OpenMP®

SC24 Booth Talk Series

Enabling Graph Execution with OpenMP Taskgraph

Chenle Yu, Sara Royuela, Eduardo Quiñones
Barcelona Supercomputing Center

ascender

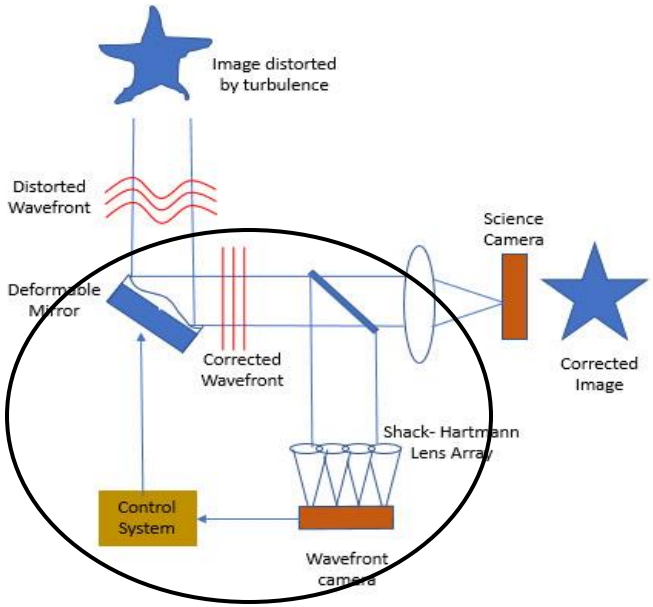
BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

HiPART

Motivation – Modern CPS

- ➔ Real-time CPS (Cyber-Physical Systems) requiring tight time constraints: autonomous cars, astronomical telescopes (e.g., Adaptive Optics), etc.
- ➔ High performance requirement
- ➔ Constantly executing the same workflow

Adaptive Optics in Astronomy



credit: andor.oxinst.com

Illustrative Code of AO

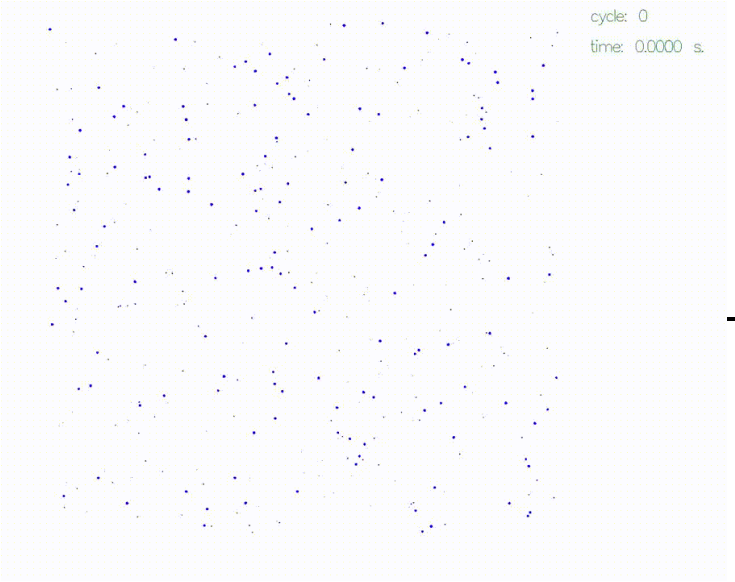
```
int operationLoop(…) {
  #pragma omp parallel
  #pragma omp single
  {
    #pragma omp task depend(out:op1)
    calibration(…)

    #pragma omp task depend(in:op1)
    fillIntensities(…)

    // other computations

    #pragma omp task depend(in:op_x)
    commandActuators(…)
  }
}
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Motivation – Iterative Applications

- ➔ Simulations (Gauss-Seidel, N-body, etc) and linear system solvers such as Cholesky decomposition
- ➔ Performance requirement
- ➔ Same execution is replayed multiple times

Illustration of Nbody Simulation



credit: en.wikipedia.org

Example Code of Nbody

```
#pragma omp single
for (timestep < IterNumber):
  for (each particle):
    #pragma omp task depend(in:op_x)
    {
      calculate_force(…)
    }
  }
}
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Objective – Performance Enhancement

Taskgraph – a graph-based execution that allows us to

- **Reduce Overhead**
- **Lower the Contention on Shared Resource**
- **Easily Map to Other Programming Models**

Example Code of AO with Taskgraph

```
int operationLoop(…) {
  #pragma omp parallel
  #pragma omp single
  {
    #pragma omp taskgraph
    {
        #pragma omp task depend(out:op1)
        calibration(…)

        #pragma omp task depend(in:op1)
        fillIntensities(…)

        // other computations

        #pragma omp task depend(in:op_x)
        commandActuators(…)
    }
}
```

- The *single* thread records the tasks (computational functions) as nodes of the taskgraph
- When calling *operationLoop,* if a taskgraph is recorded, replay the taskgraph without to create the tasks
- Fewer accesses to task queues and no task dependency resolution overhead
- Potentially map the taskgraph to accelerators with OpenMP offloading and other PMs, e.g., CUDA

Outline

- How to use *taskgraph* construct, the expected syntax in OpenMP 6.0
- How are the graphs generated?
- Performance Evaluation with Host Tasks
- Interaction with other programming model (e.g., HIP and CUDA)
- Evaluating GPU graph created with *taskgraph*
- Conclusions

Taskgraph – Current Syntax[1]

#pragma omp taskgraph […]

- *graph_id(int)*
- *graph_reset(boolean)*
- *nogroup*
- *If (boolean)*

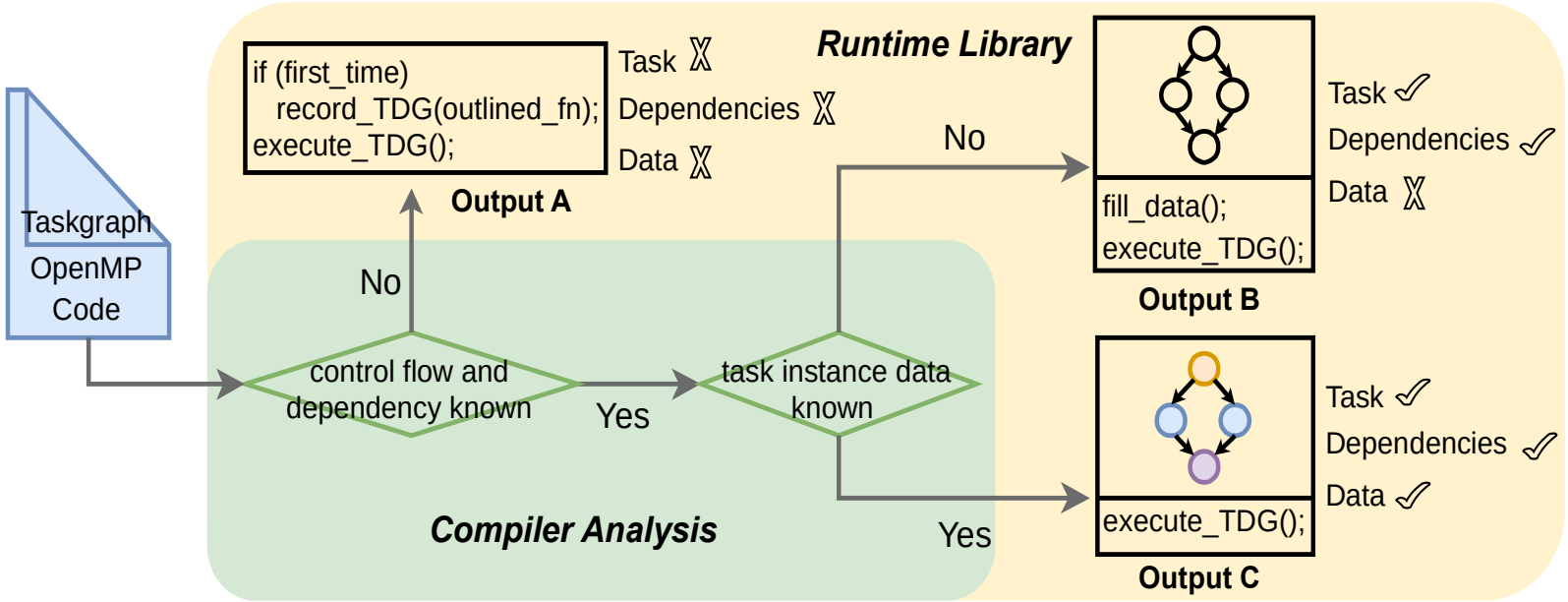
```
int operationLoop(num) {
  #pragma omp parallel
  #pragma omp single
  {
    #pragma omp taskgraph graph_id(num)
    {
        #pragma omp task depend(out:op1)
        calibration(…)

        #pragma omp task depend(in:op1)
        fillIntensities(…)

        // other computations

        #pragma omp task depend(in:op_x)
        commandActuators(…)
    }
}
```

- *graph_id(int)*: allows to instantiate different graphs with the same pragma, each associated with an ID. If implemented with a cache mechanism, could alternate the executions of different graphs.

- *graph_reset(boolean):* reset an existing graph. This allows users to overwrite an existing graph with a new set of tasks + data environment.

[1]: Refer to the technical report for more details: https://www.openmp.org/wp-content/uploads/openmp-TR13.pdf

Taskgraph – Graph Generation[2]



With compile-time analysis, depending on the available information, we either:

- Fully generate a graph with its task, data and dependencies
- Partially generate a graph, e.g., edges (dependencies) but not the tasks, as the task data is unknown
- Leave the generation of the graph to the runtime library, which incurs overhead at the first iteration

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

[2]: Implementation details can be found at https://doi.org/10.1109/TPDS.2023.3284219

Evaluation – Set up[3]

	CPU (#cores)	RAM	OS	LLVM Version
Marenostrum 4	2 x Intel Xeon 8160 (48)	96GB	SuSE Linux	LLVM 15.0

- Synthetic kernel
- Heat diffusion (Gauss-Seidel)
- Nbody simulation
- Axpy kernel
- Dot product
- Cholesky decomposition
- HOG object detection
- NAS Parallel Benchmark

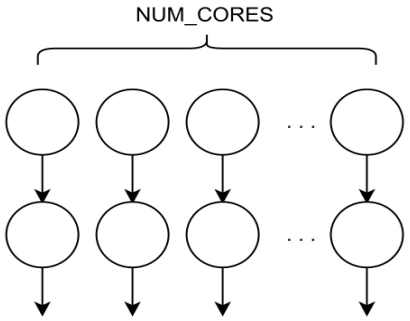
- OMP_NUM_THREADS=#Cores
- KMP_AFFINITY for core affinity
- Fix problem size, varying # tasks
- Optimization flag: -O3

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

[3] : Applications available at https://github.com/yuchenle/tdg-benchs

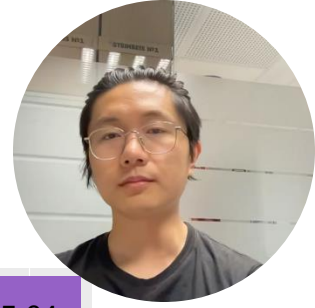
Evaluation – Overhead Reduction

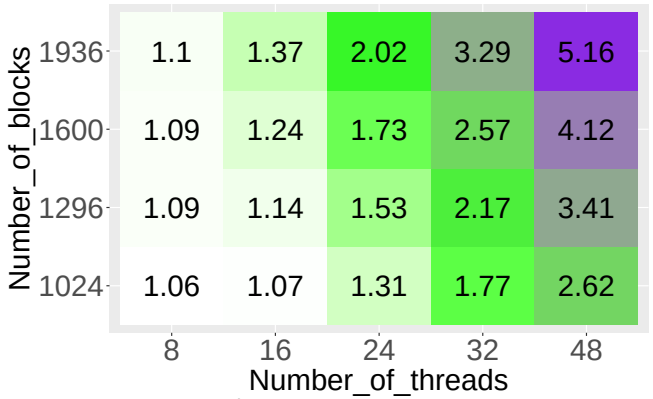
```
int deps[NUM_PROCS];
#pragma omp parallel
#pragma omp single
{
  START_TIMER;
  #pragma omp taskgraph
  {
    for (int i = 0; i < NUM_TASKS; i++) {
      index_depend = i % MAX_PAR;
      #pragma omp task depend(out:deps[index_depend])
      fn();
    }
  }
  END_TIMER;
}
```



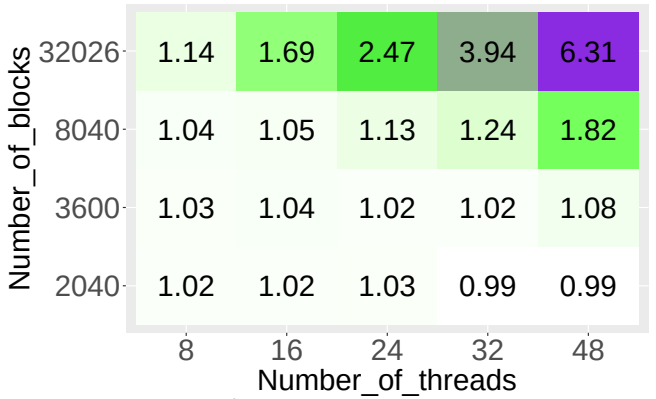
	1	10	10^2	10^3	10^4	10^5
GCC	0.2	0.2	0.25	11.8	345.4	**3582.6**
GCC + Taskgraph	0.2	0.1	0.2	0.2	29.4	**345.3**
LLVM	1.6	0.6	0.6	1.5	36.7	**466.2**
LLVM + Taskgraph	0.2	0.1	0.2	0.3	9.9	**132.2**

Evaluation – Speedup vs *task*

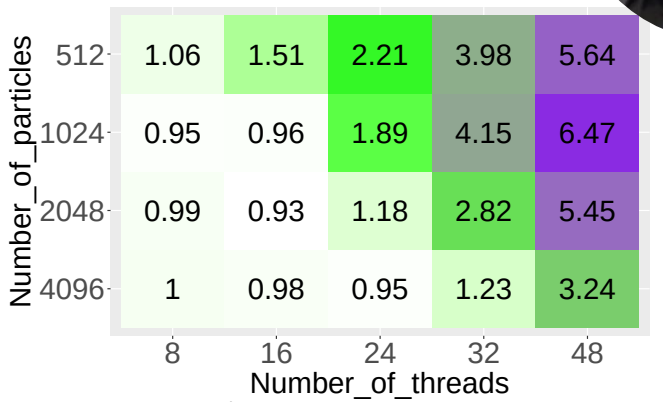




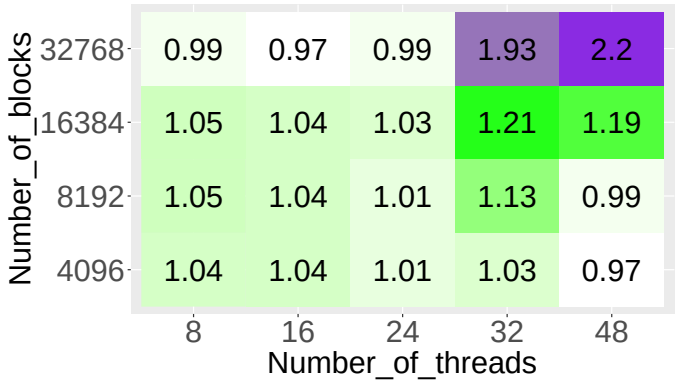
a) Heat dissipation



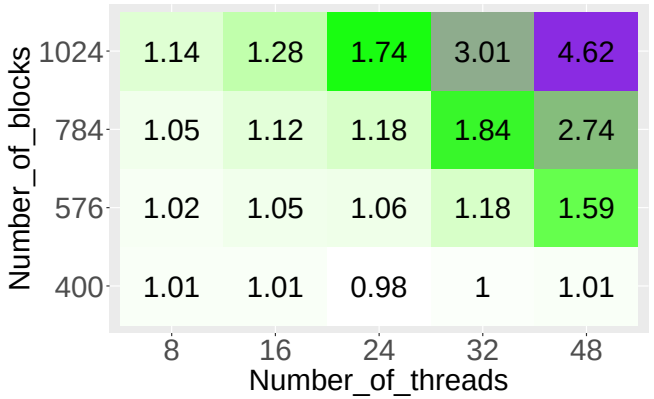
b) Hog object detector



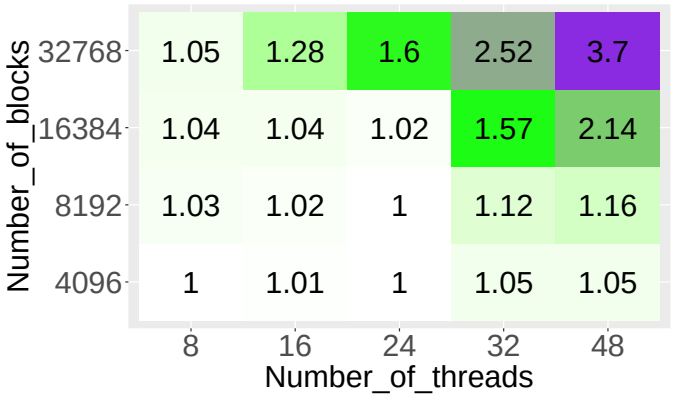
c) N-body simulation



d) SAXPY



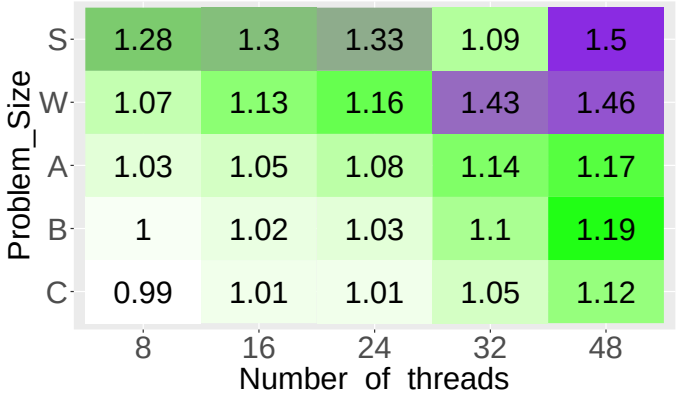
e) Cholesky decomposition



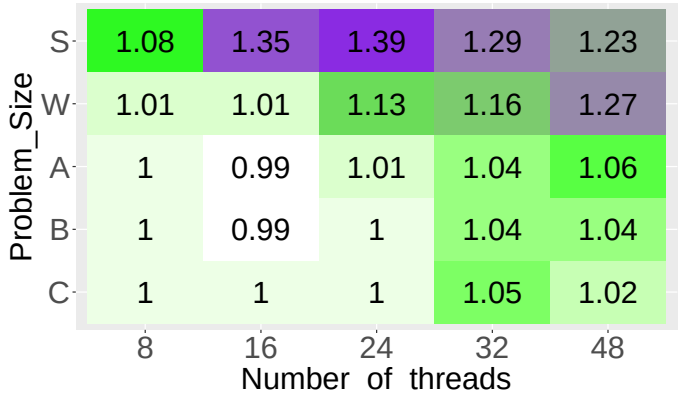
f) Dot product

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

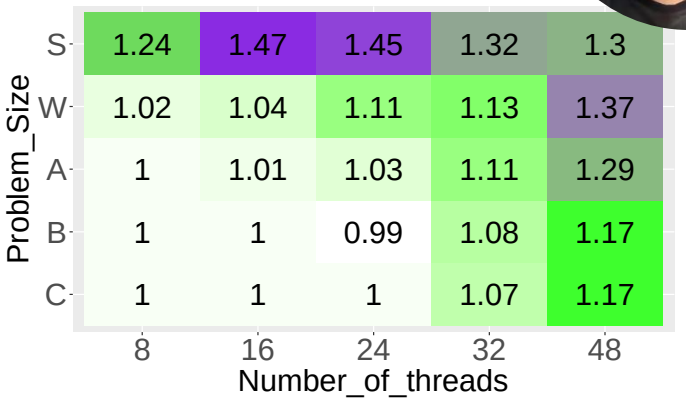
Evaluation – Speedup vs *taskloop*



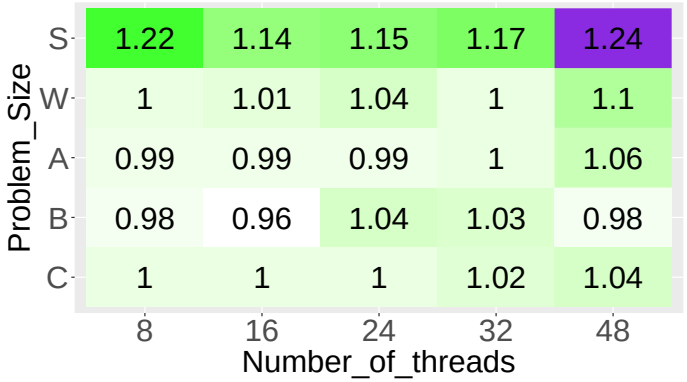
a) NAS CG



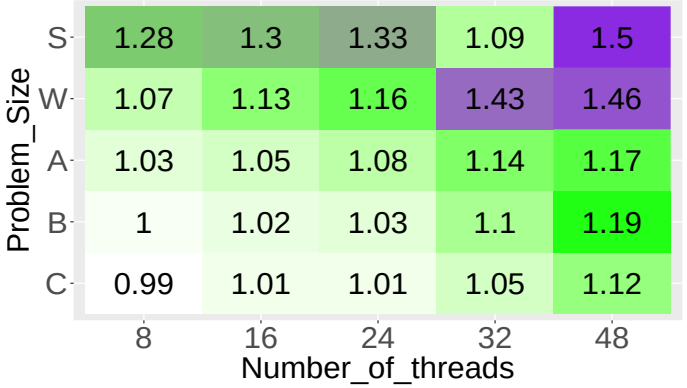
b) NAS BT



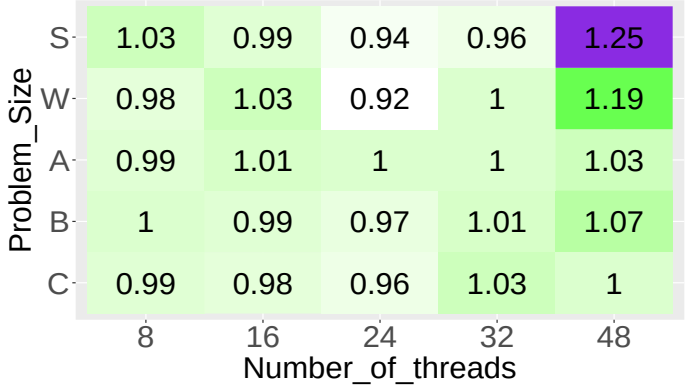
c) NAS SP



d) NAS FT

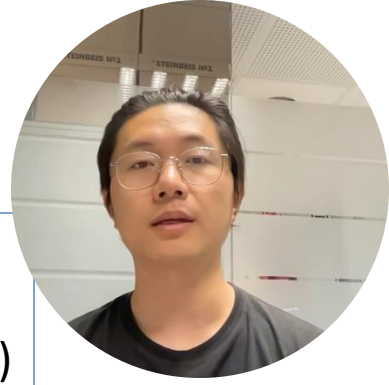


e) NAS LU



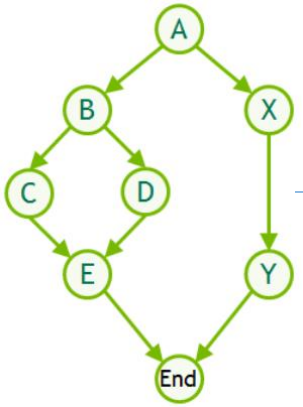
f) NAS EP

Taskgraph – Device Graph

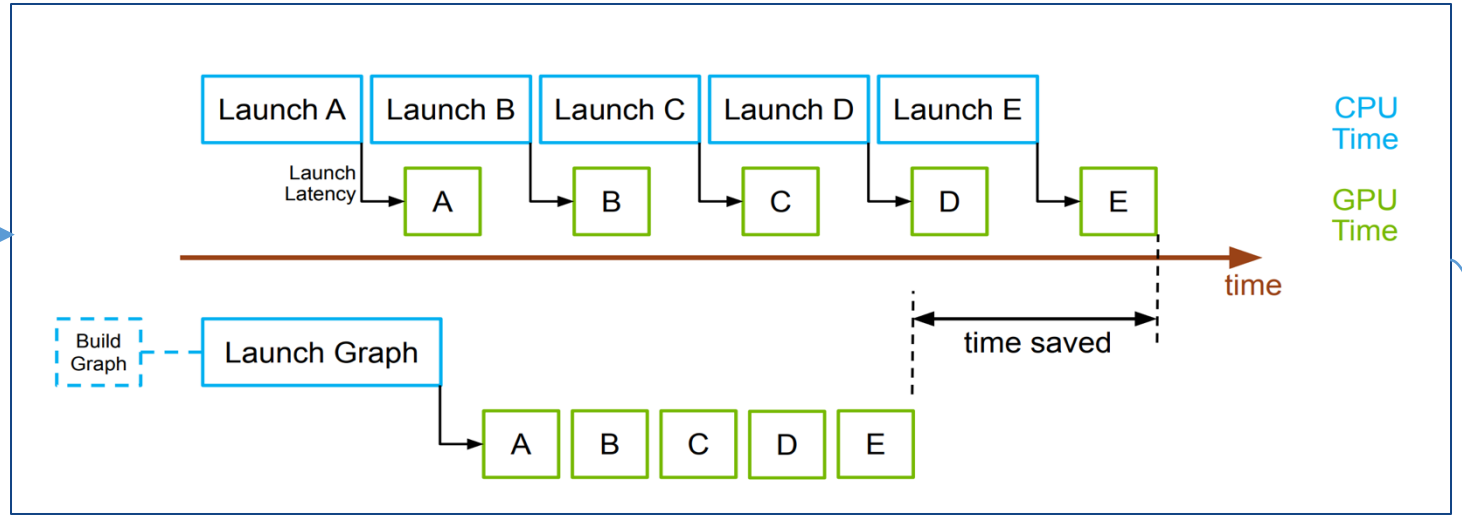


- 9 out of 10 of TOP 10 supercomputers integrate scientific GPU
- GPU kernels are short and launched repetitively (image processing, AI workload, etc)

credit: olcf.ornl.gov



credit:nvidia.com

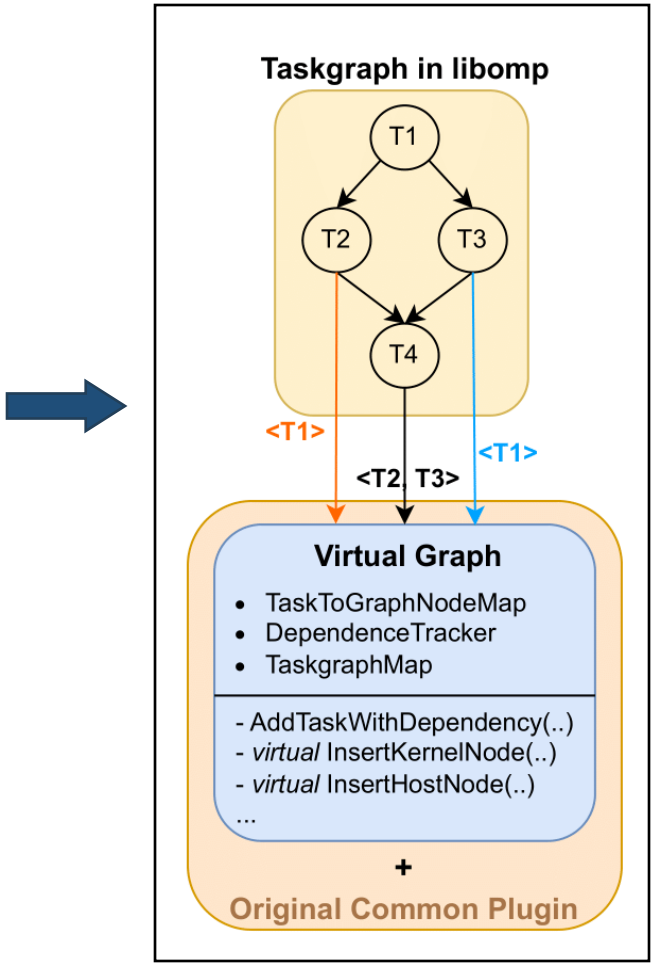


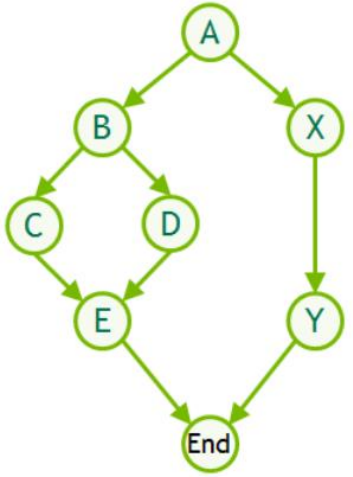
- Stream record API, unsuitable for complex dependency
- Graph API, fine-grained control but verbose:
  - `hipGraphAddKernelNode ( hipGraphNode_t* pGraphNode, … )`
  - `hipGraphAddHostNode ( hipGraphNode_t* pGraphNode, … )`
  - `hipGraphAddMemcpyNode ( hipGraphNode_t* pGraphNode, … )`

How to define a graph, taking HIP as example

Implementation – Device Graph in LLVM

```
int operationLoop(…) {
  #pragma omp parallel
  #pragma omp single
  {
    #pragma omp taskgraph
    {
        #pragma omp task depend(out:op1)
        calibration(…)

        #pragma omp task depend(in:op1)
        fillIntensities(…)

        // other computations

        #pragma omp task depend(in:op_x)
        commandActuators(…)
    }
}
```





credit:nvidia.com

Evaluation – Set up[4]

	CPU (#cores)	GPU	OS	CUDA
Data Center	2 x Intel Xeon 8460 (80)	Nvidia H100	Ubuntu 22.04	V. 12.2
Customer	Intel i7-12700 (20)	Nvidia RTX 4080	Ubuntu 22.04	V. 12.2
Embedded	ARMv8 (12)	Integrated Nvidia GPU	Tegra Linux	V. 11.4

- Gridmini
- TestSNAP
- Neutral
- Points2Image

Proxy applications

- Heat propagation
- Nbody

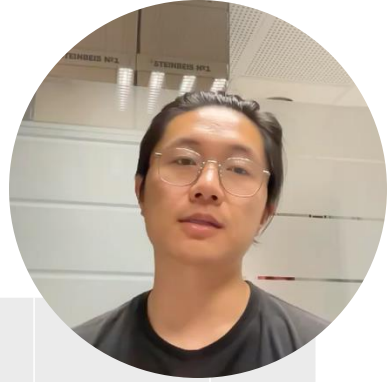
Simulations

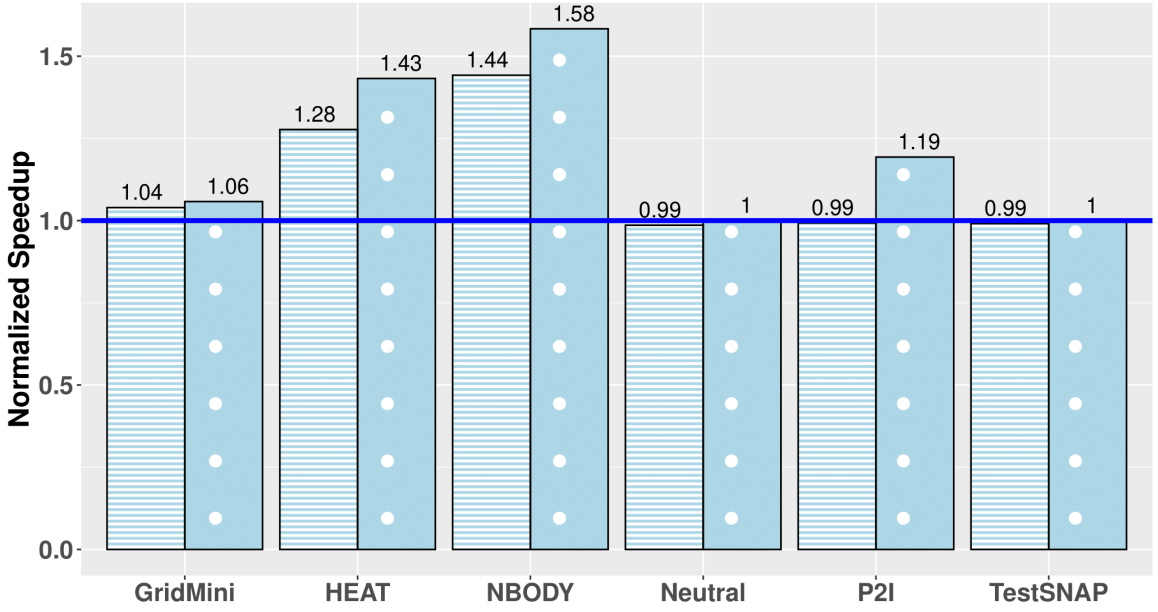
- OMP_NUM_THREADS=#Cores
- KMP_AFFINITY for core affinity
- 32 graph nodes (tasks) for all applications
- Optimization flag: -O3

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

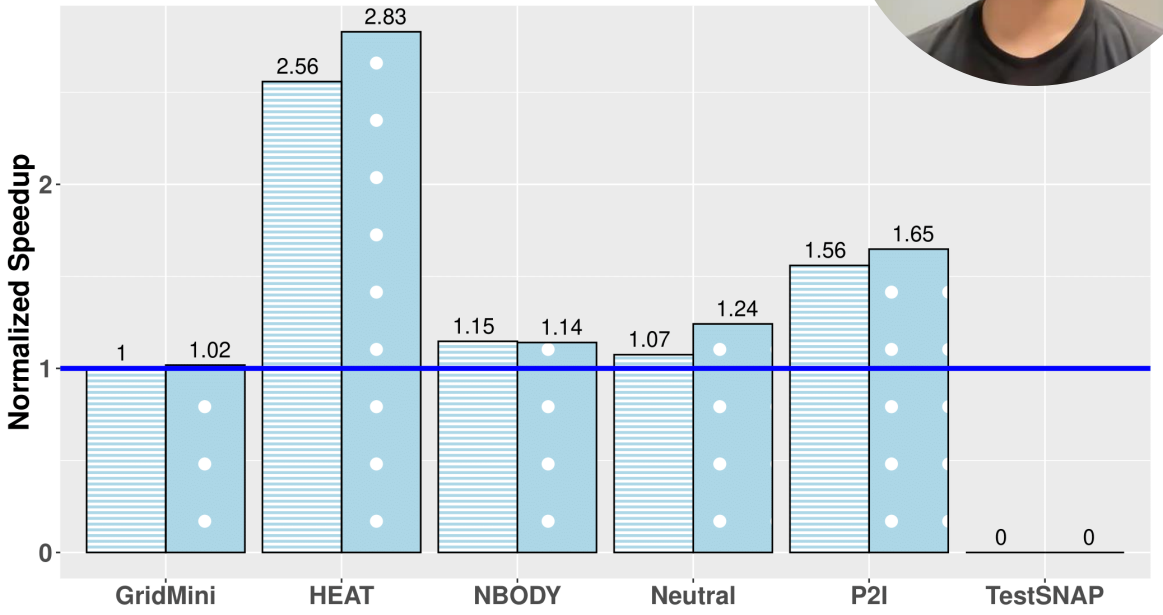
[4] : Refer to https://zenodo.org/records/11634155 for files to replicate the experiments

Evaluation – Mean Time





I7 + RTX 4080



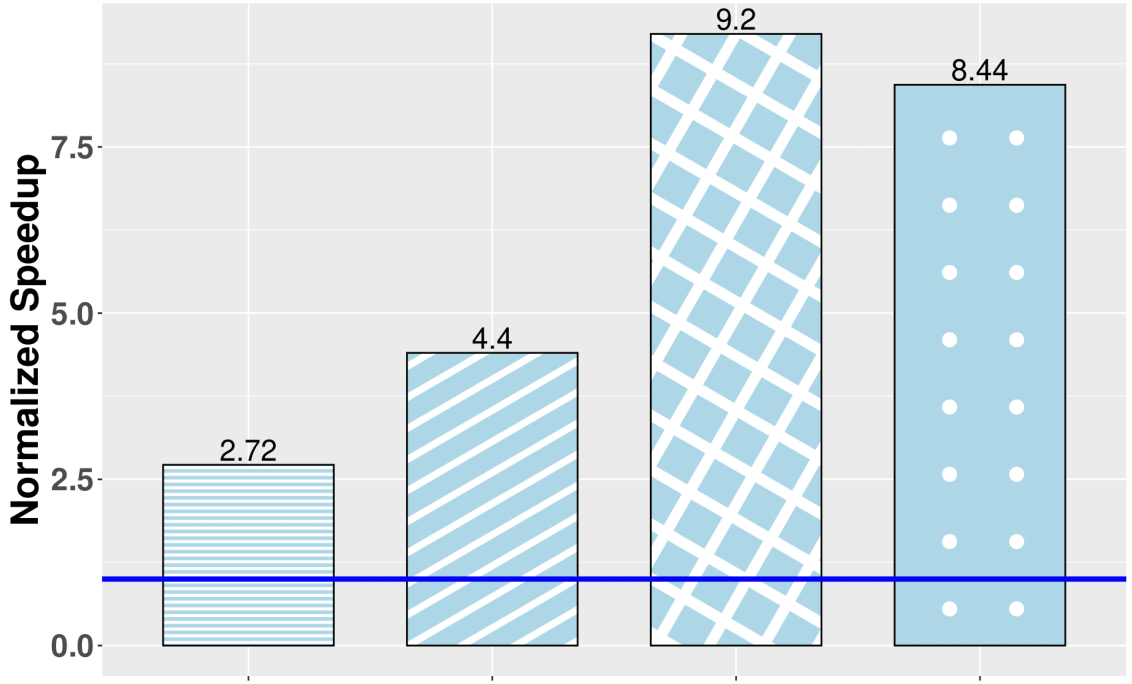
Xeon + H100

The framework is:

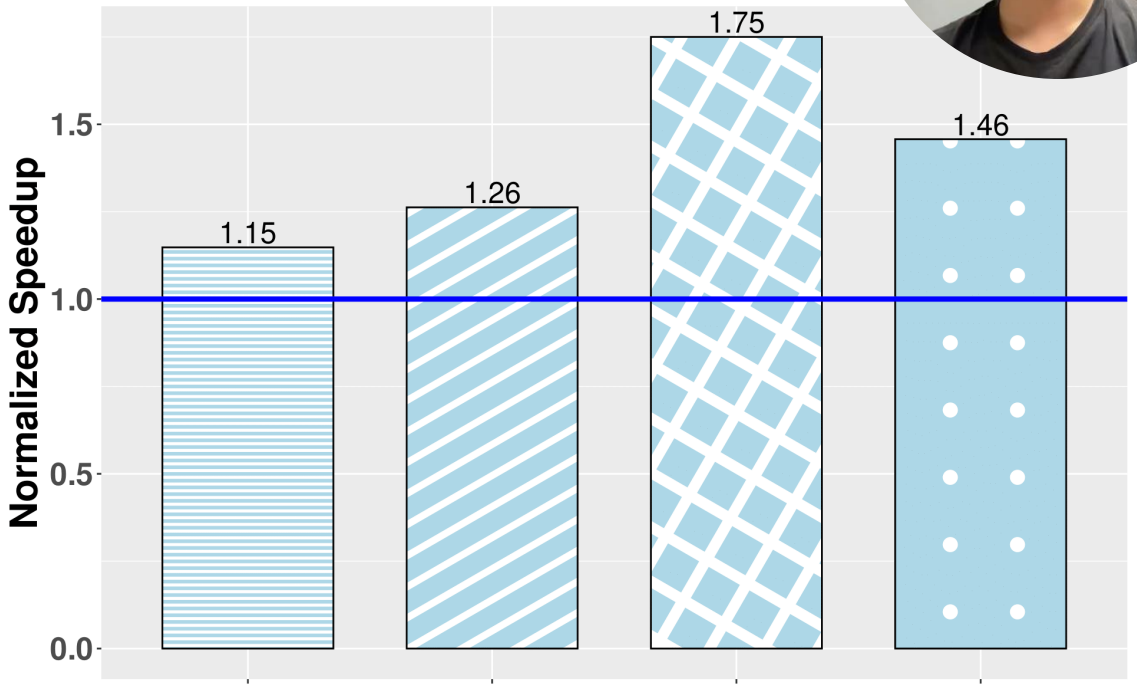
- Generally more efficient than the original accelerator task (*omp target nowait*)
- As efficient as SPMD mode (*omp target teams distribute*) or faster

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Evaluation – Scalability



Heat Propagation on Intel Xeon and H100



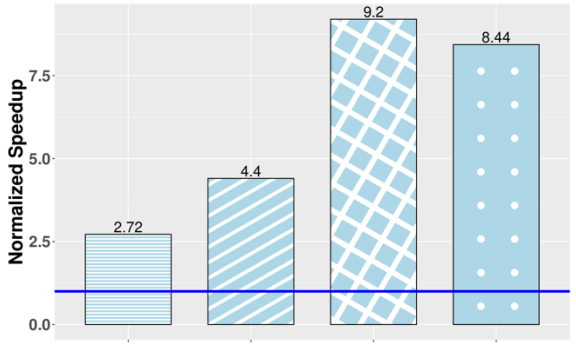
Nbody on Intel Xeon and H100

By varying the number of graph nodes, the framework is demonstrated to be more consistent

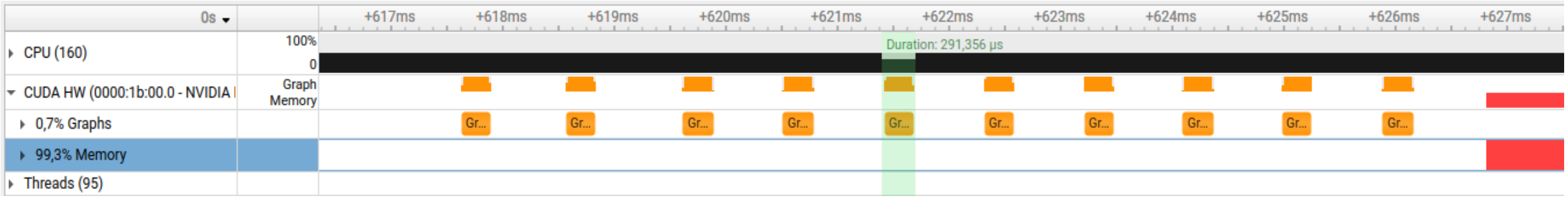
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Evaluation – Scalability

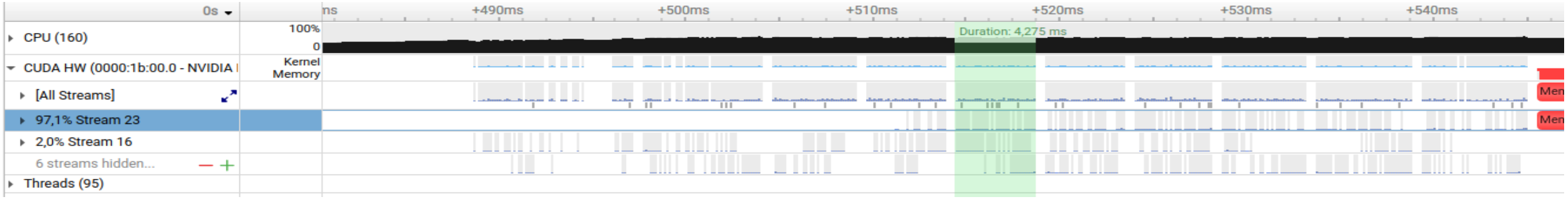
The graph execution is substantially faster but takes longer to instantiate



Heat Propagation on Intel Xeon and H100



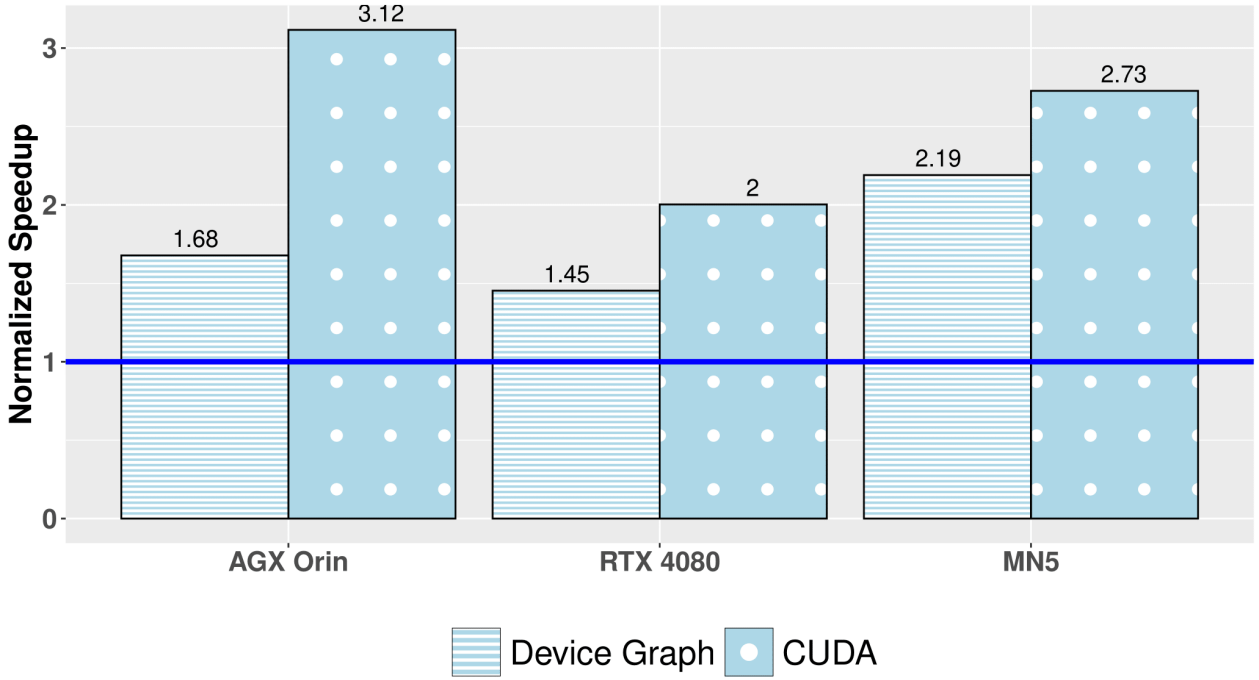
Nsight trace for device graph on Marenostrum 5



Nsight trace for target task on Marenostrum 5

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Evaluation – Versus CUDA



Points2Image benchmark from DAPHNE suite

Competitive in some cases, but mostly falling behind.

However, LLVM OpenMP implementation continuously improves[5], closing the performance gap.

[5] Co-Designing an OpenMP GPU Runtime and Optimizations for Near-Zero Overhead Execution (10.1109/IPDPS53621.2022.00055)

Conclusion

- *taskgraph* directive to be introduced in OpenMP 6.0
- Current results show clear performance gain with fine-grained tasks
- Ready to be mapped to accelerators as device graph
  (e.g., HIP and CUDA graphs)

Future step:

- LLVM upstream implementation
- Tackle current restrictions in OpenMP 6.0

OpenMP®

SC24 Booth Talk Series

openmp.org OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc24talks OpenMP SC24 booth talk videos and presentations