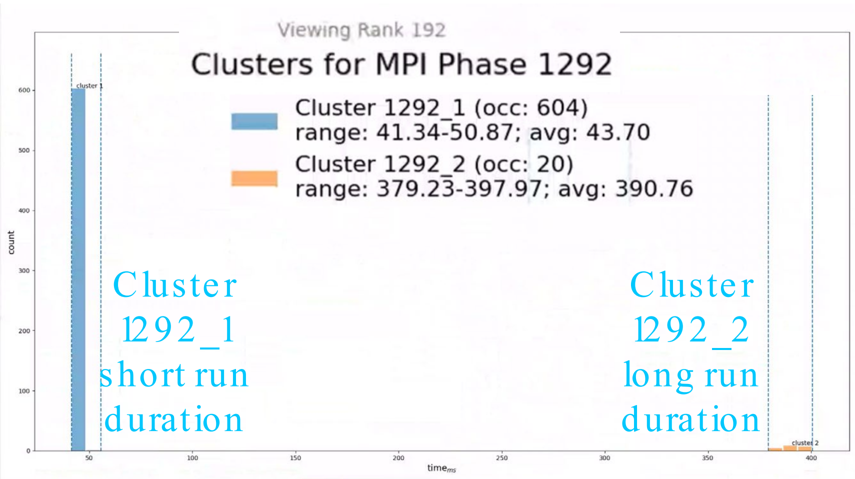
- Represent app using clusters selected
- Application Model

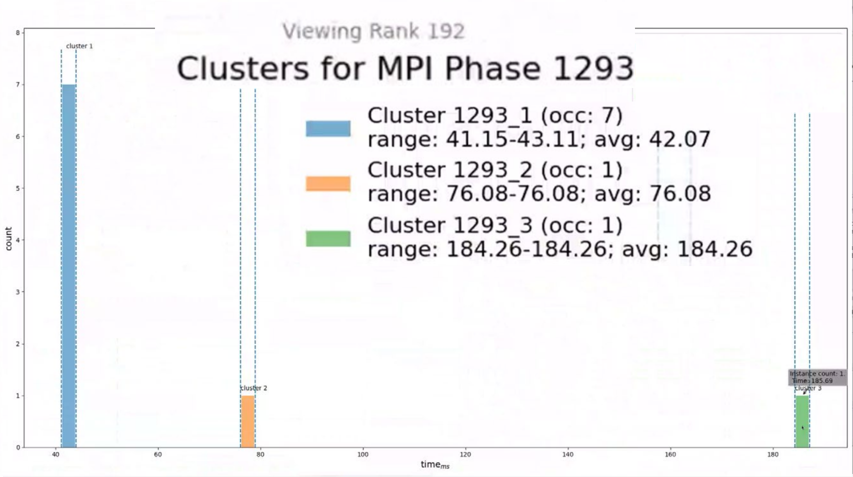
$= \sum cluster \times occurrence$

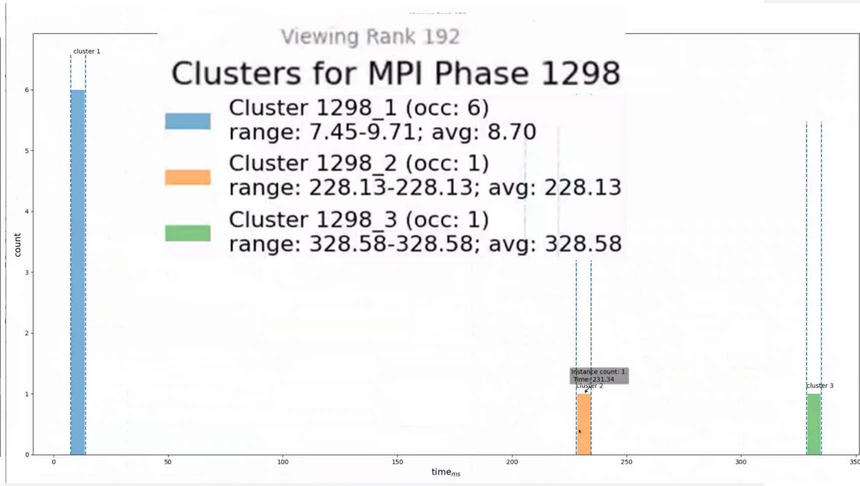
$= (1292\_1 * 604 + 1292\_2 * 20) + (1293\_1 * 7 + 1293\_2 * 1 + 1293\_3 * 1) + (1298\_1 * 6 + 1298\_2 * 1 + 1298\_3 * 1)$

$= (43.70 * 604 + 390.76 * 20) + (42.07 * 7 + 76.08 * 1 + 184.26 * 1) + (8.70 * 6 + 228.13 * 1 + 328.58 * 1)$

$= 35{,}373.74\ ms$







See backup for workloads and configurations. Results may vary.

Profiles:

Visualization of Cluster (Per Rank Behavior)

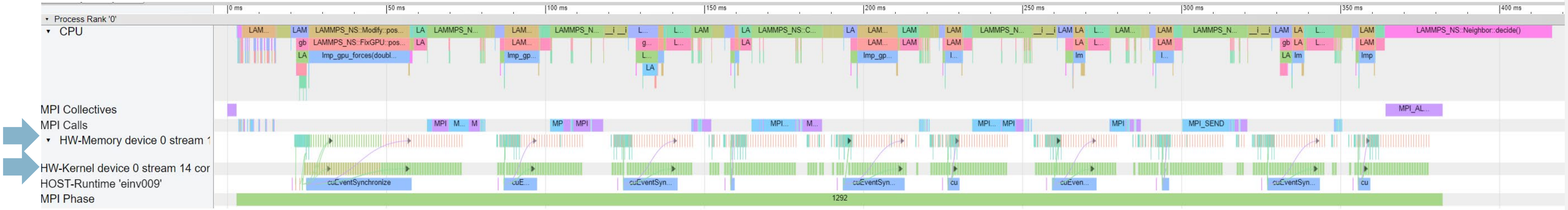
Profile contains:
CPU, MPI Collectives, MPI APIs, Host-Driver calls, Mem transfers, Kernel calls, Kairos Phase

See backup for workloads and configurations. Results may vary.

Profiles:

Visualization of Cluster (Per Device Behavior)

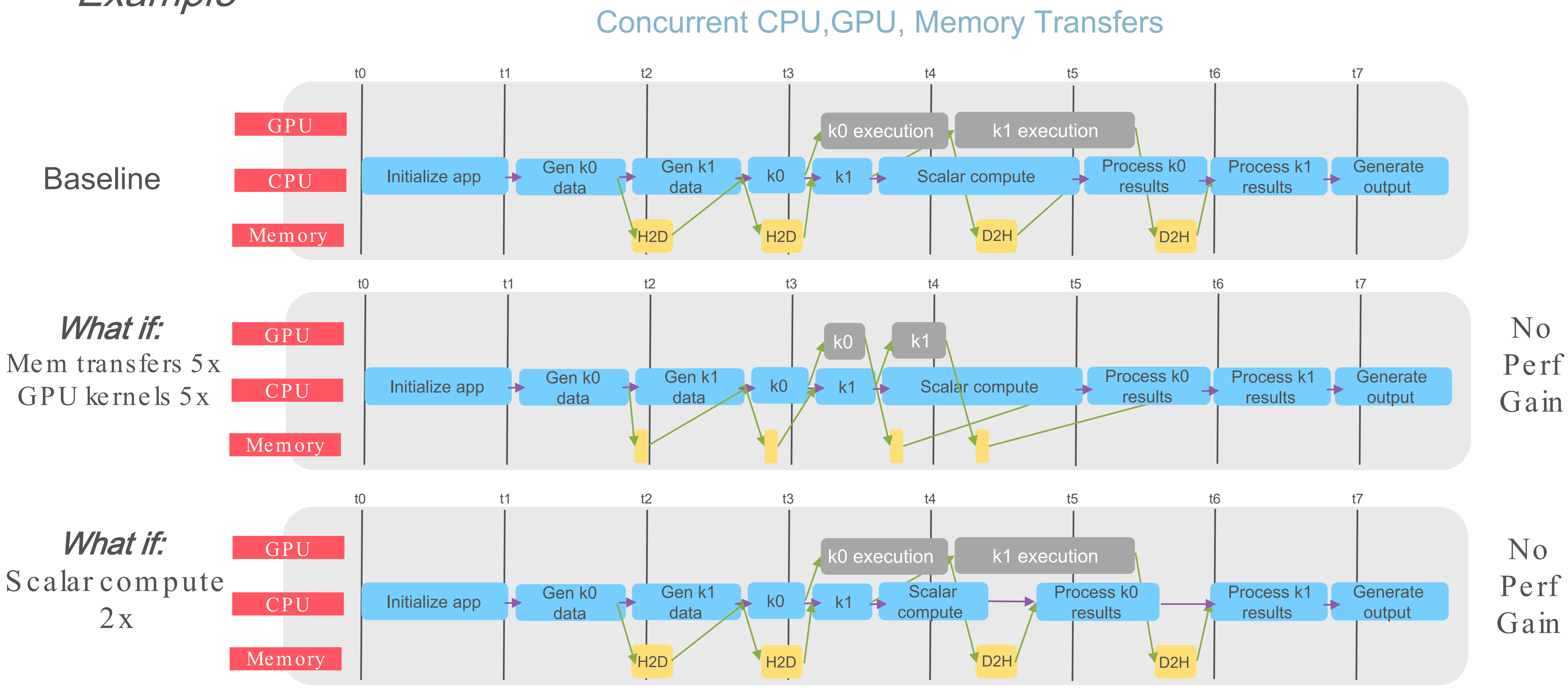
Easy to see each GPU's device utilization/mem transfers with this view (32 ranks sharing one GPU)



- View MPI Collectives, Calls, CPU data for single rank
- But view GPU sharing (all 32 rank's kernels and memory calls on same row)
- Create multi -level dependency analysis per device → DAG(s)

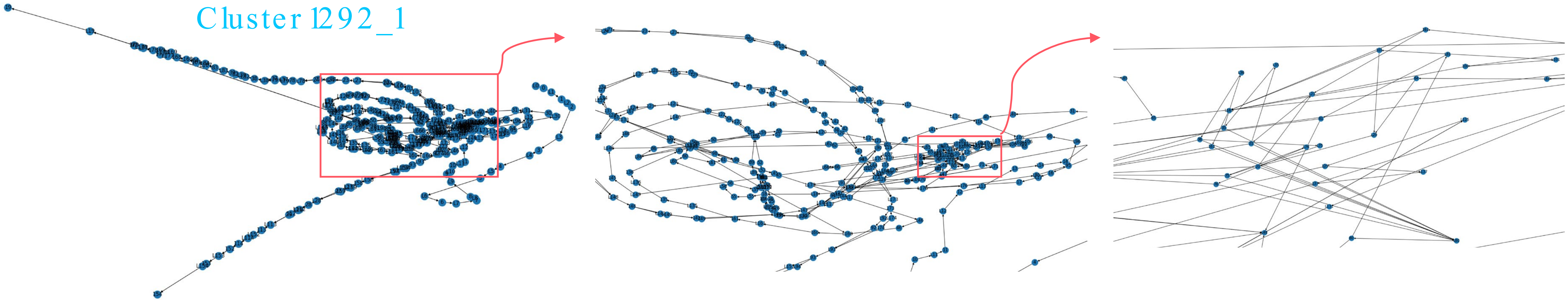
Kairos Dependency Analysis / Critical Path:

Example



Kairos Dependency Analysis/Critical Path

- Previous simple example
  - Only single rank on CPU with single GPU
- On a real application, multiple levels of concurrency
  - Multiple nodes, multiple ranks, multiple GPUs, Communication, …
- Kairos captures multi -level dependencies across execution regions creating multiple dependency graphs

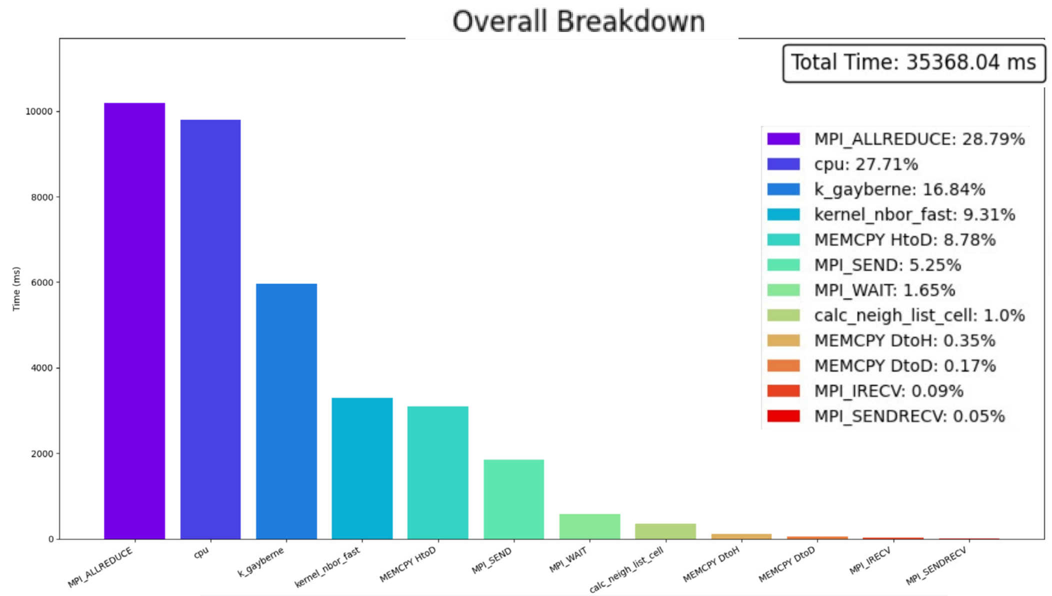


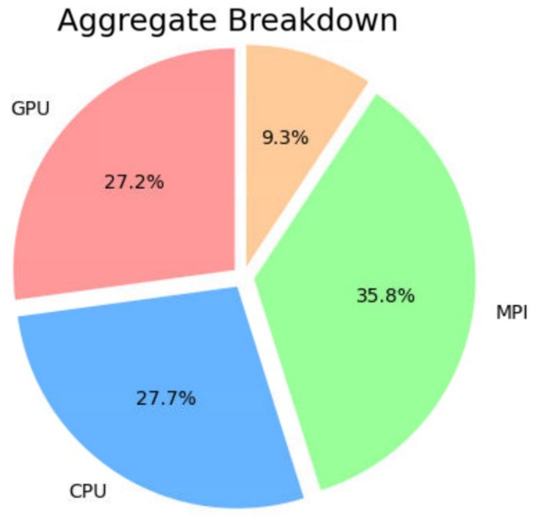
Actual Kairos dependency graphs, box locations approximate

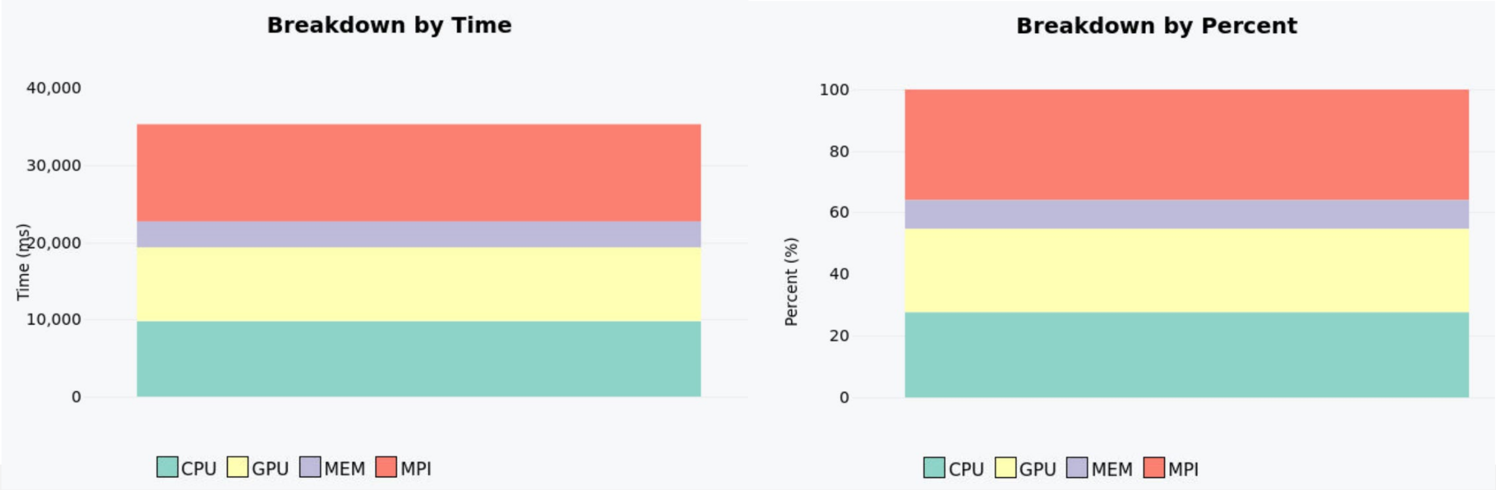
Kairos Insights:

Breakdown at cluster and app-level using critical path analysis from DAG

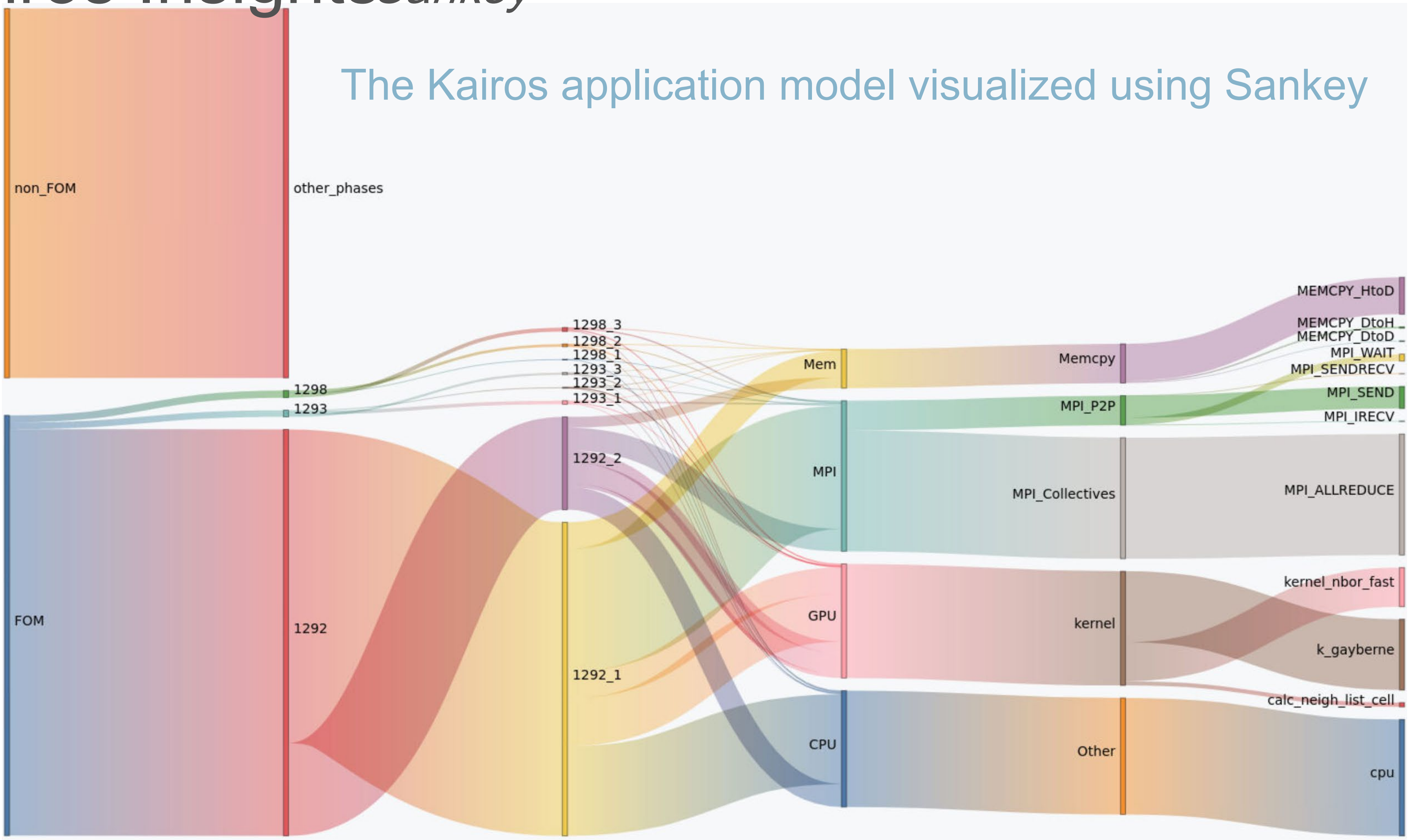
Overall Breakdown







Kairos Insights: *Sankey*

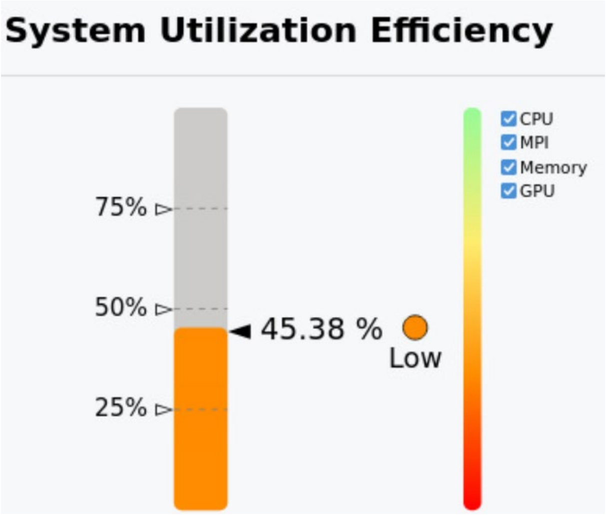


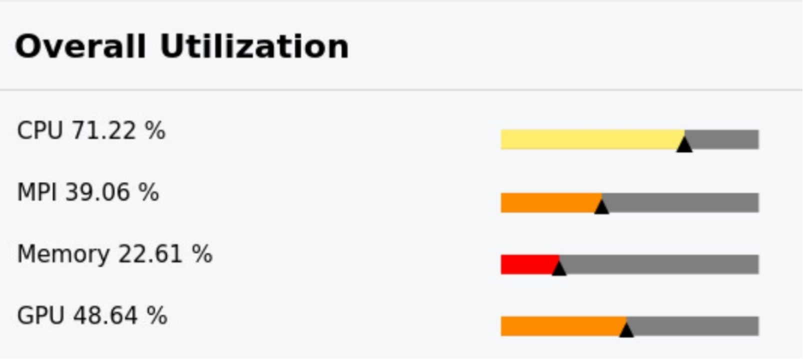
See backup for workloads and configurations. Results may vary.

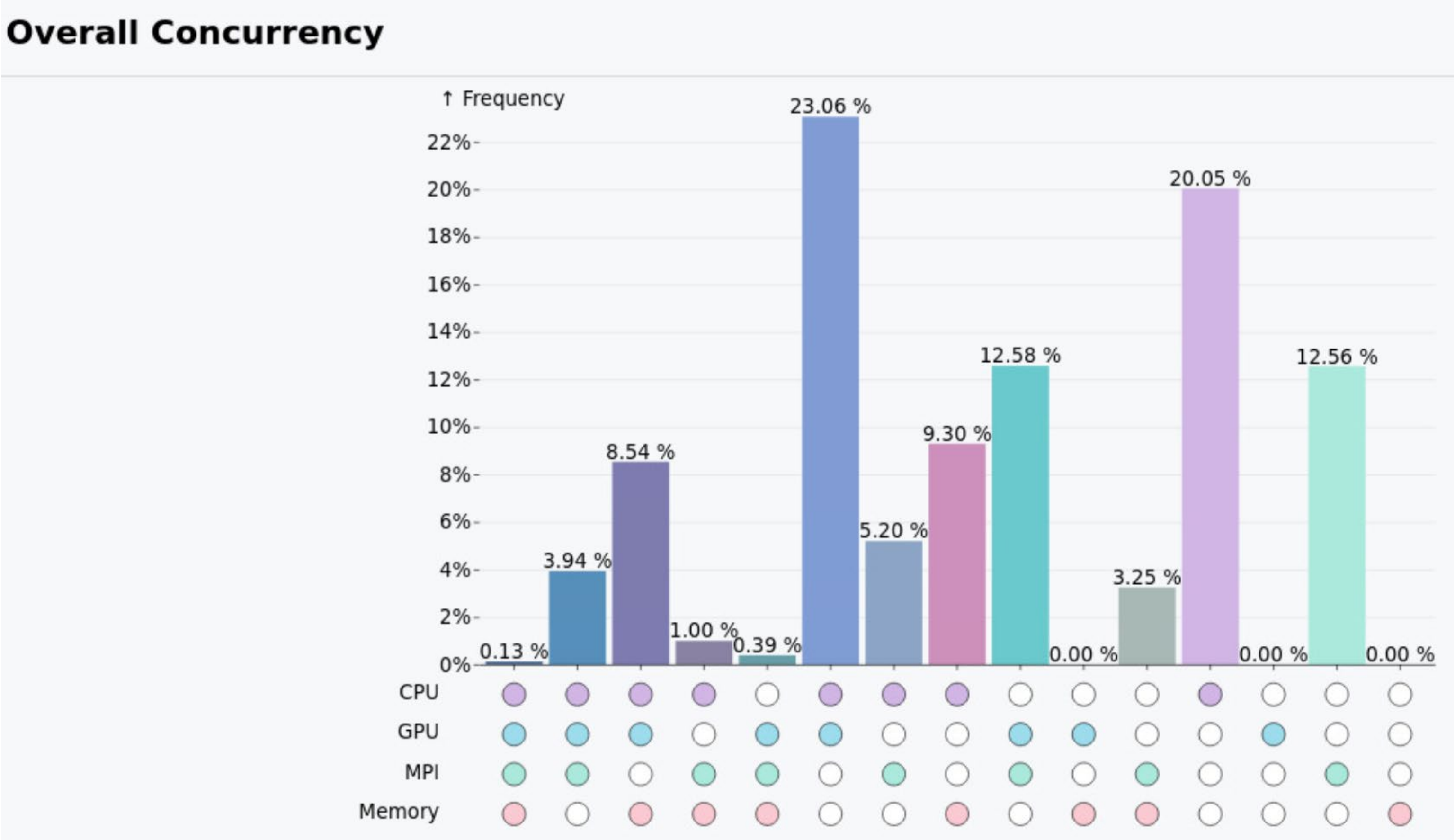
Kairos Insights

Concurrency at App-level

- Useful when comparing across systems / projections





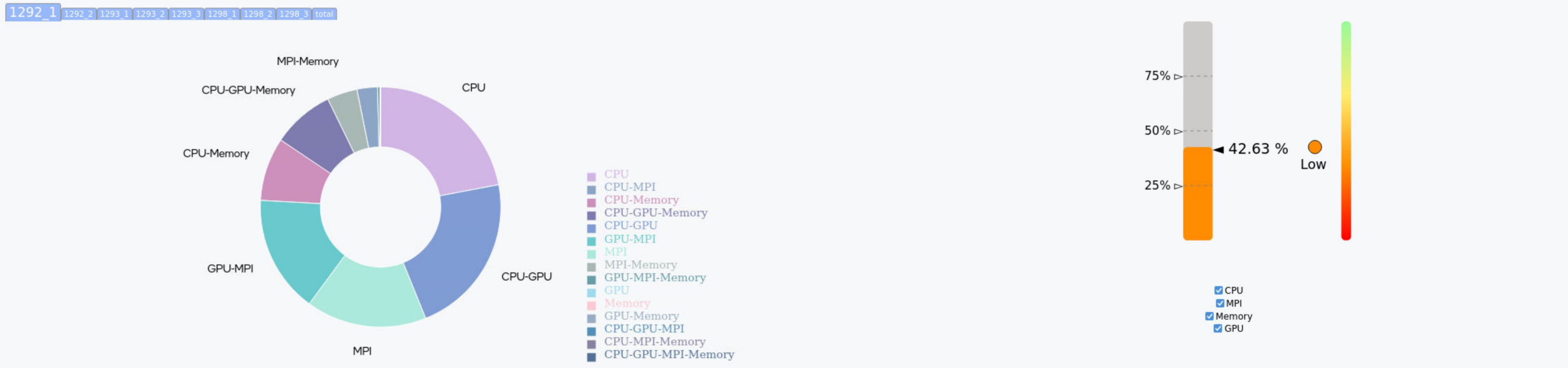


See backup for workloads and configurations. Results may vary.

Kairos Insights

Concurrency at Cluster-level

Example: Cluster 1292_1

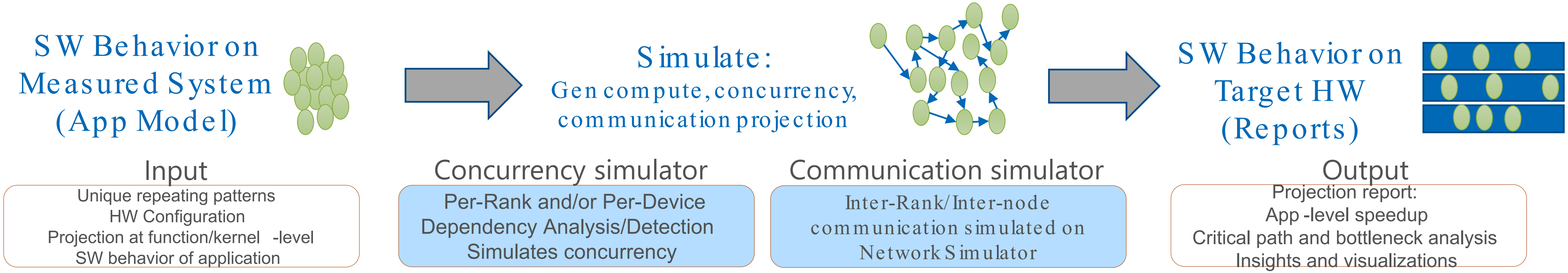


Cluster ID	FOM Pct	Concurrent Grade	CPU-GPU-MPI-Memory	CPU-GPU-MPI	CPU-GPU-Memory	CPU-MPI-Memory	GPU-MPI-Memory	CPU-GPU	CPU-MPI	CPU-Memory	GPU-MPI	GPU-Memory	MPI-Memory	CPU	GPU	MPI	Memory
total	100 % of FOM	Low 45.38	0.13 %	3.94 %	8.54 %	1.0 %	0.39 %	23.06 %	5.2 %	9.3 %	12.58 %	0.0 %	3.25 %	20.05 %	0.0 %	12.56 %	0.0 %
1292_1	**74.62 % of FOM**	**Low 42.63**	**0.0 %**	**0.0 %**	**8.35 %**	**0.0 %**	**0.4 %**	**21.86 %**	**2.76 %**	**8.54 %**	**15.8 %**	**0.0 %**	**4.06 %**	**22.02 %**	**0.0 %**	**16.21 %**	**0.0 %**
1292_2	22.09 % of FOM	Good 53.99	0.54 %	16.16 %	9.25 %	4.11 %	0.06 %	27.53 %	12.93 %	11.66 %	2.45 %	0.0 %	0.61 %	13.13 %	0.0 %	1.57 %	0.0 %
1298_3	0.93 % of FOM	Good 54.168	0.61 %	16.58 %	10.55 %	4.22 %	0.05 %	25.48 %	13.01 %	10.83 %	2.18 %	0.0 %	0.54 %	14.18 %	0.0 %	1.77 %	0.0 %
1293_1	0.83 % of FOM	Low 46.288	0.0 %	0.0 %	6.66 %	0.0 %	8.03 %	10.38 %	2.55 %	11.18 %	24.14 %	0.0 %	7.53 %	24.12 %	0.0 %	5.4 %	0.0 %
1298_2	0.64 % of FOM	Good 54.98	0.66 %	22.98 %	9.95 %	5.57 %	0.0 %	19.48 %	12.25 %	9.2 %	0.0 %	0.0 %	0.0 %	18.17 %	0.0 %	1.75 %	0.0 %
1293_3	0.52 % of FOM	Good 51.545	0.4 %	13.27 %	6.29 %	3.27 %	2.47 %	25.02 %	10.57 %	11.3 %	5.49 %	0.0 %	1.99 %	18.31 %	0.0 %	1.63 %	0.0 %
1293_2	0.22 % of FOM	Low 44.71	0.0 %	0.0 %	6.9 %	0.0 %	0.21 %	40.58 %	5.93 %	17.05 %	0.64 %	0.0 %	0.41 %	24.62 %	0.0 %	3.67 %	0.0 %
1298_1	0.15 % of FOM	No 25.0	0 %	0 %	0 %	0.0 %	0 %	0 %	0.0 %	0.0 %	0 %	0 %	0.0 %	79.53 %	0 %	20.47 %	0.0 %

See backup for workloads and configurations. Results may vary. intel.

Projection/Pathfinding Capability

- The application model can be used to create projections and pathfinding experiments to influence HW system design
- At a high-level
  - Extracts app SW components and dependencies
  - New SW and HW (CPU/GPU/NW/system) configs for target system
  - Simulate on target system, generate compute, concurrency, communication projection
  - Experiment with potential system configurations for pathfinding



Kairos Projection

What If?

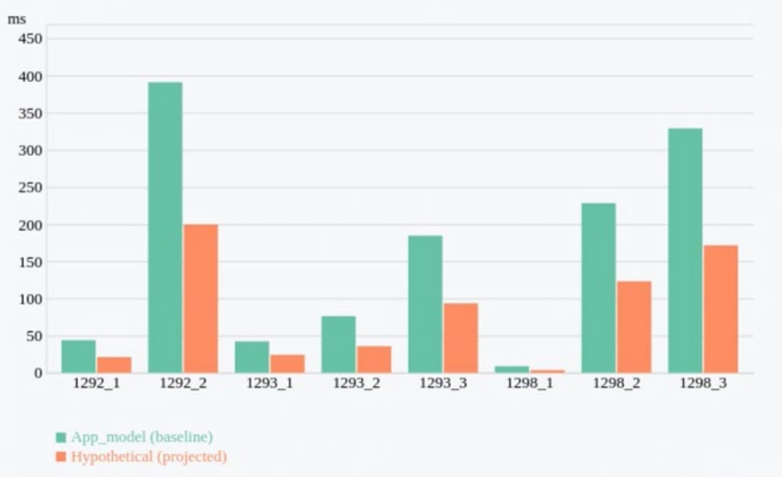
Kernels: 5x

CPU: 2.5x

Mem: 1.0x

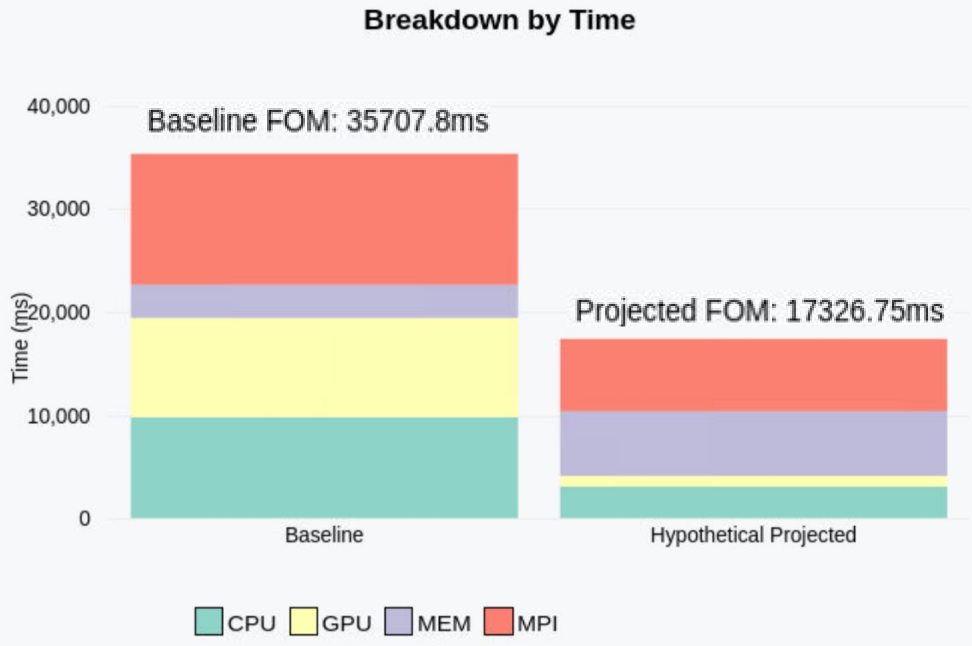
Overall end to end application speedup:

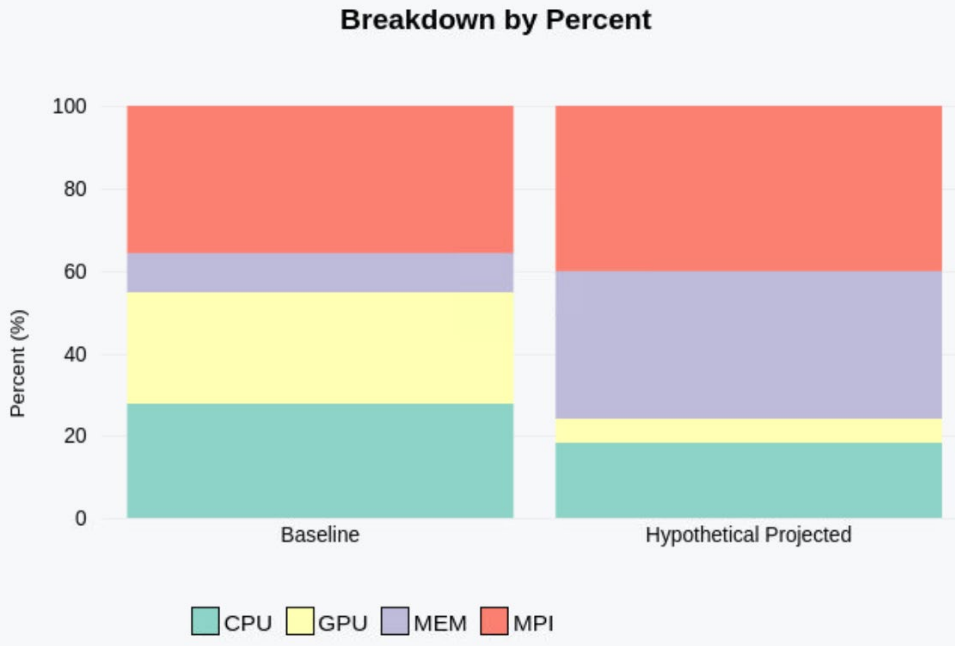
2.06x



Cluster Histogram Summary

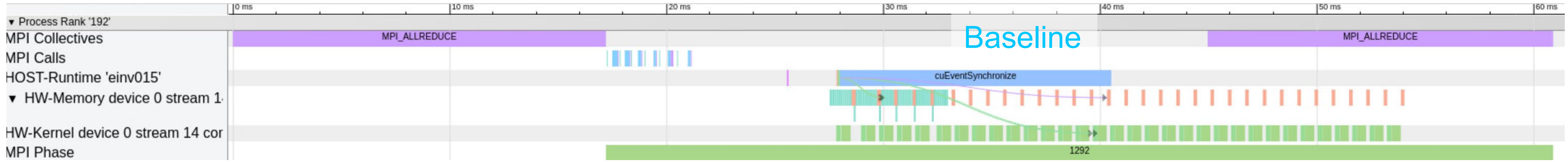
phase_number	cluster_number	occ	baseline_avg	projected_avg
1292	1	604	43.7	21.07
1292	2	20	390.76	199.55
1293	1	7	42.07	23.99
1293	2	1	76.08	35.57
1293	3	1	184.26	93.34
1298	1	6	8.7	3.5
1298	2	1	228.13	122.93
1298	3	1	328.58	171.39

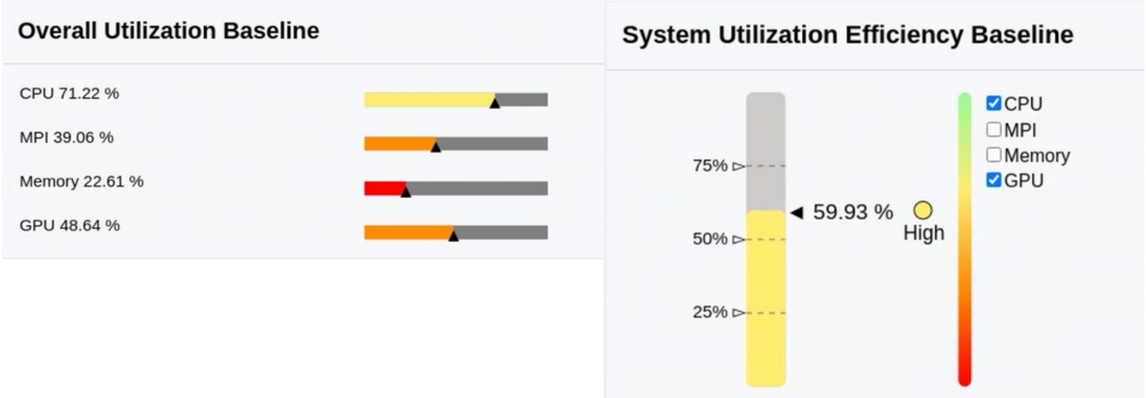


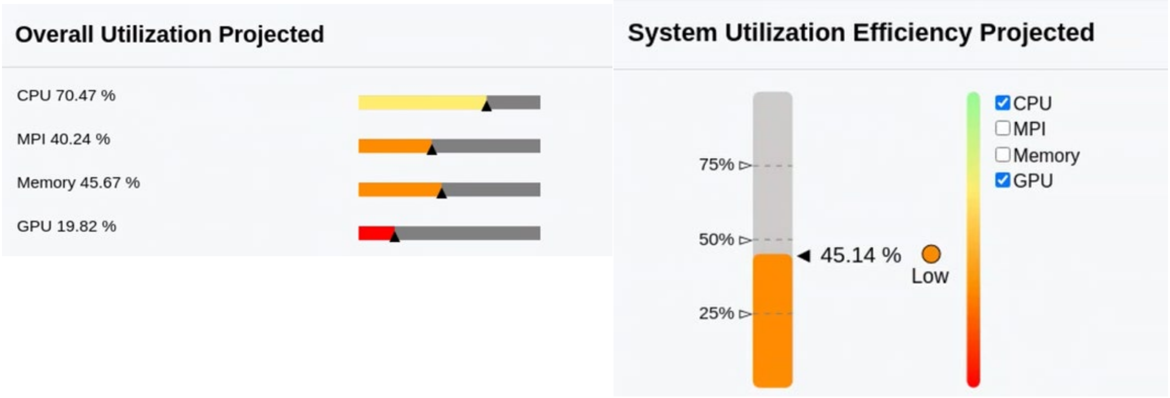


Kairos Projection

Insights

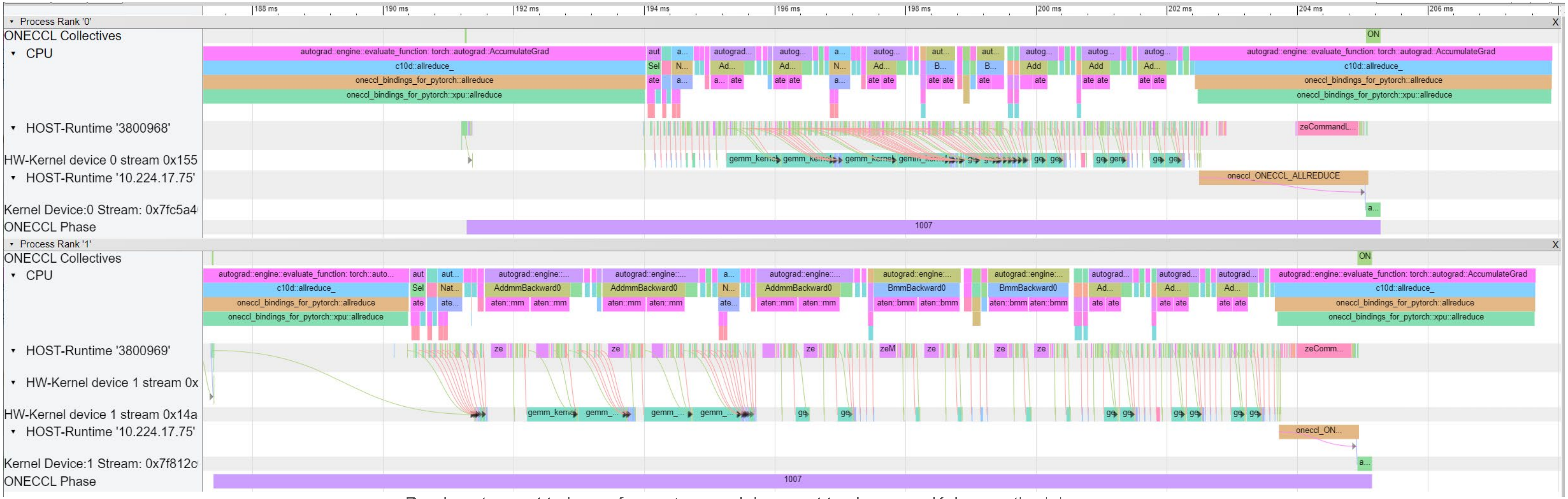






AI Example

- Kairos supports AI applications
  - AI Distributed Phases (OneCCL or NCCL based collectives)
  - The same flow and insights apply including app -model and projections



Run is not meant to be performant run, solely meant to showcase Kairos methodology

Summary

Kairos is a unique capability not otherwise available inside or outside of Intel

- Methodology to analyze and project performance at system level
- Compares behaviors between
  - Multiple implementations of same application
  - Comparisons between different system configurations
- Enables projections of end-to-end system performance for future Intel/competitive platforms
  - Pathfinding experiments to evolve Intel architecture to better support HPC/AI
  - Quick, high confidence customer projections

Back-up

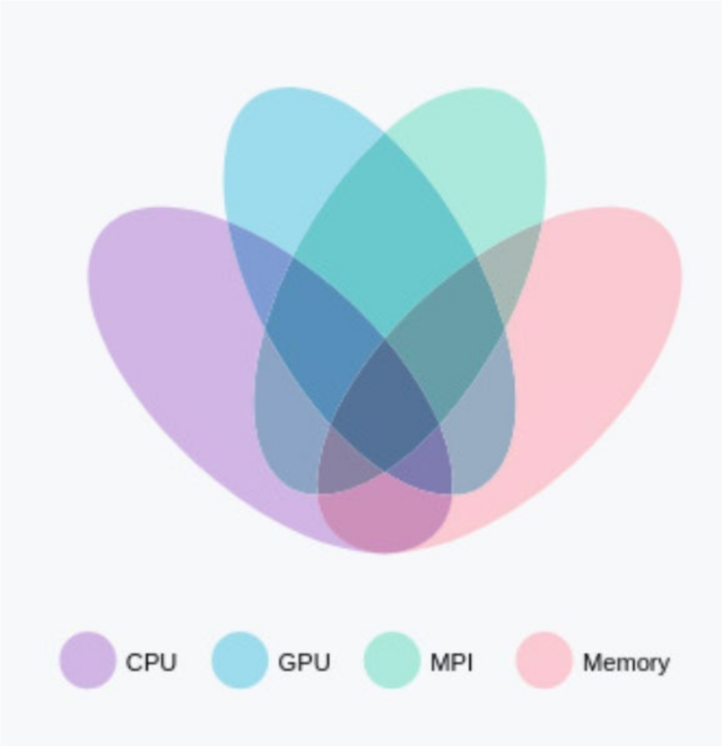
intel®

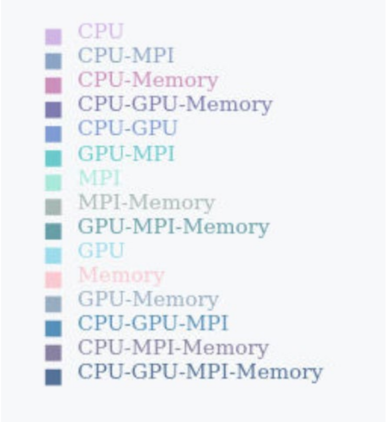
BERT Run and LAMMPS Config

- BERT:
  - app name: text-classification at huggingface /transformers
  - parameters: *--model_name_or_path bert-base-cased --task_name mrpc --max_seq_length 128 --per_device_train_batch_size 32 --learning_rate 2e-5*
- Single Node SDP config for BERT run:
  - OS: Ubuntu 22.04.3 (kernel: 5.15.47+prerelease23.4.120-1)
    - CPU: Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz
    - GPU: Intel(R) Data Center GPU Max 1550
    - OneAPI version: 2023.2.0
    - OneCCL version: 2021.10.0
- LAMMPS:
  - Intel Internal version
  - Intel Internal parameters
- Cluster configuration for LAMMPS run
  - OS: Rocky-Linux-8 (Linux version 4.18.0-477.21.1.el8_8.x86_64)
  - CPU: Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz
  - 8 Nodes, 36 cores per socket, two sockets, per node
  - Memory: DDR4_16x16GB@3200MHz
  - GPU: 2xNVIDIA A100-PCIE-40GB (but only 1GPU made visible per node during run)
  - Fabric: Mellanox_HDR
  - 32 ranks per node used with 2 OMP threads, 32 ranks per GPU

Why Kairos?

- Multi-node AI and HPC applications utilize CPUs, GPUs and interconnect concurrently to maximize performance
- Questions
  - How do we simplify?
  - How do we understand concurrent behavior?
  - How can we extract meaningful insights (critical path, bottleneck analysis, multi-level dependency analysis)?
  - How can we use our understanding of app's behavior to project performance when SW/HW changes?
  - If optimizations improve current bottlenecks, what are the next-level performance limiters?
- Problems with traditional approaches
  - Data collection limited to single-node or small number of nodes
  - Data collection significantly impacting performance
  - Providing data too detailed for meaningful insights
  - Insufficient abstraction for overall understanding of application





Introducing Kairos

Methodology to analyze and project performance at system level

- Analysis Capabilities:

  E2E application → FOM*/Region of Interest → App model →

  Insights (multi-level dependency analysis, critical path, concurrency, ..)

- Projection Capabilities:

  App Model + target configs (SW/HW) → Simulate on target platform

  (Compute projection, concurrency simulation, communication simulation) →

  Projected FOM → Insights

*FOM = Figure of Merit

intel®