Karlsruhe Institute of Technology

CERN

Gaudi Hive Project
Status Report on Recent Developments

Daniel Funke on behalf of the Concurrent Gaudi Team | Annual Concurrency Forum Meeting

CERN PH-SFT / KIT ITI

2014-04-01

The Gaudi Hive Project

Goal: enable event- and algorithm-level parallelism in the Gaudi framework

Milestones:

Nov. 2012:
- parallel demonstrator using simulated workloads [IEE NSS 1]

Oct. 2013:
- parallel execution of LHCb VELO reconstruction [CHEP 2,3]

today:
- evolved workarounds to production quality solutions
  - declaration of data dependencies
  - computation of integrated data and control flow graph
- added features essential for parallel scheduling
  - declaration of used tools
  - automatic propagation of tool's data dependencies to algorithms
  - instrumented existing framework functionalities to extract scheduling information automatically
- all while keeping Gaudi's easy-to-use interfaces, extensibility and plug-and-playability

Thanks to OpenLab (Pawel Szostek) and Techlab (Michal Husejko) for providing test beds and support for the project

Gaudi Reminder

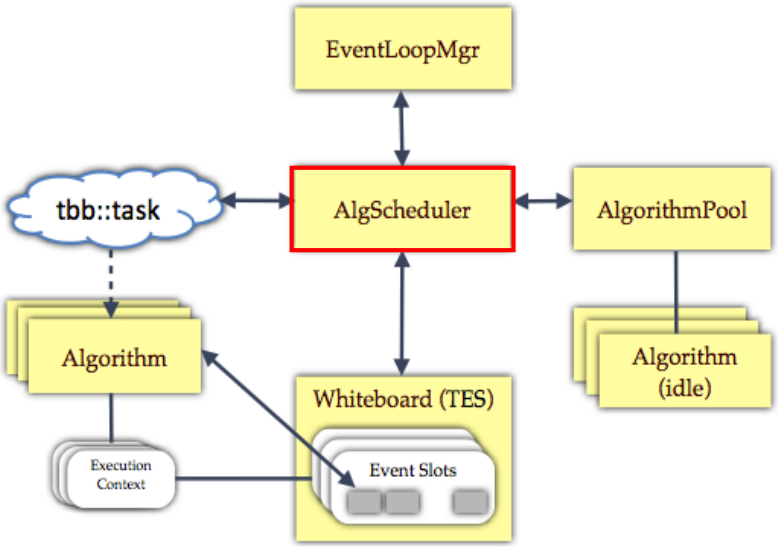
Functionality can be encapsulated in three main components:

Algorithm:
- main processing block
- consumes and produces data objects from/to data store
- steers further processing depending on data

Tool:
- computation that can be re-used by several algorithms
- may consume and produce data objects
- is owned by an algorithm (private tool) or by the tool service (public tool)

Service:
- provide fundamental framework functionality to all algorithms and tools
- is managed by the context of the framework

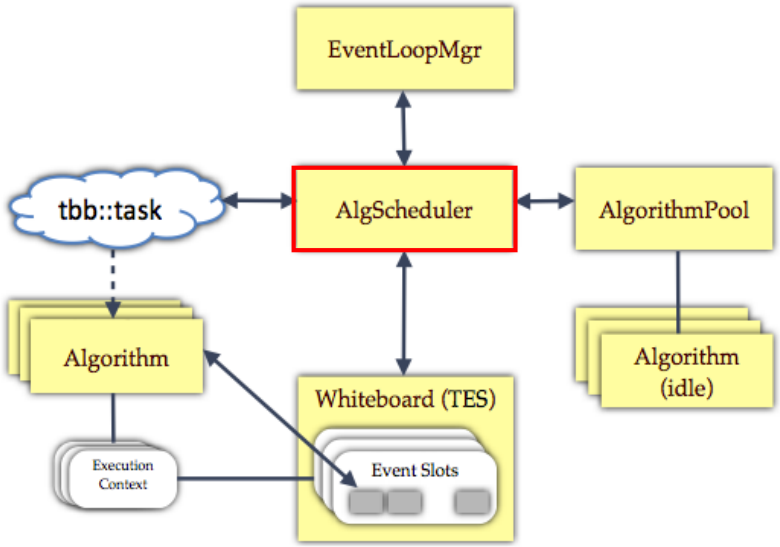
Gaudi Hive Reminder



[2]

- events processed in loop and handed over to scheduler
- scheduler acquires algorithm instances from pool and submits them to Intel TBB runtime
- each concurrently processed event has a dedicated slot in the whiteboard (multi-slot event store) to retrieve/store data items

Gaudi Hive Reminder



[2]

- events processed in loop and handed over to scheduler
- scheduler acquires algorithm instances from pool and submits them to Intel TBB runtime
- each concurrently processed event has a dedicated slot in the whiteboard (multi-slot event store) to retrieve/store data items

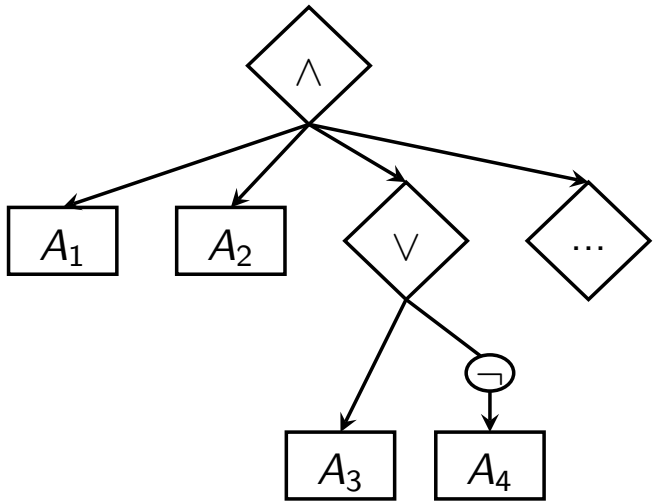
Scheduler requires knowledge of algorithms' relation to one another:

- algorithm A is prerequisite to B
- if A ran successfully, B is superfluous
- ...

Control Flow

Algorithms can be grouped in logical units (sequences):

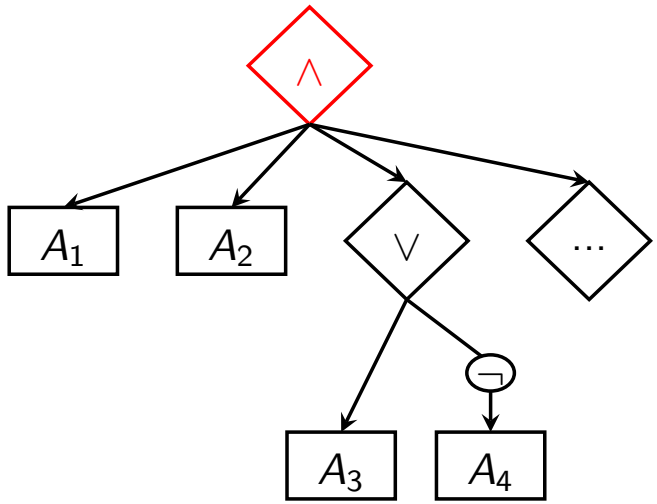
- and/or behavior of sequence with/without early return
- sequences can be part of other sequences
- an algorithm can be part of several sequences



Control Flow

Algorithms can be grouped in logical units (sequences):

- and/or behavior of sequence with/without early return
- sequences can be part of other sequences
- an algorithm can be part of several sequences



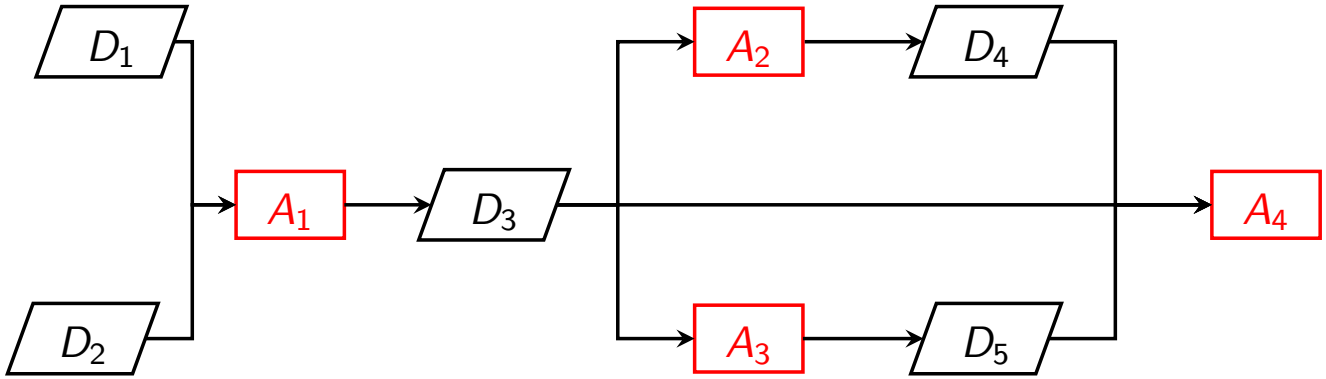
Example

Assuming an early return AND-sequence,
if A_1 produces false, $A_2 \ldots A_4$ not required to executed

Data Flow

Algorithms require and produce data objects

- establishes dependencies between algorithms
- visualized as data flow graph



See talk by Illya Shapoval on
„GaudiHive: Evolution of Execution Flow Management“

NEW

Declaring Data Dependencies

Use data handles to access data store from algorithms and tools:

Code

```
class MyAlgorithm : public GaudiAlgorithm{

private:
  DataObjectHandle<LHCb::Tracks> m_tracks;
  DataObjectHandle<LHCb::Tracks> m_filteredTracks;

public:
    MyAlgorithm( ... ) : GaudiAlgorithm( ... ) {
      declareInput("Tracks", m_tracks,
                   LHCb::TrackLocation::Default);
      declareOutput("FilteredTracks", m_filteredTracks,
                    "Analysis/FilteredTracks");
    }

    void execute() {
      LHCb::Tracks * tracks = m_tracks.get();
    }
};
```

NEW

Declaring Data Dependencies

Karlsruhe Institute of Technology

Data handles provide:

- declaration syntax similar to declareProperty (...)
- transparent use of alternative locations for a data object
- configurability from Python

Code

```
myAlg = MyAlgorithm('AnalysisFilter')
myAlg.Inputs.Tracks.Path = 'Skim/Tracks' # use pre-filtered tracks
```

- **automatic** deduction of data dependencies between algorithms

NEW

Declaring Tools

Karlsruhe Institute of Technology

Declare tools used by algorithm at configuration time:

Code

```
class MyAlgorithm : public GaudiAlgorithm{

private:
  ToolHandle<ITrackExtrapolator> m_extrapolator;
  ToolHandle<IMaterialLocator>   m_materialLocator;

public:
    MyAlgorithm( ... ) : GaudiAlgorithm( ... ) {
      declarePrivateTool(m_extrapolator, "TrackLinearExtrapolator");
      // optionally make it a property
      declareProperty("TrackExtrapolator", m_extrapolator);

      declarePublicTool(m_materialLocator, "DetailedMaterialLocator");
    }
};
```

NEW

Declaring Tools

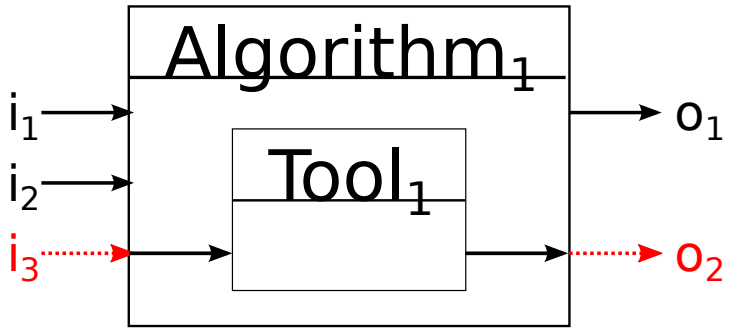
Tool handles provide:

- declaration syntax similar to declareProperty (...)
- optional configurability of private tools from Python

Code

```
myAlg = MyAlgorithm('AnalysisFilter')
myAlg.TrackExtrapolator.Iterations = 1 # rough estimate
```

- **automatic** propagation of tools in- and output to algorithm



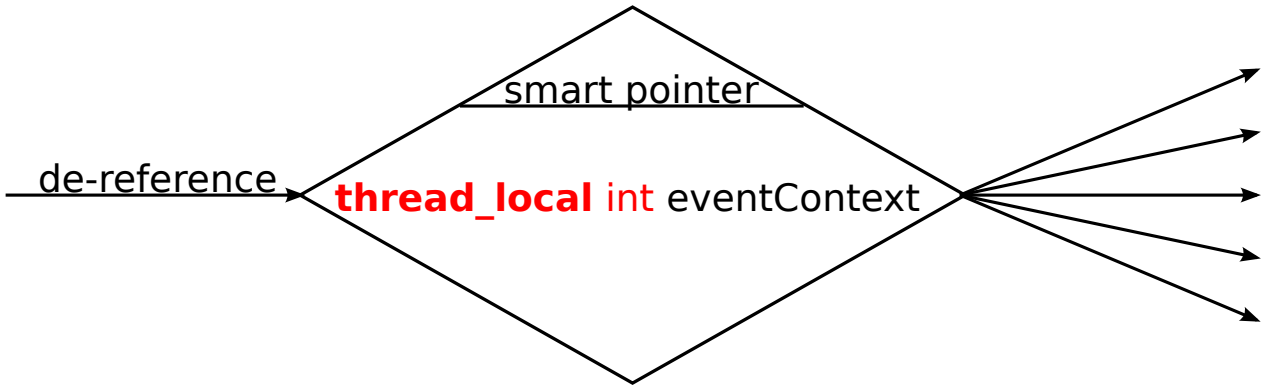
NEW

Context-aware Data Access

With concurrently processed events, event-specific data must

a) be stored in the data store
   - thread-safe and context-aware
   - event-context transparently set by framework through thread local index

b) use the context-aware smart pointer from Hive project
   - smart pointer de-references to object associated with processed event
   - thread local index set by framework



Starting Fresh

Future experiments can concurrently process data using Gaudi Hive
$\Rightarrow$ all required components are available [4,5]

Design principles to follow:

- algorithms access data store only via data handles
- tools are declared at configuration time
- data store is used for algorithms' intermediate results
- services are re-entrant or context-aware
- const-correctness is enforced

Migrating an Existing Experiment

Existing experiments have large code base
$\Rightarrow$ minimally intrusive solutions required to reduce migration effort

Some challenges to be addressed:

- missing declarations of input/output of algorithms and tools
- wide use of caches within algorithms and tools
- public tools are abused for back-channel communication
- ...

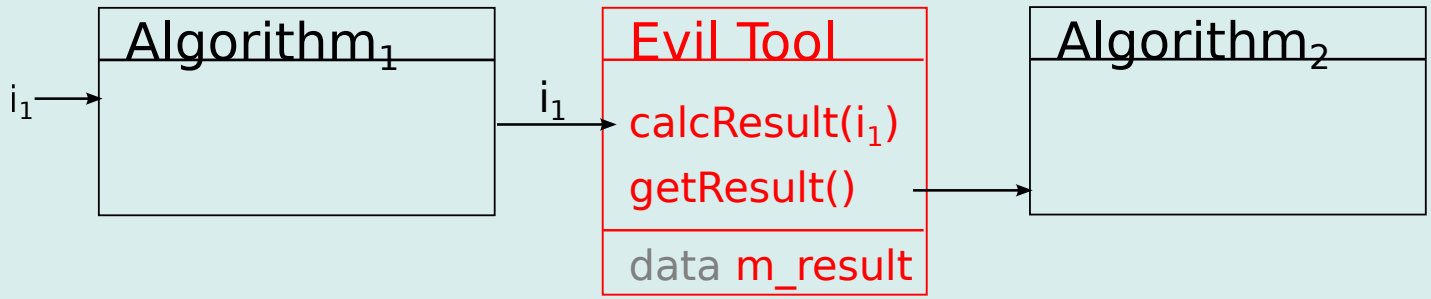
Migrating an Existing Experiment

Existing experiments have large code base
⇒ minimally intrusive solutions required to reduce migration effort

Some challenges to be addressed:

- missing declarations of input/output of algorithms and tools
- wide use of caches within algorithms and tools
- public tools are abused for back-channel communication
- ...

Example already a problem now



implicit dependency between Algorithm$_1$ and Algorithm$_2$
⇒ can **not** be automatically deduced by framework

NEW

Managing Migration Effort

Parallel processing does not come for free
but adapting features from Gaudi Hive can be done incrementally:

- data handles can be introduced algorithm by algorithm
  - sequential workflow is unaltered
  - added benefit of **static configuration checking** with data handles in place
- existing functionalities of the framework were instrumented to ease migration
  - tool retrieval via tool<T>(...) method of GaudiCommon
    $\Rightarrow$ properly registers tool usage with parent algorithm/tool for **automatic** dependency propagation
  - transparent use of context-aware smart pointer
- gradually revise use of caches, public tools, ...

Summary and Outlook

Future experiments can start developing parallel data processing today [5]

For existing experiments, gradually merge Hive project into Gaudi package

- extend parallelized LHCb workflow (currently VELO reconstruction) to trigger/full reconstruction/simulation

  $\Rightarrow$ declaring data dependencies yields additional benefit of static **configuration checking**
- develop further best practices for migration

Technical developments:

- use data handles for asynchronous write to data store
- investigate new graph-based scheduling strategies
- explore use of accelerators

References

1. P. Mato, Evolving LHC Data Processing Frameworks for Efficient Exploitation of New CPU Architectures, IEEE NSS 2012, November
2. D.Piparo, Preparing HEP Software for Concurrency - Lessons learned from the Concurrent Gaudi Project, CHEP 2013, October
3. B. Hegner, Introducing Concurrency in the Gaudi Data Processing Framework, CHEP 2013, October
4. Concurrency for HEP Twiki:
   `https://twiki.cern.ch/twiki/bin/view/C4Hep/WebHome`
5. Gaudi Hive git repository:
   git clone −b dev/hive http://cern.ch/gaudi/GaudiMC.git

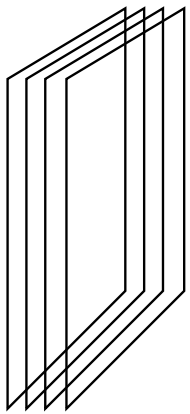
Backup

Scheduling

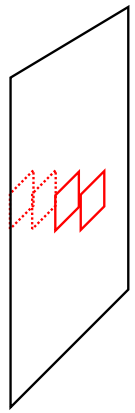
Different scheduling strategies transparently available:

- **Parallel Sequential** mimic multi-process approach
  $\Rightarrow$ but with reduced memory footprint

multi-process



multi-thread



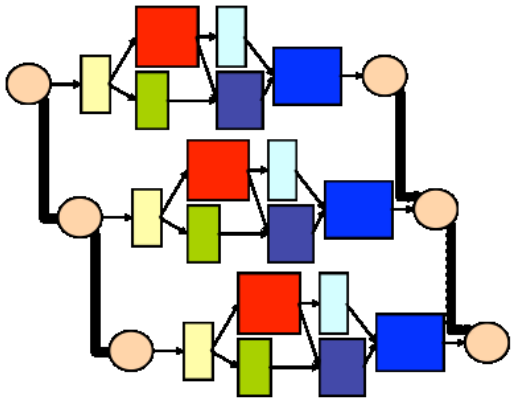
context specific state

Scheduling

Different scheduling strategies transparently available:

- **Parallel Sequential** mimic multi-process approach
  $\Rightarrow$ but with reduced memory footprint
- **Forward** schedule executable (control-flow) algorithms as soon as their input becomes available (data-flow)

Only forward scheduler exploits intra-event parallelism



[2]

Scheduling

Different scheduling strategies transparently available:

- **Parallel Sequential** mimic multi-process approach
  $\Rightarrow$ but with reduced memory footprint
- **Forward** schedule executable (control-flow) algorithms
  as soon as their input becomes available (data-flow)

Only forward scheduler exploits intra-event parallelism

Future plans:

- backward schedule only algorithms required to produce final result
- use accelerators: bunch up events to make load-off profitable