Contribution ID: **61** Type: **Oral**

Gaudi Components for Concurrency: Concurrency for Existing and Future Experiments

Tuesday 2 September 2014 16:10 (25 minutes)

HEP experiments produce enormous data sets at an ever-growing rate.
To cope with the challenge posed by these data sets,
experiments' software needs to embrace all capabilities modern CPUs offer.
With decreasing $^{\text{memory}}/_{\text{core}}$ ratio,
the one-process-per-core approach of recent years becomes less feasible.
Instead, multi-threading with fine-grained parallelism needs to be exploited
to benefit from memory sharing among threads.

Gaudi is an experiment-independent data processing framework,
used for instance by the ATLAS and LHCb experiments at CERN's Large Hadron Collider.
It has originally been designed with only sequential processing in mind.
In a recent effort, the framework has been extended to allow for multi-threaded processing.
This includes components for concurrent scheduling of several algorithms
– either processing the same or multiple events,
thread-safe data store access and resource management.

In the sequential case, the relationships between algorithms are encoded implicitly
in their pre-determined execution order.
For parallel processing, these relationships need to be expressed explicitly,
in order for the scheduler to be able to exploit maximum parallelism while respecting
dependencies between algorithms.
Therefore, means to express and automatically track these dependencies need to be provided by the framework.

The experiments using Gaudi have built a substantial code base,
thus a minimally intrusive approach and a clear migration path for the adoption of
multi-threading is required for the extended framework to succeed.

In this paper, we present components introduced to express and track
dependencies of algorithms to deduce a precedence-constrained directed acyclic graph,
which serves as basis for our algorithmically sophisticated scheduling approach for
tasks with dynamic priorities.
We introduce an incremental migration path for existing experiments towards
parallel processing and highlight the benefits of explicit dependencies even
in the sequential case, such as sanity checks and sequence optimization
by graph analysis.

Summary

Non-intrusiveness is one of the primary goals of the project,
in order to ease the effort of introducing multi-threading to the
experiments' large existing code bases.
The concurrent components therefore present an option to the user and do not interfere with
existing sequential production code.

This allows for an incremental revision of the existing implementation and adoption of the components offered.
In order to process data concurrently the user must

1. explicitly declare data in- and outputs to each algorithm,
2. declare which reusable modules – called Tools in the Gaudi context – are used by an algorithm, and
3. revise thread-unsafe code such as use of caches, back-channel communication, race conditions in the update of shared data, …

The first two points can be achieved by using the newly introduced data and tool handles, respectively.
To ease their use, the interfaces of the handles were designed according to components familiar to Gaudi developers.
The third point requires a case by case analysis of the operations performed by the algorithm.

Having explicitly stated the algorithms' in- and outputs allows the automated deduction of the data flow among them.
In Gaudi, the execution path is not only influenced by data dependencies but also by control flow constructs.
These are expressed by grouping algorithms in sequences.
Each algorithm produces a binary decision, which causes the further execution or abortion of its sequence.
By composing several sequences, a complex control flow can be modeled.
Including both data dependencies and control flow constructs in a unified graph, gives a coherent picture of the relationships between algorithms.
The information required to construct the control flow portion of the graph can be extracted from the sequence hierarchy automatically and requires no further user intervention.

The unified data- and control-flow graph is a precedence-constrained directed acyclic graph.
The scheduler identifies algorithms with fulfilled data dependencies in the graph, which are candidates for execution.
An algorithm's execution priority is determined by its connectedness and membership to the critical path; it can become zero if it is no longer required to execute by the control flow.
Candidates with non-zero priority are submitted by the scheduler for execution in the order of their priorities.

The graph structure also serves a secondary purpose:
it allows the analysis of a given configuration statically for unfulfillable data dependencies and superfluous control flow constructs.
Furthermore, the length of the critical path and the maximum number of concurrently executable algorithms can be determined to aid in resource planning.

The gradual migration strategy and the benefits of static configuration analysis present a strong argument for the adoption of the components and concepts developed in the concurrent Gaudi project.
The adoption presents an opportunity for general code improvement as well as the possibility to explore the use of accelerators and co-processors in the future.

Authors: FUNKE, Daniel (KIT - Karlsruhe Institute of Technology (DE)); SHAPOVAL, Illya (CERN, KIPT, UNIFE, INFN-FE)

Co-authors: HEGNER, Benedikt (CERN); PIPARO, Danilo (CERN); CLEMENCIC, Marco (CERN); Dr MATO VILA, Pere (CERN)

Presenter: FUNKE, Daniel (KIT - Karlsruhe Institute of Technology (DE))

Session Classification: Computing Technology for Physics Research

Track Classification: Computing Technology for Physics Research