Automated Mapping of Task-Based Programs onto Distributed and Heterogeneous Machines

Thiago Teixeira

Alexandra Henzinger

Rohan Yadav

Alex Aiken

SC'23 at Denver, CO
Nov 14

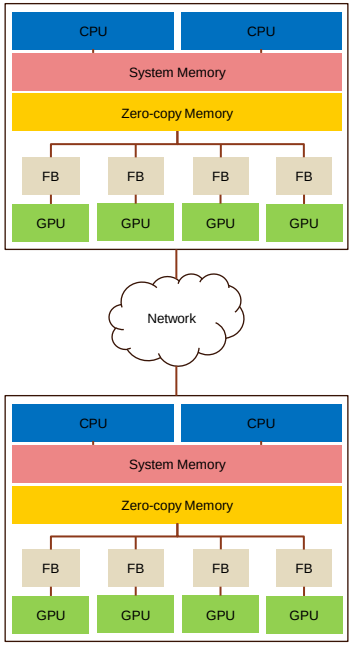
https://gitlab.com/thiagotei/automap

Stanford University

Mapping on Existing Heterogeneous Systems

```
local function make_stencil(radius)
  local __demand(__cuda)
    task stencil(private : region(ispace(int2d), point),
                                        interior : region(ispace(int2d), point),
                                        xm : region(ispace(int2d), point),
                                        xp : region(ispace(int2d), point),
                                        ym : region(ispace(int2d), point),
                                        yp : region(ispace(int2d), point),
                                        times : region(ispace(int1d), timestamp),
                                        print_ts : bool)
    where
                                        reads writes(private.{input, output}, times),
                                        reads(xm.input, xp.input, ym.input, yp.input)
    do
      if print_ts then
        var t = c.legion_get_current_time_in_micros()
        for x in times do x.start = t end
      end
      var interior_rect = get_rect(interior.ispace)
      var interior_lo = int2d { x = interior_rect.lo.x[0], y = interior_rect.lo.x[1] }
      var interior_hi = int2d { x = interior_rect.hi.x[0], y = interior_rect.hi.x[1] }
      var xm_rect = get_rect(xm.ispace)
      var xm_lo = int2d { x = xm_rect.lo.x[0], y = xm_rect.lo.x[1] }
      var xp_rect = get_rect(xp.ispace)
      var xp_lo = int2d { x = xp_rect.lo.x[0], y = xp_rect.lo.x[1] }
      var ym_rect = get_rect(ym.ispace)
      var ym_lo = int2d { x = ym_rect.lo.x[0], y = ym_rect.lo.x[1] }
      var yp_rect = get_rect(yp.ispace)
      var yp_lo = int2d { x = yp_rect.lo.x[0], y = yp_rect.lo.x[1] }
      for i in xm do
        var i2 = i - xm_lo + interior_lo + { -radius, 0 }
        private[i2].input = xm[i].input
      end
      for i in ym do
        var i2 = i - ym_lo + interior_lo + { 0, -radius }
        private[i2].input = ym[i].input
      end
      for i in xp do
        var i2 = i - xp_lo + { x = interior_hi.x + 1, y = interior_lo.y }
        private[i2].input = xp[i].input
      end
      for i in yp do
        var i2 = i - yp_lo + { x = interior_lo.x, y = interior_hi.y + 1 }
        private[i2].input = yp[i].input
      end
      [make_stencil_interior(private, interior, radius)]
    end
    return stencil
end
local stencil = make_stencil(RADIUS)
```

Application



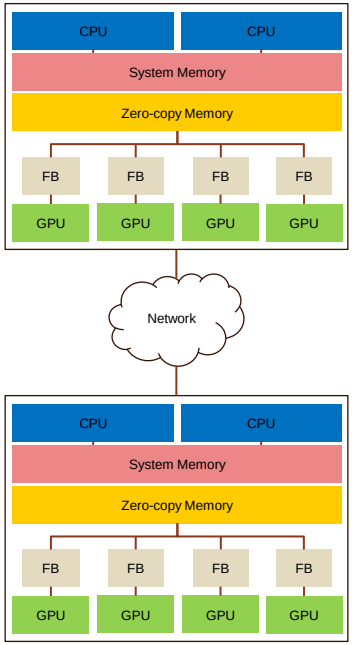
Mapping on Existing Heterogeneous Systems

```
local function make_stencil(radius)
  local __demand(__cuda)
    task stencil(private : region(ispace(int2d), point),
                                      interior : region(ispace(int2d), point),
                                      xm : region(ispace(int2d), point),
                                      xp : region(ispace(int2d), point),
                                      ym : region(ispace(int2d), point),
                                      yp : region(ispace(int2d), point),
                                      times : region(ispace(int1d), timestamp),
                                      print_ts : bool)
    where
                                      reads writes(private.{input, output}, times),
                                      reads(xm.input, xp.input, ym.input, yp.input)
    do
      if print_ts then
        var t = c.legion_get_current_time_in_micros()
        for x in times do x.start = t end
      end
      var interior_rect = get_rect(interior.ispace)
      var interior_lo = int2d { x = interior_rect.lo.x[0], y = interior_rect.lo.x[1] }
      var interior_hi = int2d { x = interior_rect.hi.x[0], y = interior_rect.hi.x[1] }
      var xm_rect = get_rect(xm.ispace)
      var xm_lo = int2d { x = xm_rect.lo.x[0], y = xm_rect.lo.x[1] }
      var xp_rect = get_rect(xp.ispace)
      var xp_lo = int2d { x = xp_rect.lo.x[0], y = xp_rect.lo.x[1] }
      var ym_rect = get_rect(ym.ispace)
      var ym_lo = int2d { x = ym_rect.lo.x[0], y = ym_rect.lo.x[1] }
      var yp_rect = get_rect(yp.ispace)
      var yp_lo = int2d { x = yp_rect.lo.x[0], y = yp_rect.lo.x[1] }
      for i in xm do
        var i2 = i - xm_lo + interior_lo + { -radius, 0 }
        private[i2].input = xm[i].input
      end
      for i in ym do
        var i2 = i - ym_lo + interior_lo + { 0, -radius }
        private[i2].input = ym[i].input
      end
      for i in xp do
        var i2 = i - xp_lo + { x = interior_hi.x + 1, y = interior_lo.y }
        private[i2].input = xp[i].input
      end
      for i in yp do
        var i2 = i - yp_lo + { x = interior_lo.x, y = interior_hi.y + 1 }
        private[i2].input = yp[i].input
      end
      [make_stencil_interior(private, interior, radius)]
    end
    return stencil
end
local stencil = make_stencil(RADIUS)
--
```

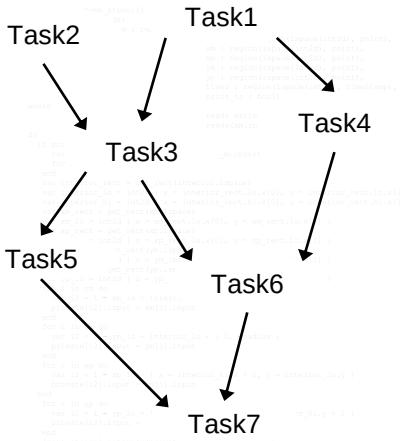
Application

Mapping assigns:

- computation to processors
- data to memories



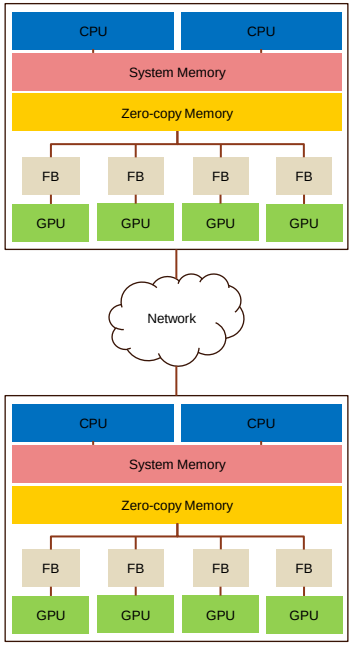
Mapping on Existing Heterogeneous Systems



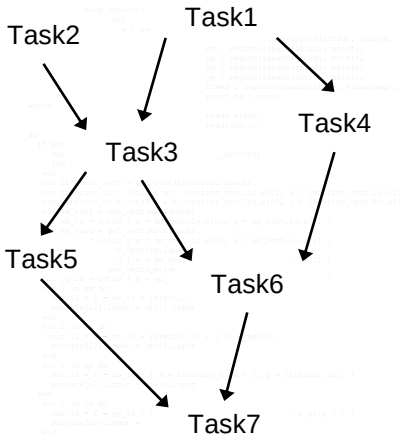
Application

Mapping assigns:

- cc tasks on to processors
- data to memories



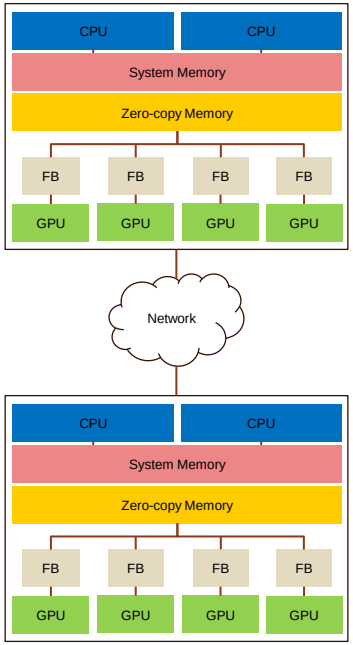
Mapping on Existing Heterogeneous Systems



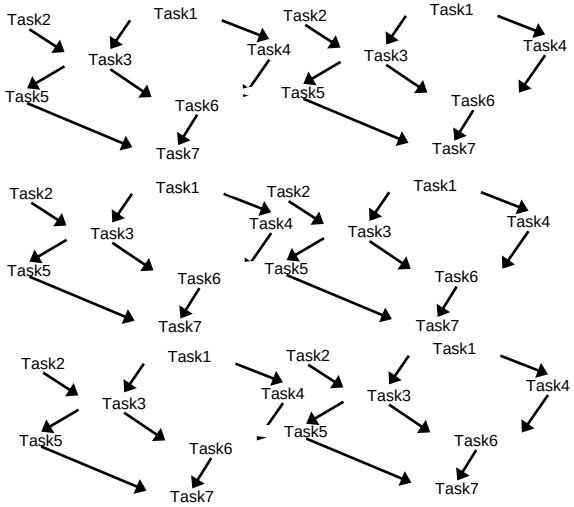
Application

Mapping assigns:

- tasks to processors
- collections to memories



Mapping on Existing Heterogeneous Systems



Application

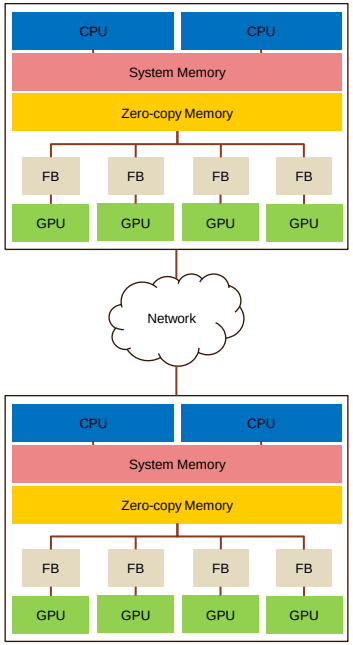
2^{128} possible mappings!!!

Input- and machine-specific!

Hard to maintain!

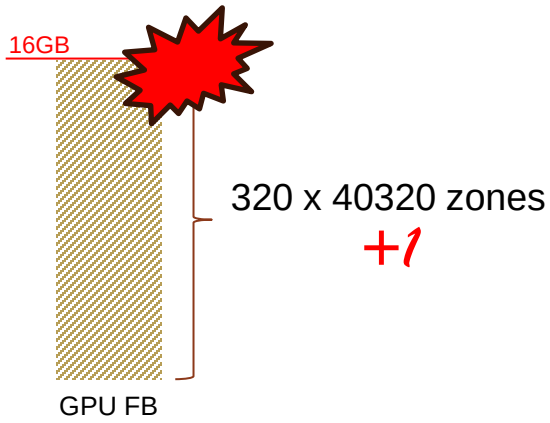
Previous works considered assignment *only* for tasks

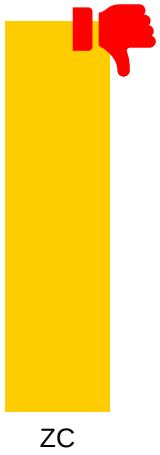
We consider *multiple* possible assignments for data as well!

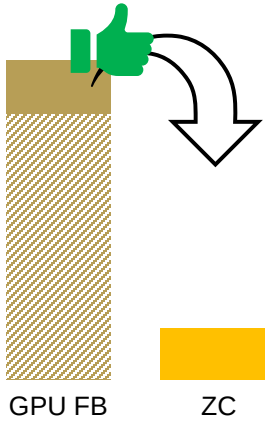


Concrete Example: Simulation OOM

Users may want to run simulations with *bigger* input than what fits in memory

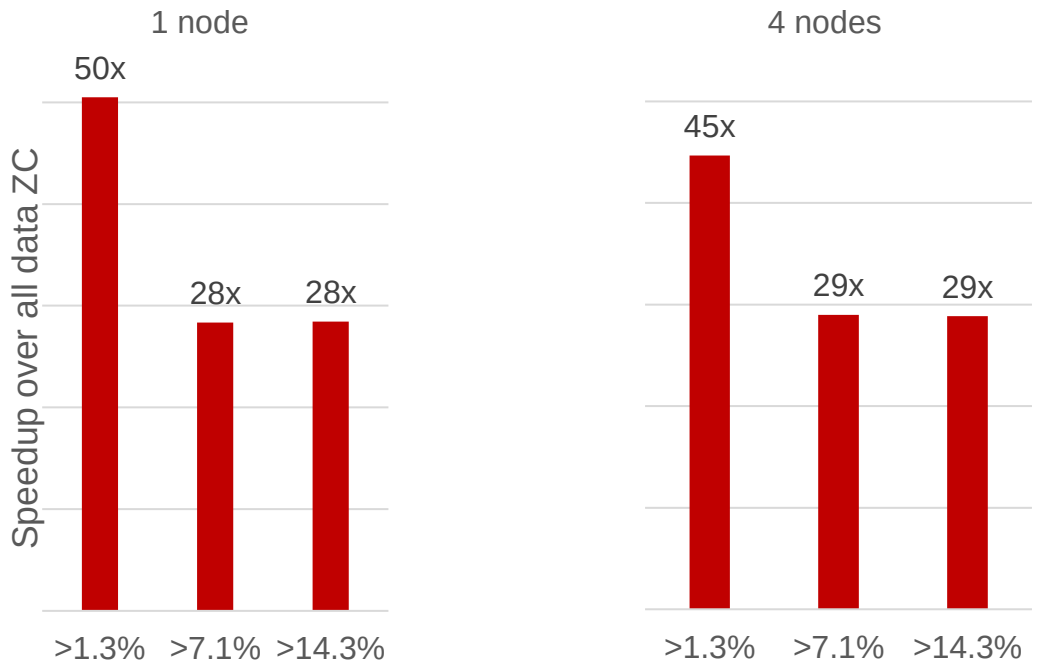






What collections to move off of frame-buffer?

Naïve offloading vs AutoMap



>x% means x% more zones than will fit in Frame-buffer

Stanford University

AutoMap

Automate the discovery of good mappings

- Traverses the space of possible mappings and provides the fastest found
- Used in an offline search that tests different mappings

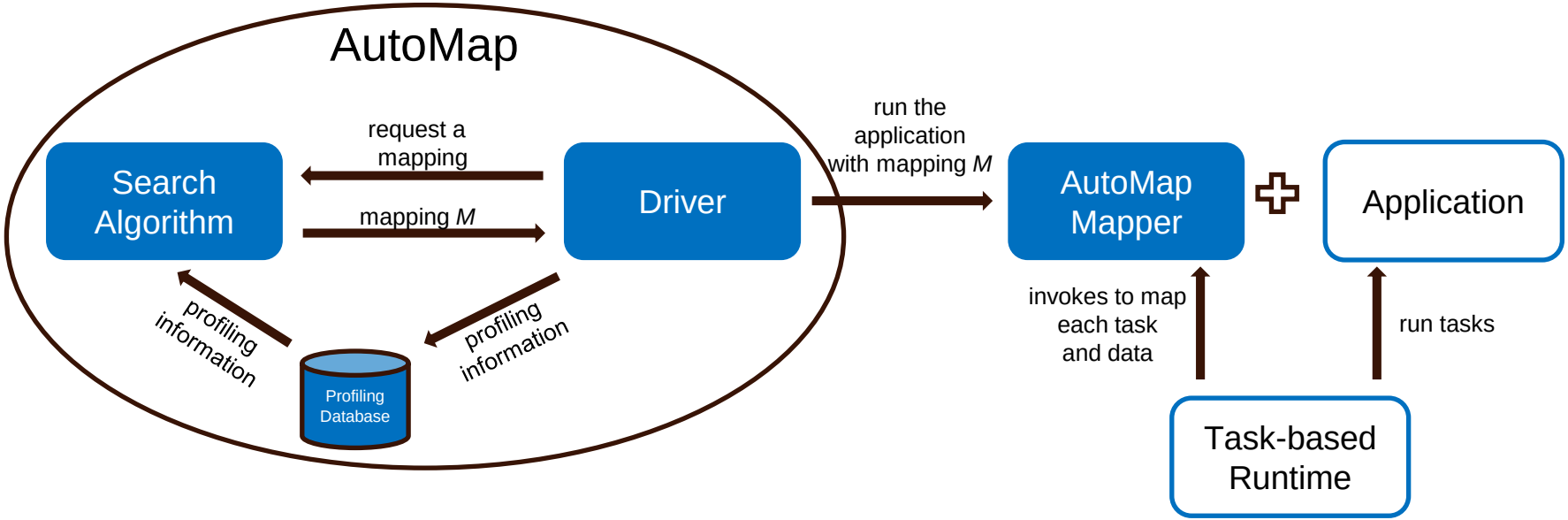
Discrete Search Space:

- Processors kinds: {GPU, OMP} per task
- Memory kinds: {Frame-buffer, Zero-copy, System} per data collection argument

Goals:

- Performance: find better mappings than humans
- Portability: tune mappings to an architecture and/or input
- Productivity: reduce manual work and help non-experts

AutoMap's Workflow



Optimization Problem

Search space can be immense

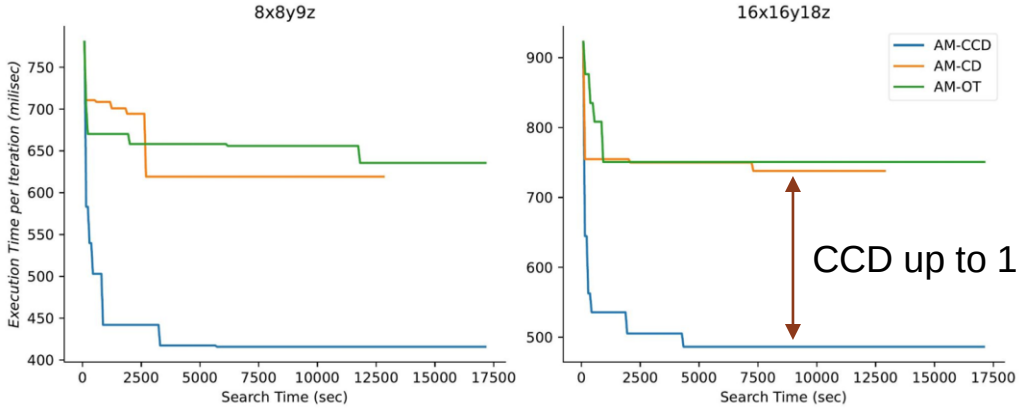
- $O(P^T M^C)$ where *P* is the number of processor kinds, *T* is the number of tasks, *C* is the number of collections arguments, and *M* is the number of memory kinds

Could we solve this with a generic optimization algorithm?

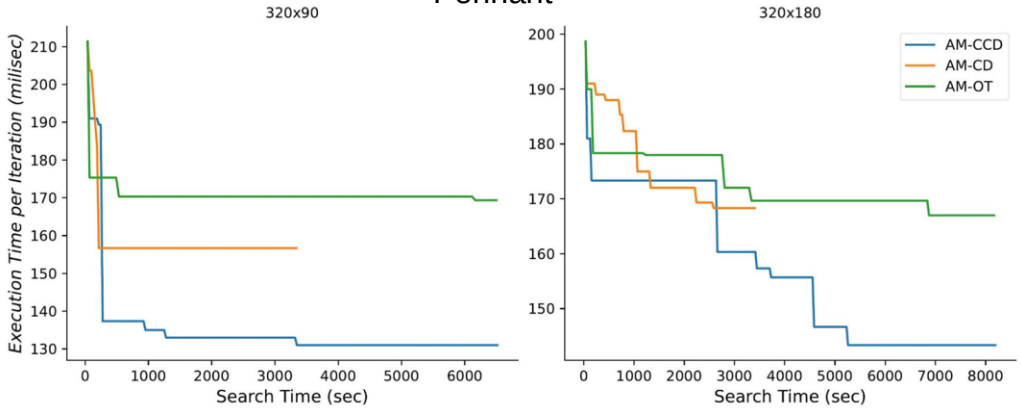
- Evolutionary mutation techniques, differential evolution, Nelder-Mead search, Torczon hillclimbers…

Search Algorithms Comparison

HTR



Pennant



OpenTuner spends up to ~90% of the search time suggesting invalid mappings!

Why a Custom Algorithm?

Aware of the relationship between mappings tasks and its collection arguments

Explore mapping to the same memory collection arguments that refer partially or totally to the same data regions (overlap)

Constrained Coordinate Descent (CCD)

CD optimizes one dimension at a time avoiding the combinatorial explosion of possibilities in high dimensional spaces

$$O(P^T M^C) \rightarrow O(PTCM)$$

CCD is multiple executions (i.e., *rotations*) of the CD with different levels of colocation constraints

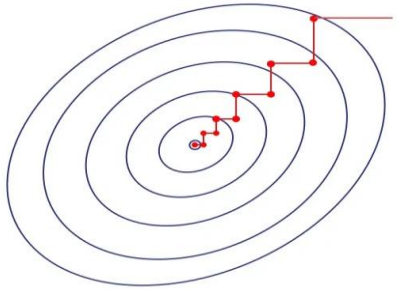
Coordinate Descent (CD) Search Algorithm

Starting point: mapping with all tasks on GPUs, all collections on Frame-buffer

CD loops over:

- Tasks from longest running to shortest
- Collection arguments from largest to smallest

CD makes one move at a time



Two possible moves:

- Move one task to a different processor (and its arguments to new memories)
- Move one collection argument to a different memory kind

Co-location Constraints for Coordinate Placement

Enforces overlapping collections arguments onto the same memory

Without constraints it is *unlikely* to explore mappings where overlaping collections are co-located

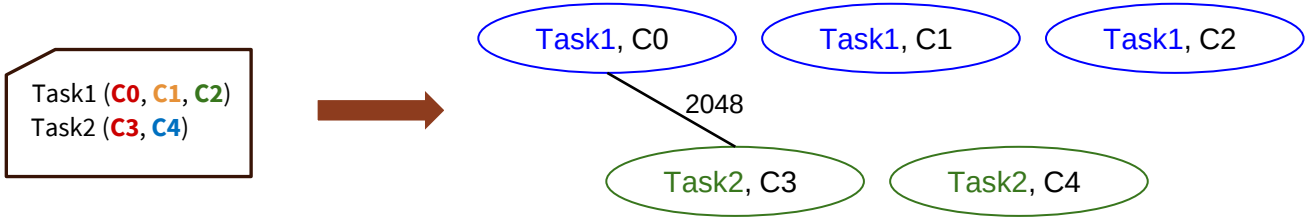
Moves may cause cascading changes to a mapping

Co-location Constraints Representation

Graph:

- Nodes are `<task, collection>` tuples
- Edges represent overlap (i.e., they reference same portions of the data)
- Edge weight is the size of common data in bytes

Edges enforce same placement for the collections arguments



Constrained Coordinate Descent (CCD)

Initial rotation has a high penalty for data movement

- All overlapping collection arguments are placed into the same memory (graph with all edges)

Relaxed as the search proceeds

- Periodically remove remaining lowest-weight edges that allows more freedom in data placement
- Gradually relaxed to balance the costs of compute and data movement

Captures important trade-off between tasks running as fast as possible and minimize data movement

Constrained Search Example

Both tasks on CPU.

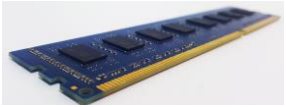
Frame-buffer Memory

GPU

Task1 (**C0**, **C1**, **C2**)
Task2 (**C3**, **C4**)

CPU

Zero-copy memory



C1 C2 C0 C3 C4

System Memory

Constrained Search Example

Move Task1 to GPU.

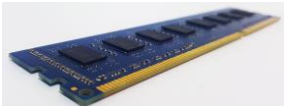
C1 C2 C0

Frame-buffer Memory

Task1 (C0, C1, C2) **GPU**

Task2 (C3, C4) **CPU**

Zero-copy memory



C3 C4

System Memory

Experiments

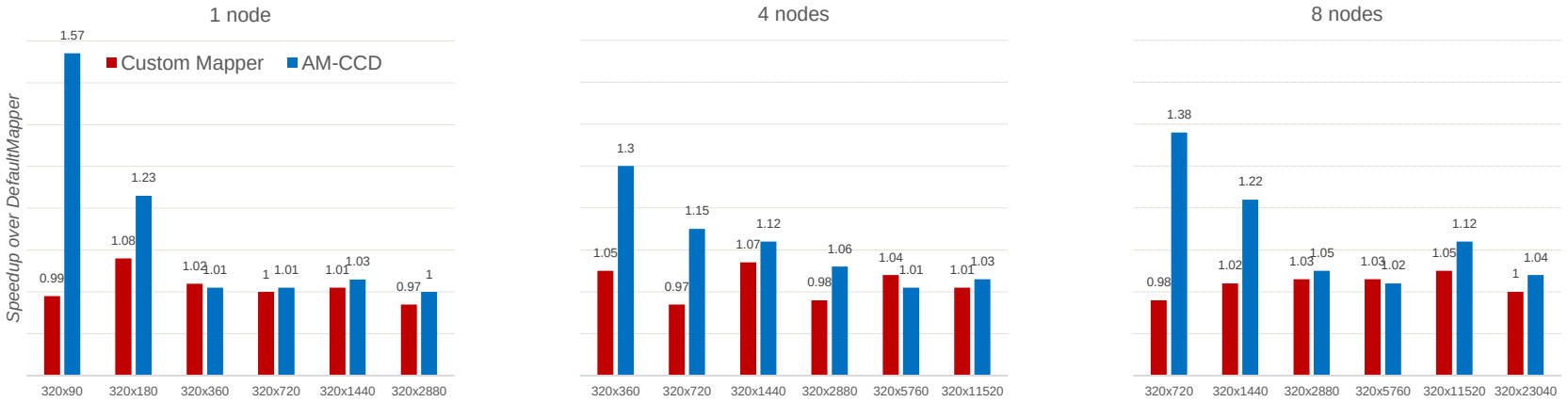
The benchmarks use Legion's task-based programming model

2 machine configurations:

- Shepard: 56-core Intel Xeon 8276 cpus and 1 Nvidia P100 per node.
- Lassen: 40-core IBM P9 cpus and 4 Nvidia V100 per node.

Pennant

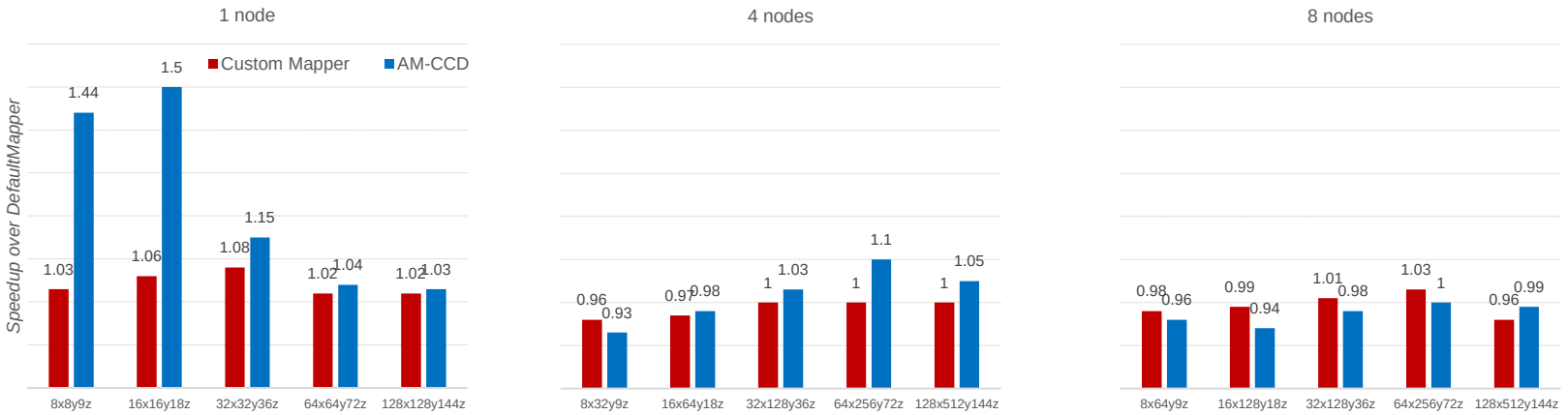
Lagrangian hydrodynamics simulation with 2^{128} possible mappings



Small inputs: tasks assigned to OpenMP and collections to System Memory.
Large inputs: tasks assigned to GPU and collections to Frame-buffer Memory.

HTR

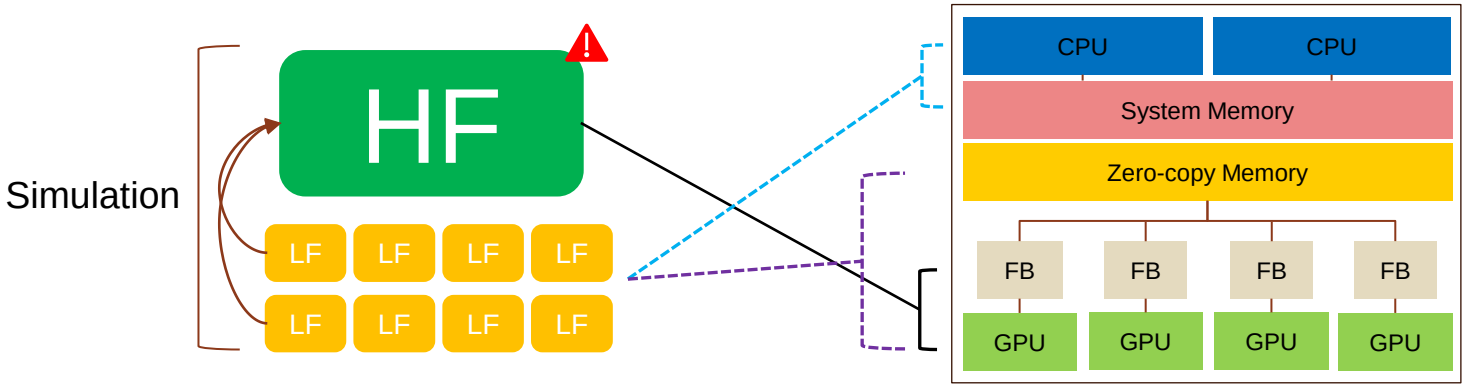
Multi-physics solver with 2^{100} possible mappings



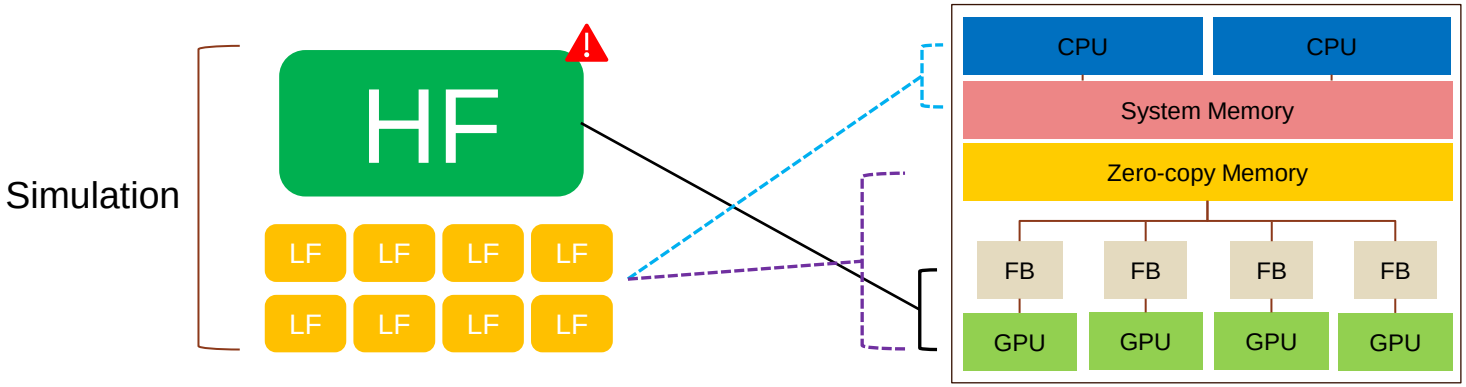
Input increases

OMP and Sys Mem → OMP and ZC → GPU and FB Mem with some in OMP+ZC → GPU and FB Mem

Multi-fidelity Ensemble CFD

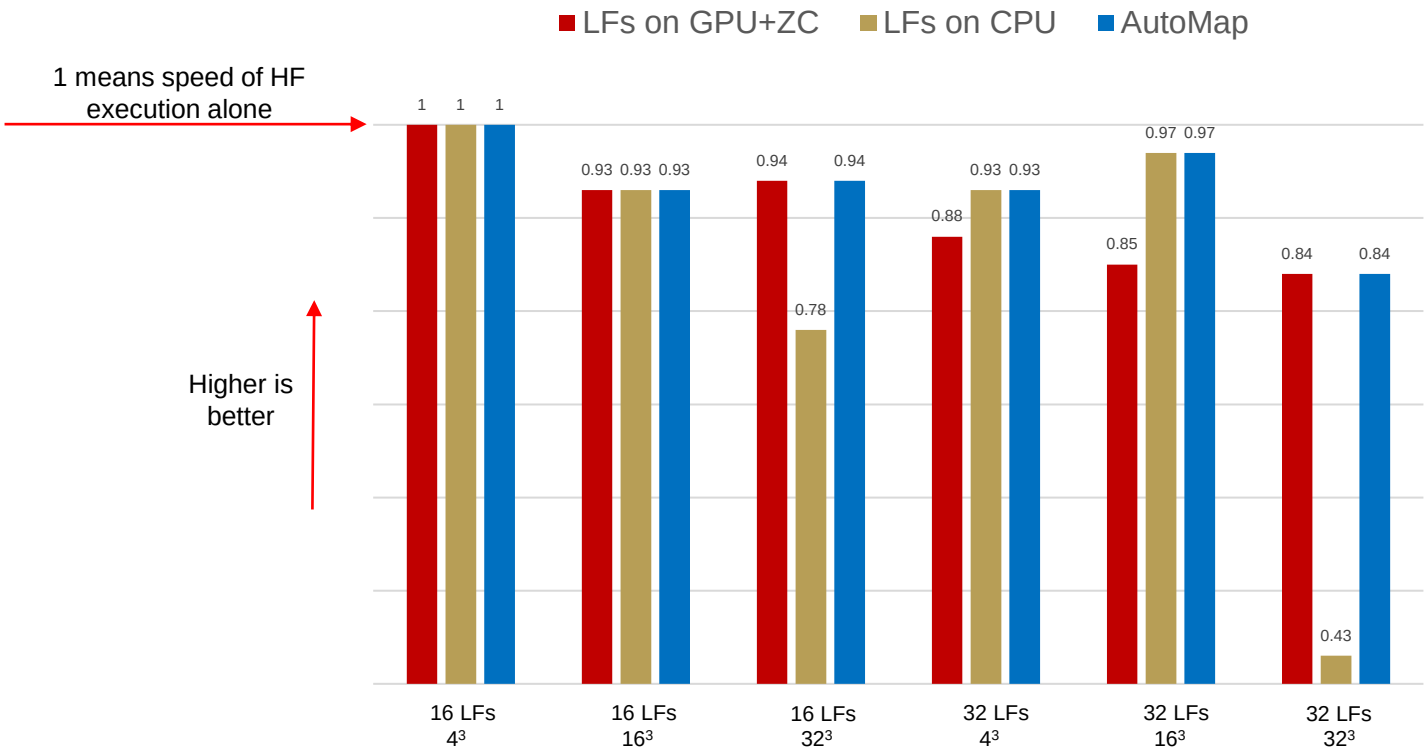


Multi-fidelity Ensemble CFD



LFs mapping?

Multi-fidelity Ensemble CFD



Conclusions

AutoMap allows finding faster mappings with no user intervention

- Consider the jointly mapping of data and tasks
- Novel search algorithm that balances the trade-off costs between computation and communication
- Up to 2.41x over hand-written, custom mapper
- Out-of-core mappings up to 50x faster than all on Zero-copy

Colocation constraints through coordinated placement instrumental for finding better mappings

`https://gitlab.com/thiagotei/automap`