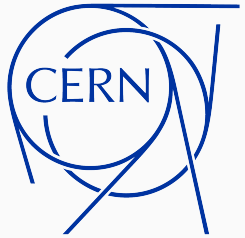


Graph-based Task Scheduling on Heterogeneous Resources

An update

Josh Ott

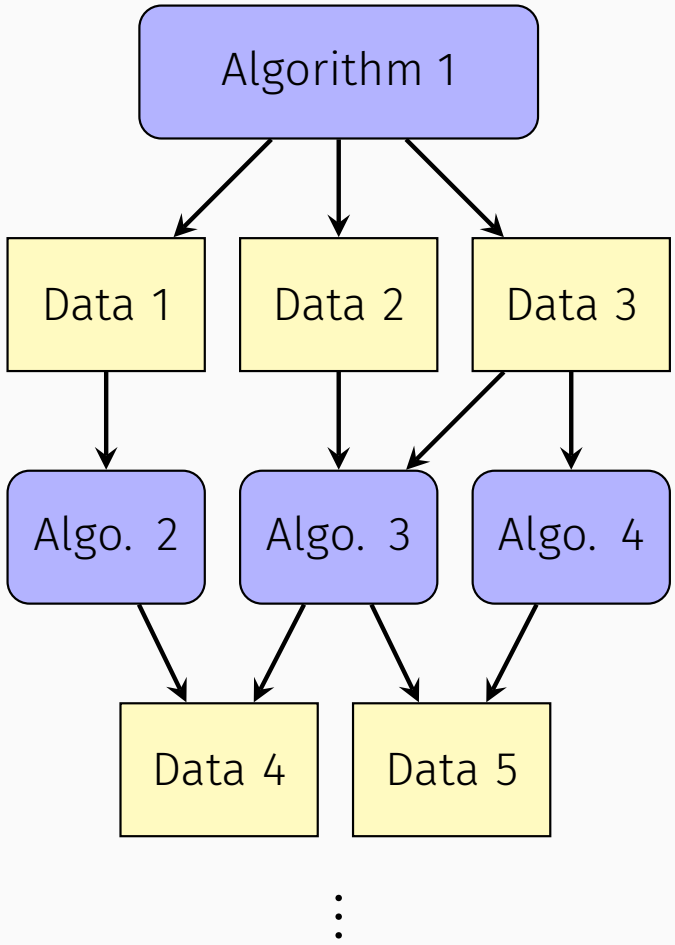
North Carolina State University

July 25, 2024

Recap

Bird's-eye view

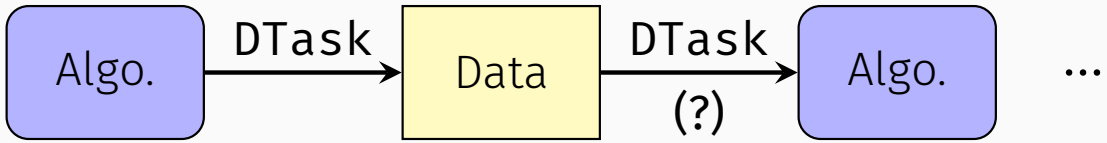
- Scheduling graphs of many tasks with dependencies on an assortment of "workers"
- Test the viability of Julia for HEP software stacks
- Let `Dagger.jl` handle the details*
- Build it with parallel, heterogeneous computing in mind



Data Dependencies

Treatment of data

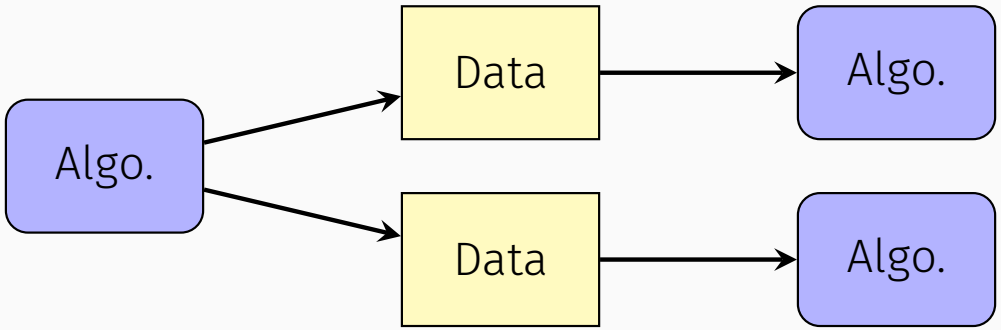
Algorithm and data nodes treated equally – not ideal!



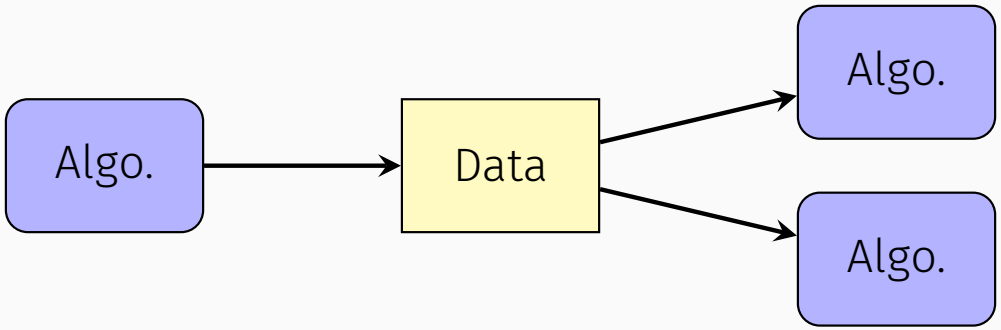
Dagger requires this to recognize dependence.
Further, the data generated did not match the graph metadata specifications.

Graph structures

A graph with topology



would actually get treated like



Dagger datadeps

Dagger provides the function `spawn_datadeps()` to handle mutable arguments and dependencies:

```
Dagger.spawn_datadeps() do
    Dagger.@spawn add!(InOut(B), In(A))
    Dagger.@spawn copyto!(Out(C), In(B))
end
```

This ensures the second task runs after the first.

All data as arguments

Rather than using the function output, treat all outputs as mutable arguments.

```
Dagger.spawn_datadeps() do
    for v in vertices
        Dagger.@spawn algorithm(
            In.(inputs)...,
            Out.(outputs)...)
    end
end
```

All data as arguments

Rather than using the function output, treat all outputs as mutable arguments.

```
Dagger.spawn_datadeps() do
    for v in vertices
        Dagger.@spawn algorithm(
            In.(inputs)...,
            Out.(outputs)...)
    end
end
```

Now we just need to produce something meaningful!

`DataObjects` with metadata

To utilize the graph metadata, we package everything in structs

```
mutable struct DataObject
    data
    size::UInt
end

function algorithm(inputs..., outputs...)
    for output in outputs
        output.data = zeros(Int8, output.size)
    end
end
```

Scheduling with discretion

Rather than scheduling data nodes, we populate them with `DataObject`s up front.

```
for v in data_vertices
    size = get_prop(graph, v, :size)
    data = DataObject(nothing, size)
    set_prop!(graph, v, :res_data, data)
end

Dagger.spawn_datadeps() do
    for v in alg_vertices
        ...
```

Problems solved!

With those additions, we've taken care of:

- ✓ transfering only the data we need,
- ✓ generating realistic data, and
- ✓ only scheduling algorithms.

Problems solved!

With those additions, we've taken care of:

- ✓ transfering only the data we need,
- ✓ generating realistic data, and
- ✓ only scheduling algorithms.

Unless...

All of it was unusable.

Datadeps revisited
What they don't tell you

Beyond the docs and in the source code, one finds

```
function spawn_datadeps(f::Base.Callable)
```

> *"At the end of executing* `f`, `spawn_datadeps` *will* ***wait for all launched tasks to complete****, rethrowing the first error, if any. The result of* `f` *will be returned from* `spawn_datadeps`*."*

Datadeps revisited
What they don't tell you

Beyond the docs and in the source code, one finds

```
function spawn_datadeps(f::Base.Callable)
```

> *"At the end of executing* `f`*,* `spawn_datadeps` *will* ***wait for all launched tasks to complete****, rethrowing the first error, if any. The result of* `f` *will be returned from* `spawn_datadeps`*."*

Asynchronicity is crucial, so this will not do.

Lessons

- Few alternatives to this approach
- Until a Dagger update changes this behavior, data dependencies are on the backburner.

Lessons

- Few alternatives to this approach
- Until a Dagger update changes this behavior, data dependencies are on the backburner.

To be continued!

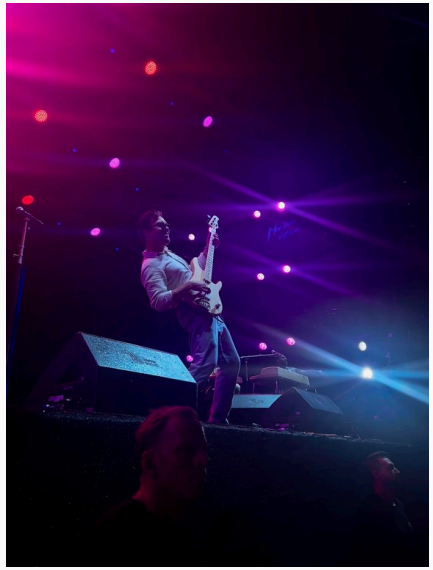
Travel!










Thanks :)