Heterogeneous frameworks current status

Mateusz Fila, Benedikt Hegner (CERN EP-SFT)
Oleksandr Shchur (Ukrainian Catholic University)
Josh Ott (North Carolina State University)

EP R&D Software Working Group Meeting
20.11.2024

Task: heterogeneous frameworks

People:

CERN EP-SFT:

- Mateusz Fila - fellow
- Benedikt Hegner - staff

Students:

- Oleksandr Shchur - Ukrainian Remote Student
- Josh Ott - Summer Student Programme 2024

Connections with other CERN activities

- Participating in discussions of NextGenTrigger Task 1.7: *Common software developments for heterogeneous architectures.*
- Following the developments in scheduling in Gaudi and Athena

Event-processing application frameworks

Many HEP domain specific applications follow similar concept:

For each event:

- Load event data
- Checkout non-event data, meta-data
- Apply transformations, filters
- Write output

Frameworks (Gaudi, CMSSW, …) support creating such application by providing:

- **Execution engine**
- Configuration layer
- Event data, non-event data, meta-data management
- Shared resources, services
- …

Heterogeneous frameworks

A few decades of experimental framework evolution:

- Single-process, single-thread event loop
- Parallel event processing with multiple processes
- Parallel event processing with multiple threads
- Gradual introduction of offload to accelerators and multi-node super-frameworks

Our R&D project:

- Explore new libraries and systems for heterogeneous computing
- Prototype new heterogeneous schedulers starting from a greenfield
- Use realistic workflows extracted from the current frameworks used by the LHC experiments

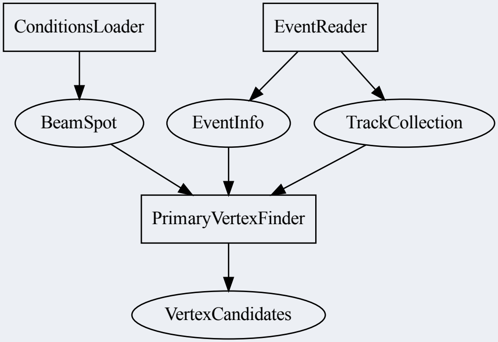
Workflow description

Data-flow graph

Directed acyclic graph (DAG) describing data dependencies between data transformations.

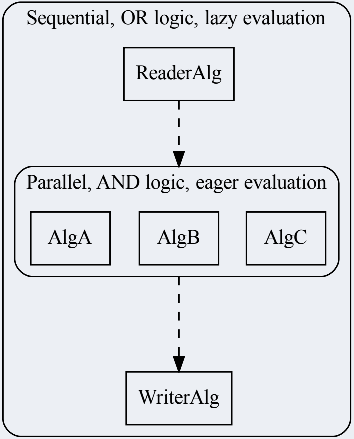
A data-flow graph consists of:

- Algorithms - vertices describing the transformations (rectangular shape)
- Data objects - vertices describing the data (oval shape)
- Directed edges representing dependency relation



Control-flow graph

DAG or subgraphs describing non-data dependencies and conditional scheduling of the algorithms.



Workflows extracted

- ATLAS standard reconstruction test job (q449)
  - $\sim$800 algorithms
  - $\sim$3700 data-objects
- FCC ALLEGRO full simulation
- Artificial workflows featuring specific scenarios (sequential and parallel chains, complex dependencies, ...)

Extracted information:

- data-flow graph
- control-flow graph
- algorithms' timings

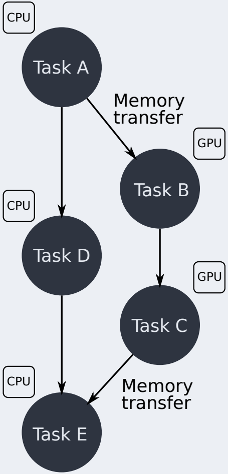
Data-object memory footprints measured for selected workflows.

Measuring data-object memory footprints

Data-object memory footprints

Memory footprints information is used as a baseline when building mockup workflows as they affect the data-transfers in heterogeneous setups

- Most current compute accelerators utilize relatively slow buses
- Communication between nodes is usually even slower
- Moving data takes significant time, especially introducing latency



Data-object memory footprints – challenges

It's difficult to define a size of relevant data as often the data-model components are connected

Memory footprint can be affected by:

- type size
- memory layout and alignment
- ownership
- indirections

Approximate values are enough for the use case

```
struct MCHit {
  uint                   cellID;
  Vector3                position;
  set<HitContribution> contribs;
};

struct HitContribution {
  uint      pdg;
  float     deposit;
  Particle* particle;
};
```

Extracting memory footprints

No silver bullet solution for all the data models:

- Individual object info not accessible from heap profilers like Massif
- Specialized tools such as object introspection limited to working with static libraries
- Can be inferred from container size for POD-based data models such as key4hep/EDM4hep

- Tracking malloc allocations - overheads, assumptions about ownership and indirections

class	size [B]
AnyDataWrapper<int>	8
AnyDataWrapper<double>	8
AnyDataWrapper<Vector3F>	24
AnyDataWrapper<Vector4F>	32

We managed to extract reasonably accurate measures for some of the workflows

Scheduling with new task-parallel libraries

Taskflow

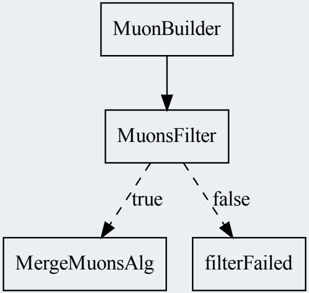
taskflow/taskflow

"A General-purpose Task-parallel Programming System — write parallel programs with high performance and simultaneous high productivity"

- Advertised as a modern replacement for oneTBB
- Actively developed and maintained, stable release 3.8
- Modern C++ API based on tasks
- Developed at the University of Wisconsin–Madison

Selected taskflow features

- Single node scheduling
- No built-in data-handling
- Built-in support for parallel-pipeline pattern
- Built-in support for **GPU-offloading** and **conditional tasks**, features of high importance for event-processing frameworks but with limited support in oneTBB

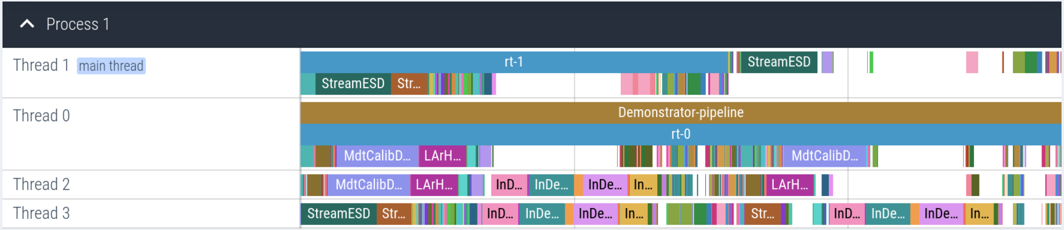


Building demonstrator with taskflow

Selected highlights from developing the demonstrator project for scheduling with taskflow m-fila/taskflow-fwk :

- Excellent documentation, very clean API
- Require some effort to reproduce Gaudi decision-hub logic
- Still need to manually take care of CUDA streams and devices when using taskflow cudaFlow API
- Currently no built-in support for coroutine-like concurrency with resumable tasks

Scheduler demonstrator using taskflow



- Single process, no-GPU
- Mocking-up ATLAS reconstruction workflow with CPUCrunchers
- 4 worker threads
- 3 events total
- 2 concurrent event slots

```
taskflow_demo --threads 4 --slots 2 --event-count 3
--logs-trace trace.json --dfg data/ATLAS/q449/df.graphml
```

Current status

- Implemented data-flow graph scheduling with taskflow
- Working on expressing custom control-flow logic implemented in Gaudi with taskflow conditional tasks
- Investigating efficient offloading with taskflow cudaFlow API
- No major issues encountered so far

Framework demonstrator in Julia

Demonstrator project

FrameworkDemo.jl key4hep/key4hep-julia-fwk :

- Explore possibilities of creating an event-processing application framework in Julia
- Not “just” port existing framework to Julia
- First try high level approach using available task schedulers, parallel execution frameworks in Julia
- Initial goal: demonstrate scheduling realistic HEP workflow in Julia using Dagger.jl package

Dagger.jl

JuliaParallel/Dagger.jl

“Dagger.jl is a framework for parallel computing across all kinds of resources, like ***CPU****s and* ***GPU****s, and across* ***multiple threads*** *and* ***multiple servers****.”*

- Under active development, current version 0.18.13, unstable public API
- Active and responsive community, weekly user meetings, active channel at Julia’s slack
- Developed at MIT

Selected Dagger features

- Single thread used to run the core scheduler,
  tasks executed in workers
- Hierarchical worker pool aware of multiple threads, processes and devices
- Configurable worker pool, works out of the box with Julia arguments `--threads`, `--procs`, `--machine-file`
- Supports nested parallelism
- Built-in distributed data-handling

Building demonstrator with Dagger

Tried two APIs for expressing algorithm dependencies:

Task dependencies:

```
task = Dagger.@spawn Producer()
Dagger.@spawn Consumer(task)
```

Overheads and boilerplate to express algorithms producing multiple data-objects

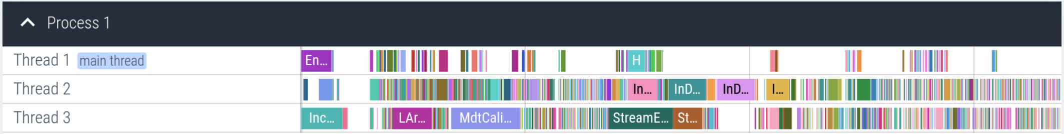
Data dependencies:

```
object = DataObject()
Dagger.spawn_datadeps() do
    Dagger.@spawn Producer(
                      Out(object))
    Dagger.@spawn Consumer(
                       In(object))
end
```

Encountered scaling issues with scheduling using multiple processes

Mockup application demonstrator

Trace from ATLAS reconstruction data-flow mockup:



Room for improvements: currently $\sim$ 3 times slower than the taskflow demonstrator, noticeable overheads for short (<10 ms) tasks

```
julia -t 3 --project bin/schedule.jl
--logs-trace trace.json data/ATLAS/q449/df.graphml
```

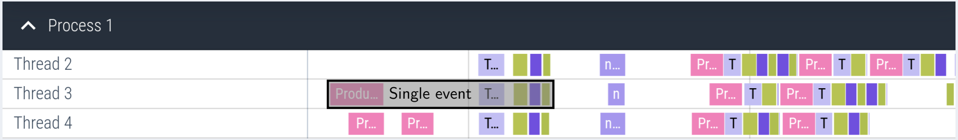
Event pipelining

Parallel execution of multiple events resembles parallel-pipeline pattern

- Not yet natively supported by Dagger
- Currently demonstrator is spawning an independent task-graph for each event

- Worth investigating upcoming Dagger “streaming tasks” aimed at repeatable tasks

Event pipelining example

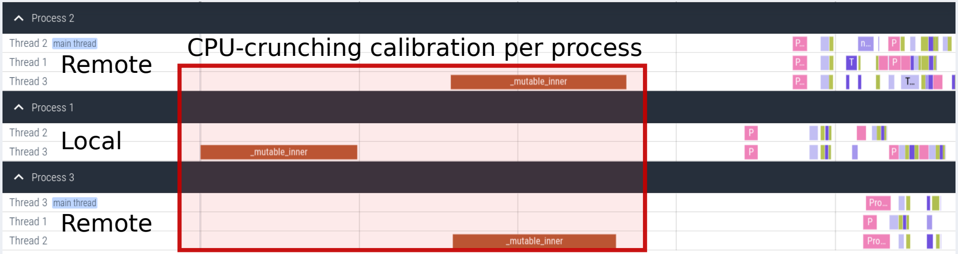


- Single process
- 1 scheduler thread
- 3 worker threads
- Sequential chain of algorithms
- 10 events total
- 3 concurrent events

We still need to understand and minimize the overheads

```
julia --project -t 4 bin/schedule.jl --event-count 10
--max-concurrent 5 data/demo/sequential/df.graphml
```

Distributed scheduling example



- 1 local process
- 2 remote processes over SSH

- Sequential chain of algorithms
- 16 events total, 8 concurrent events

Proof of principle with many aspects to improve: minimizing data-migrations, efficient communication protocols, compatibility with cluster managers...

```
julia --project -L setup.jl --machine-file=machines.txt -t 3
bin/schedule.jl --event-count 16 --max-concurrent 8
data/demo/sequential/df.graphml
```

Interacting with Dagger community

Dagger.jl is very ambitious and
not everything is working without problems...

...but we had positive experience working
on our issues with the Dagger developers:

- Opened issues - answered, fixed
- Requested new features - implemented task names
- Opened PRs - timely reviewed
- Problems, questions - answered on slack or over zoom

Summary and outlook

Summary

Heterogeneous frameworks R&D:

- Extracted information about workloads used by the experiments
- Ongoing development of single-node demonstrator using taskflow
- Investigating scheduling in new programming languages
  - Demonstrator for framework in Julia using Dagger.jl
- No critical limitations identified so far,
  investigation with both demonstrators will continue

Outlook

Phase I:

- Continue investigating demonstrators with taskflow and Dagger
- Evaluate schedulers' throughput and scaling

Phase II:

- Shift focus from single-node schedulers to distributed schedulers
- Study energy-efficienct scheduling with heterogenous nodes
- Implementation strategies for next generation frameworks

Taskflow-demonstrator m-fila/taskflow-fwk
FrameworkDemo.jl key4hep/key4hep-julia-fwk

- Mateusz Fila, *CERN EP-SFT*
  mateusz.jakub.fila@cern.ch
- Benedikt Hegner, *CERN EP-SFT*
- Oleksandr Shchur, *Ukrainian Catholic University, CERN Ukrainian Remote Student Program*
- Josh Ott, *North Carolina State University, CERN Summer Student Programme*

The work has been supported by the CERN Strategic Programme on Technologies for Future Experiments. https://ep-rnd.web.cern.ch/