Lukas Gianinazzi (Noéda), Tal Ben-Nun (LLNL), and Torsten Hoefler (SPCL / ETHZ)

Scientific computing needs bandwidth

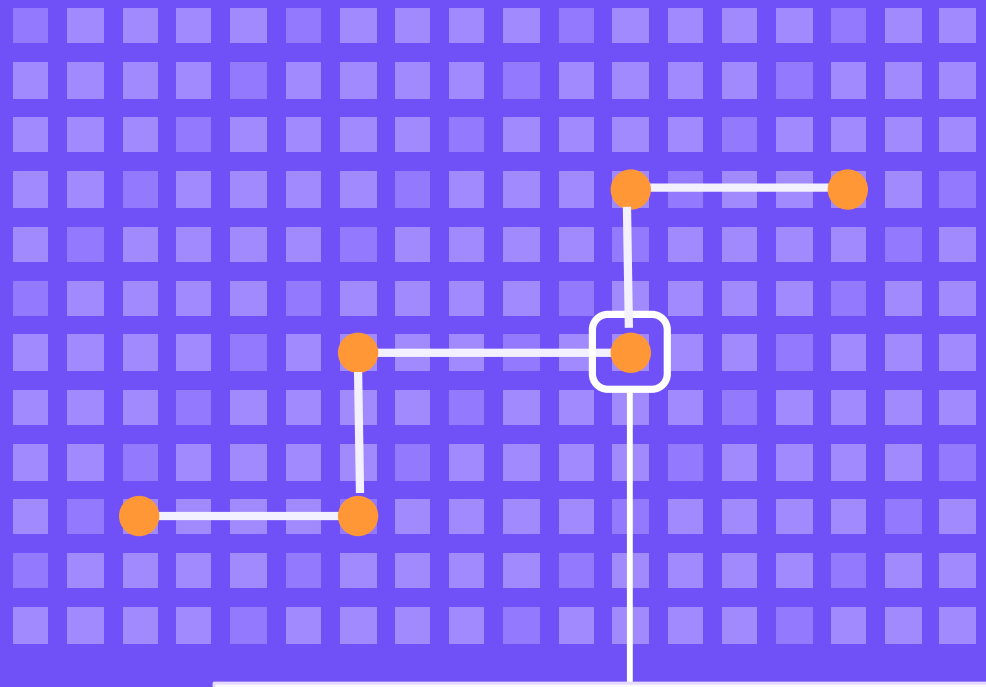
peak on-chip memory bandwidth on Cerebras **WSE-2:**

20 PB/s

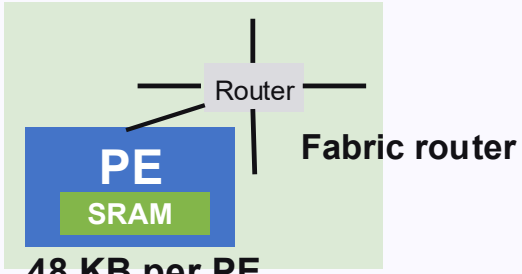
But it comes with a programmability cost:

- **Where data lives** — Explicit placement
- **How it moves** — Routing and channel constraints
- **When it runs** — Asynchronous task scheduling

CEREBRAS WSE-2 / 730K+ PROCESSING ELEMENTS



INSIDE ONE PROCESSING ELEMENT (PE)



SpaDA gives control over spatial algorithm design

The compiler handles the hardware details.

SpaDA

`place` where data lives

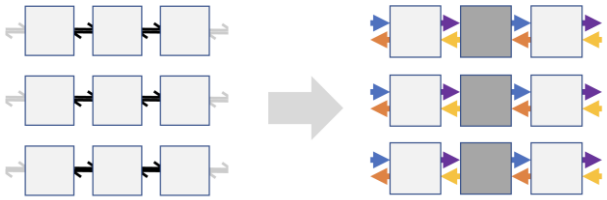
`dataflow` how data moves

`compute` what runs

Also an IR for GT4Py

Optimizing compiler

Conflict-free routing

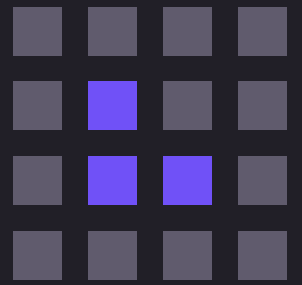


Logical streams — Hardware channels (colors)

Async task scheduling **Vectorization**

Cerebras CSL

Executable



Runs on the WSE

SpaDA makes spatial structure explicit.

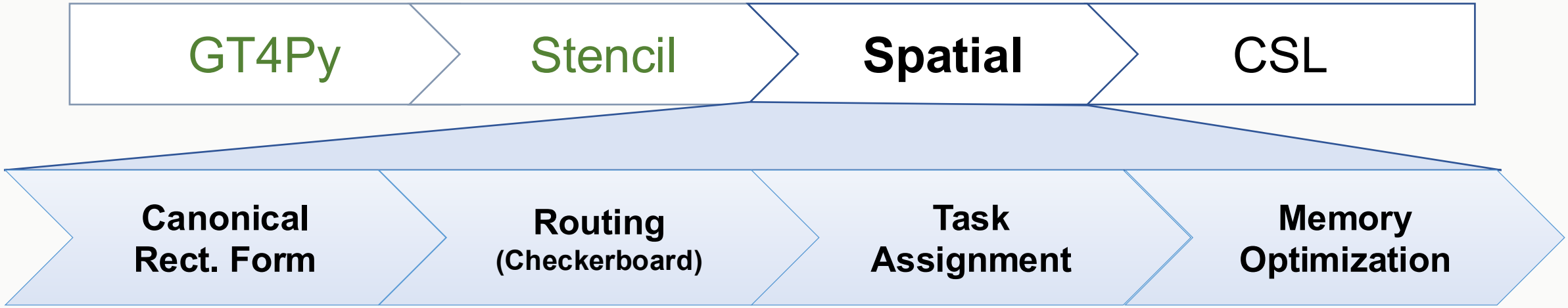
```
kernel @tree_reduce<L, K>(stream<f32, K>[1<<L, 1] readonly a_in,
                          stream<f32, K>[1, 1] writeonly out) {
  place i16 i, i16 j in [0:1<<L, 0] {
    f32[K] loc;
  }
  phase {  // Phase 1: All PEs load input
     // … ommited
  }
  // L bottom-up reduction stages
  for i16 stage in [0:L] {  phase {
    dataflow i16 i, i16 j in [0:1<<L:1<<stage, 0] {
      stream<f32> w = relative_stream(-(1<<stage), 0)
        { hops = auto, channel = stage };
    }
    compute i16 i, i16 j in [0:1<<L:1<<(stage+1), 0] {
      await foreach i16 k, f32 x in [0:K], receive(w) {
          loc[k] = loc[k] + x; }
    }
    compute i16 i, i16 j in [1<<stage:1<<L:1<<(stage+1), 0] {
       await send(loc, w); }
    }
  }
}
```

place

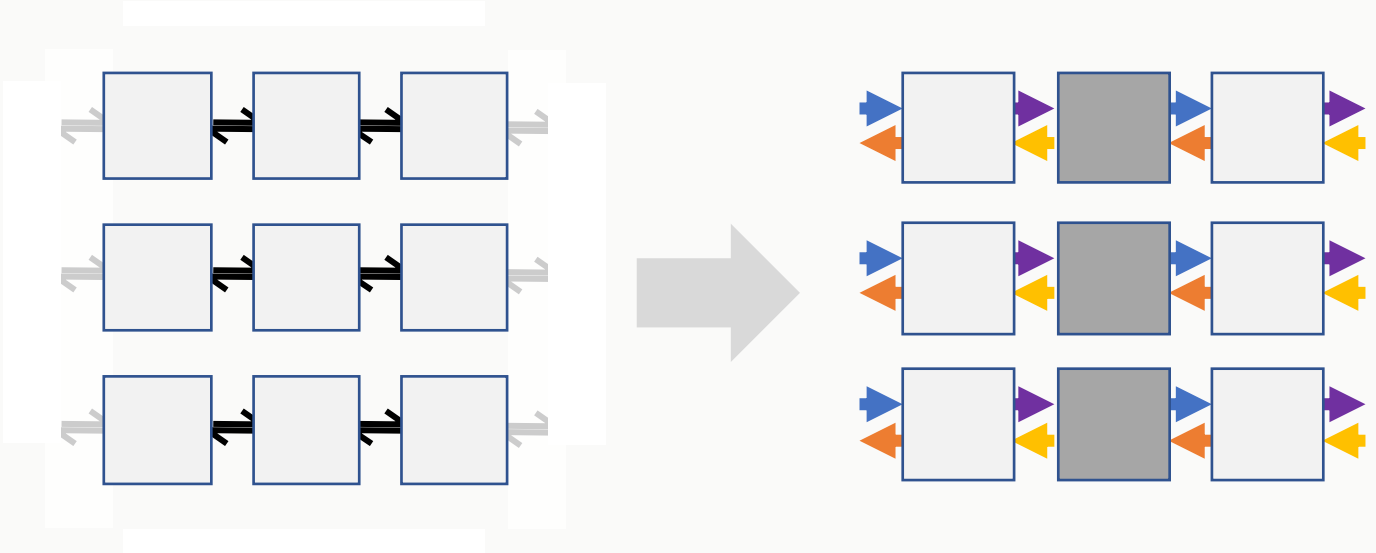
dataflow

compute

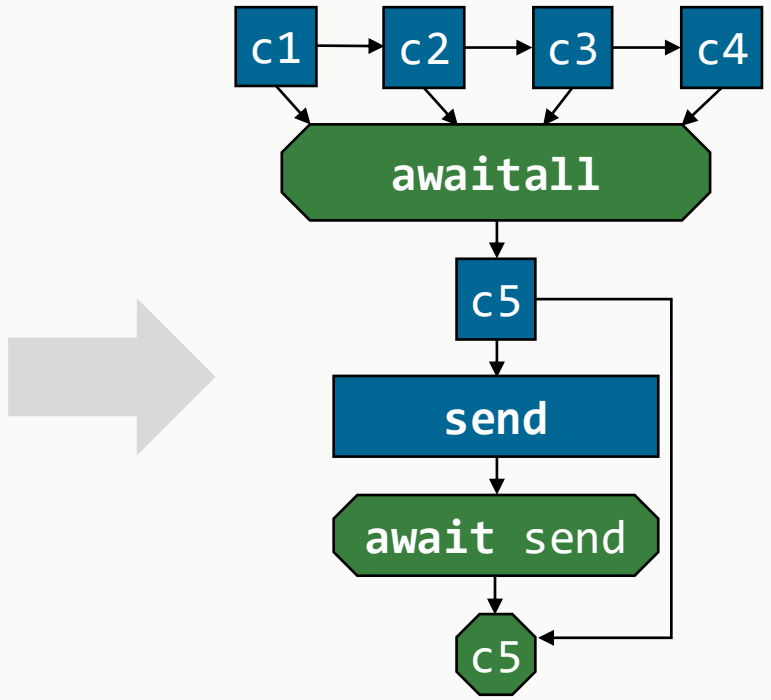
A multi-stage lowering pipeline

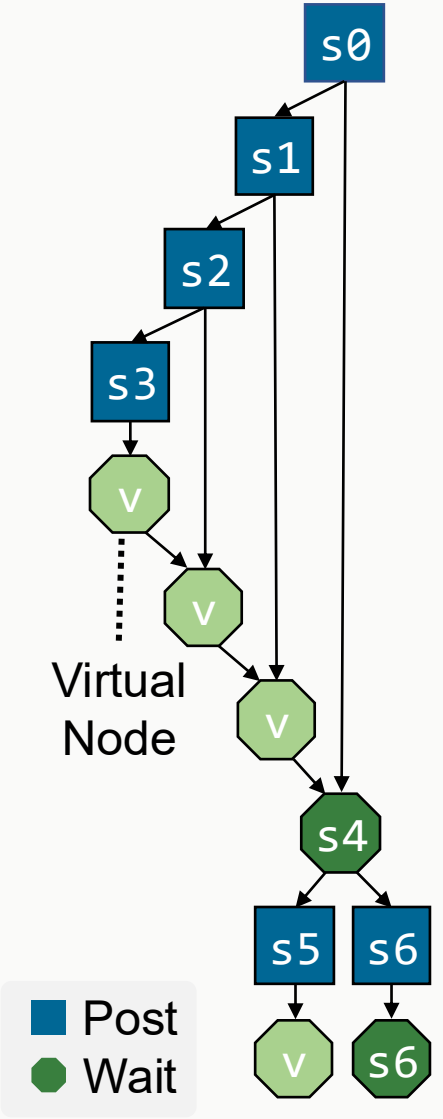


Assigning channels without conflicts.

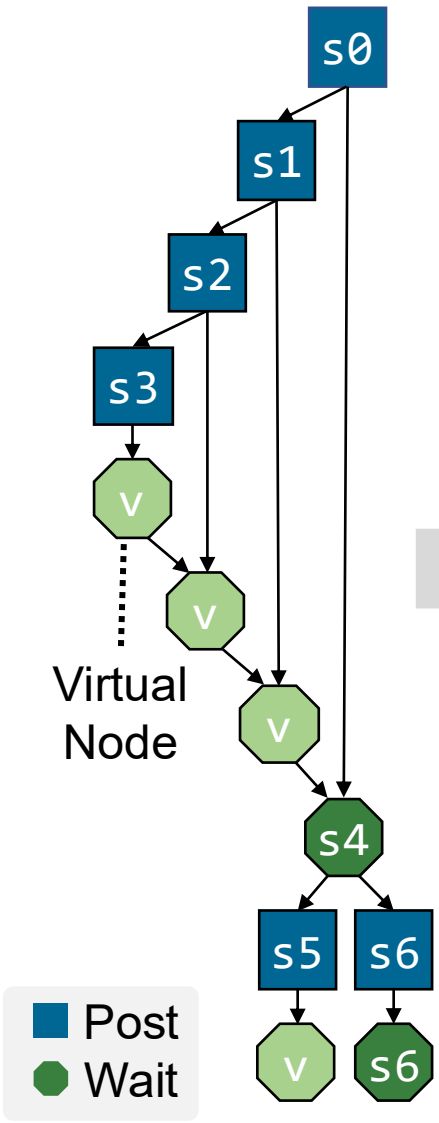


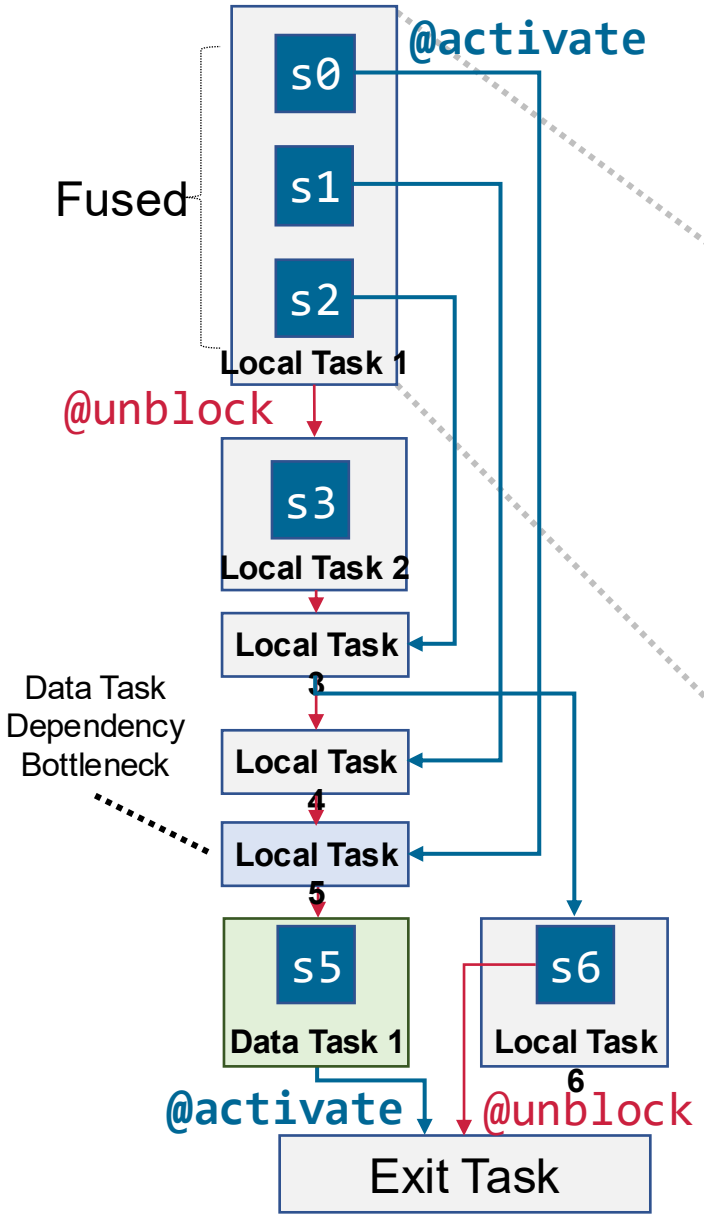
```
compute i16 i, i16 j in [...] {
  completion c1 = ...     // s0
  completion c2 = ...     // s1
  completion c3 = ...     // s2
  completion c4 = ...     // s3
  awaitall;               // s4
  completion c5 = ...     // s5
  await send(arr, strm); // s6
}
```





SpaDA **Completion DAG** **Post-Wait Graph**






```
...
task task_1() {
    ...
    // s2
    @fmach(a_dsd, b_dsd_in, -4.0,
           .async = true,
           .activate = task_3_id);
    // Terminator
    @unblock(task_2_id);
}
...
```

Post-Wait Graph

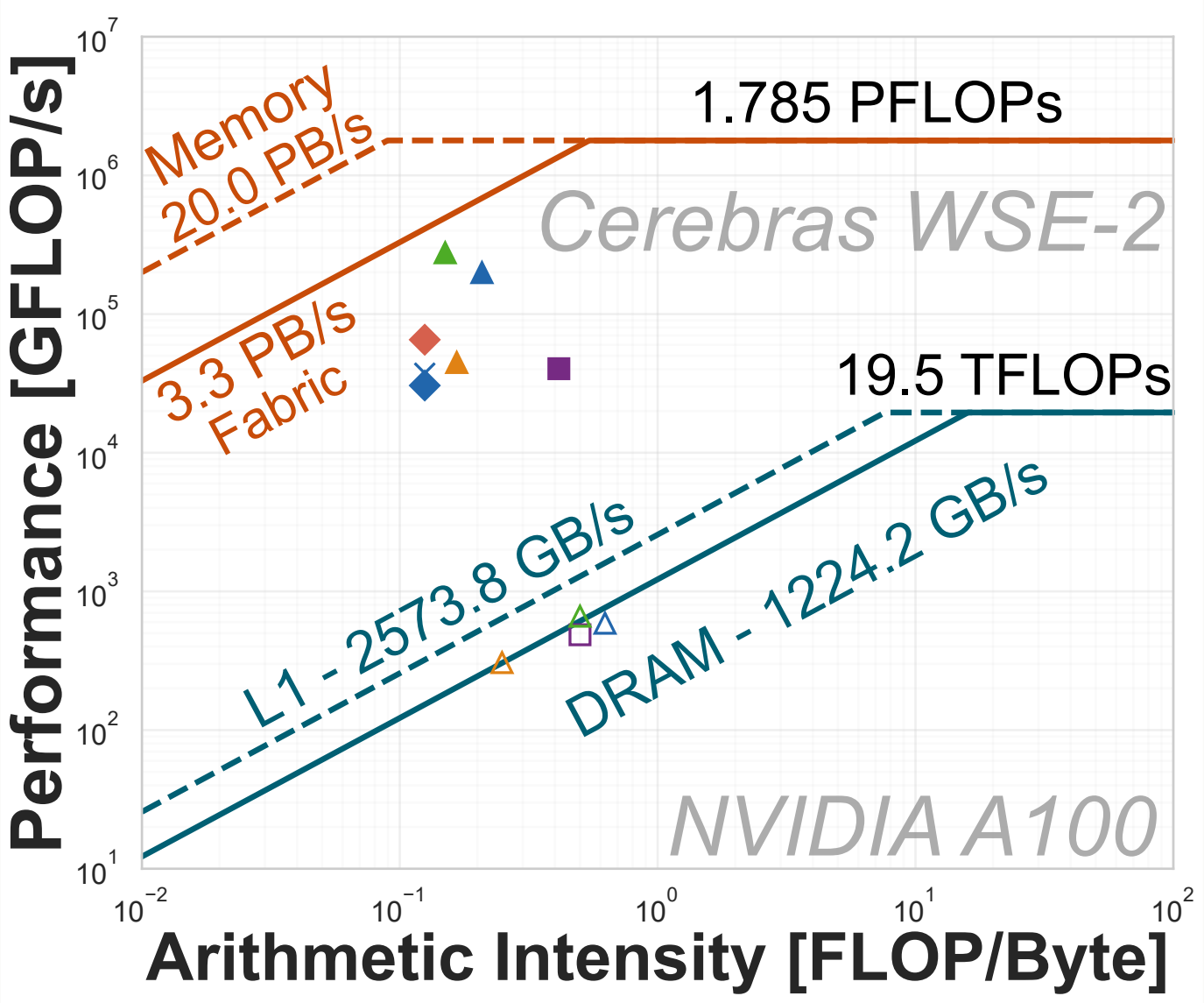
WSE-Specialized Task Graph

Resulting CSL Code

Dense Linear Algebra
- ■ SpaDA GEMV (Two-phase)
- □ A100 GEMV CUBLAS

Communication Collectives
- ◆ SpaDA Chain Reduce (512×512)
- ◆ SpaDA Chain Reduce (750×994)
- × Chain Reduce (HPDC24, 512×512)

Multi-Dimensional Stencils
- ▲ SpaDA 2D Laplacian
- ▲ SpaDA UVBKE
- ▲ SpaDA Vertical Stencil
- △ A100 2D Laplacian
- △ A100 UVBKE
- △ A100 Vertical Stencil



Less code,

Near-handwritten performance.

4.68–13.13×

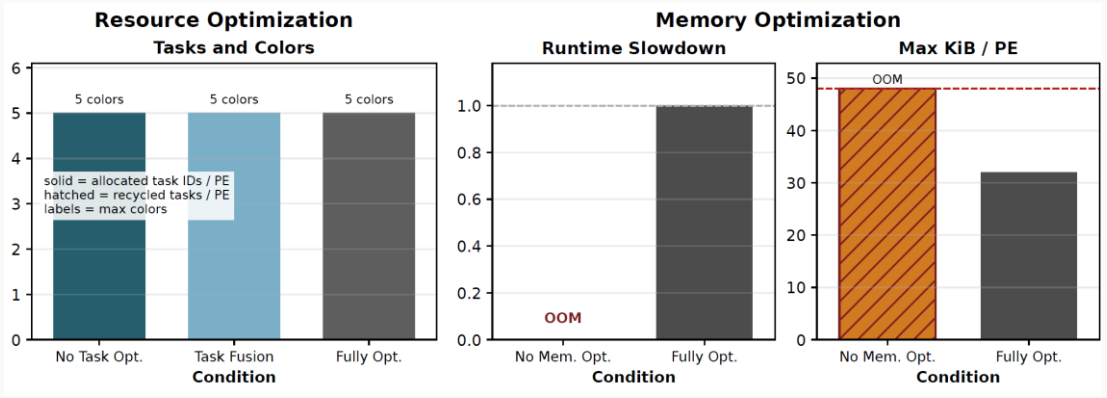
fewer lines of code

1.04×

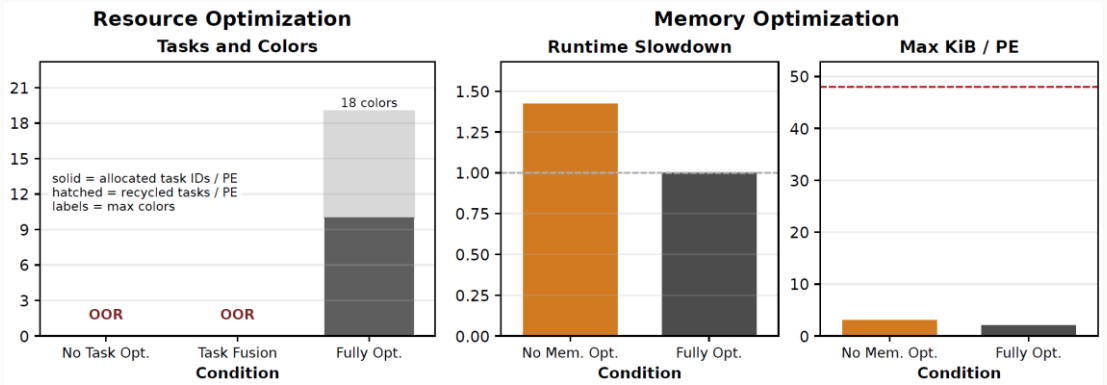
runtime vs. handwritten CSL*

"Programming spatial dataflow architectures without giving up performance."

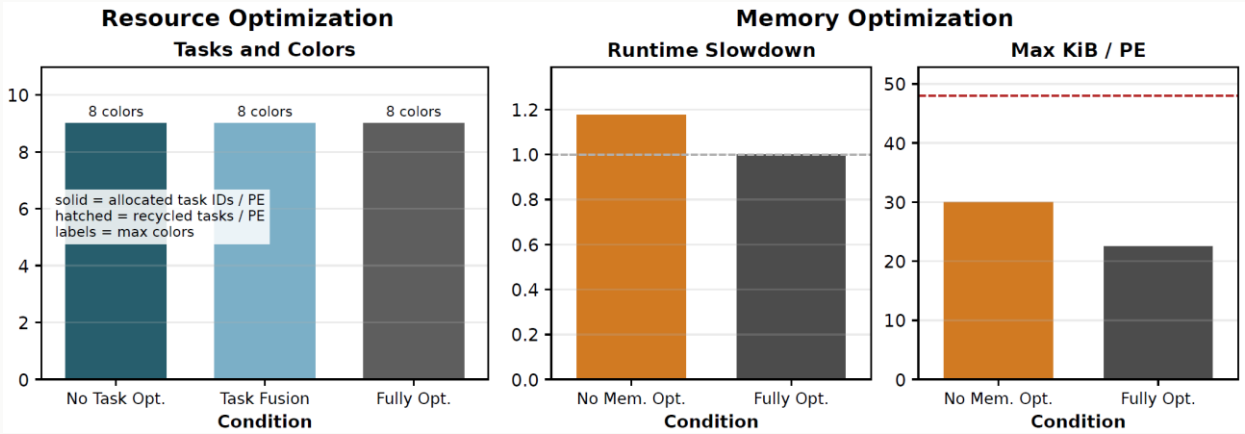
* for Communication Collectives Benchmarks



Two-Phase Reduction (2D)



Tree Reduction (2D)



UVBKE Stencil

48 KiB / PE — Stats before compile failure