Toward a parallel version of Marlin: MarlinMT.

4th AIDA annual meeting

Rémi Ete, F.Gaede

DESY

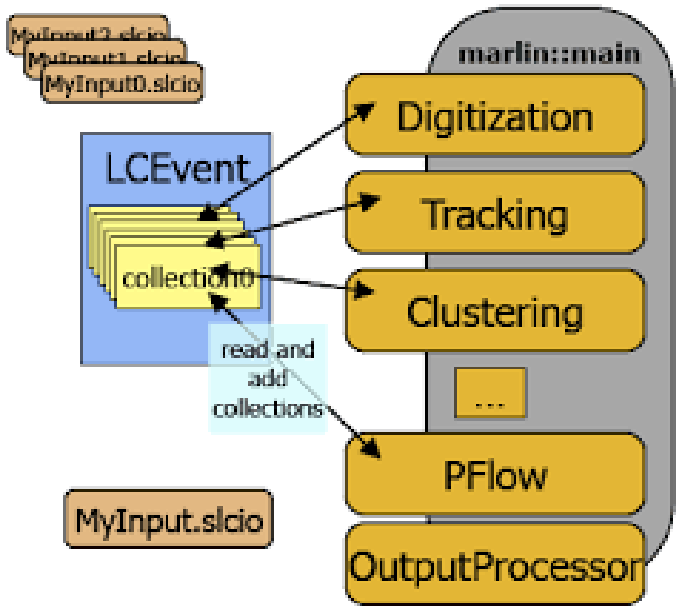
April 4, 2019

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

The Marlin framework

A Modular Analysis and Reconstruction for LINear collider

- An event processing framework for HEP
- Event data model: LCIO
- Non-ROOT framework
- General HEP application pipelining
  - Read event (one by one) from file(s)
  - Pass it through the processor chain
- Main usage is reconstruction
  - Post simulation framework
  - Pre analysis framework
- Some remarks
  - Data condition done via processor (`ConditionProcessor`)
  - Histogramming done via processor (`AIDAProcessor`)
  - **No parallelism, all serial**



The Marlin framework

Motivation for a multi-threaded version

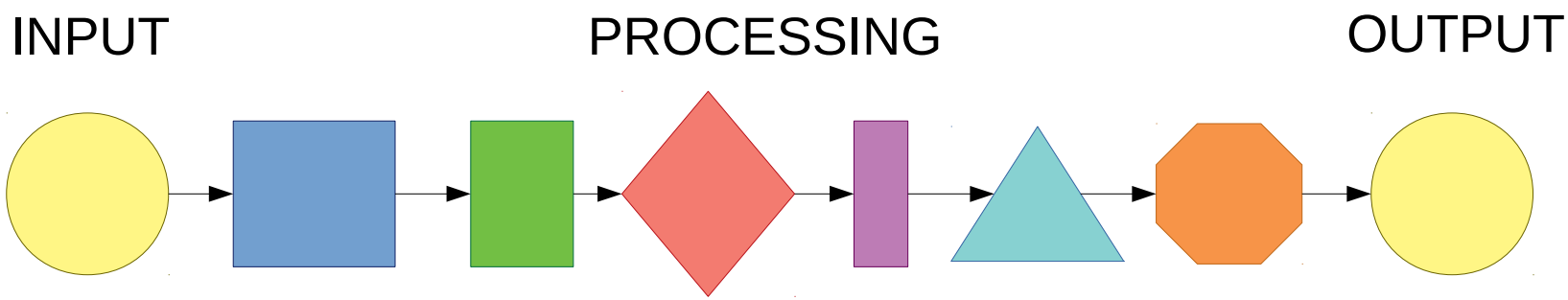
- Limitation to multi-process parallelism
  - **Duplication of process memory** (detector geometry, user data, etc ...)
    $\rightarrow$ LHC showed strong limitation there: CMS tot mem (2018) $\sim$ 8Gb
  - **Duplication of IO calls** (fread/fwrite)
    $\rightarrow$ Concurrent access to cache by different processes
    $\rightarrow$ Reduce data caching: speedup application
- Clumsy map-reduce strategy at process level
  - Splitting of hundreds of files
  - Requires understanding of file splitting
  - Re-combination can be tricky (event numbering)
  - **Introduces additional data managment**
- **Duplicated application startup time for each process**
  - Geometry loading
  - Background map loading
  - Potential database access

Event processing pipeline in HEP

From simple pipeline ...

General HEP event processing framework follow a pipeline pattern

- One event at a time
- One task at a time
- Keep only one core busy

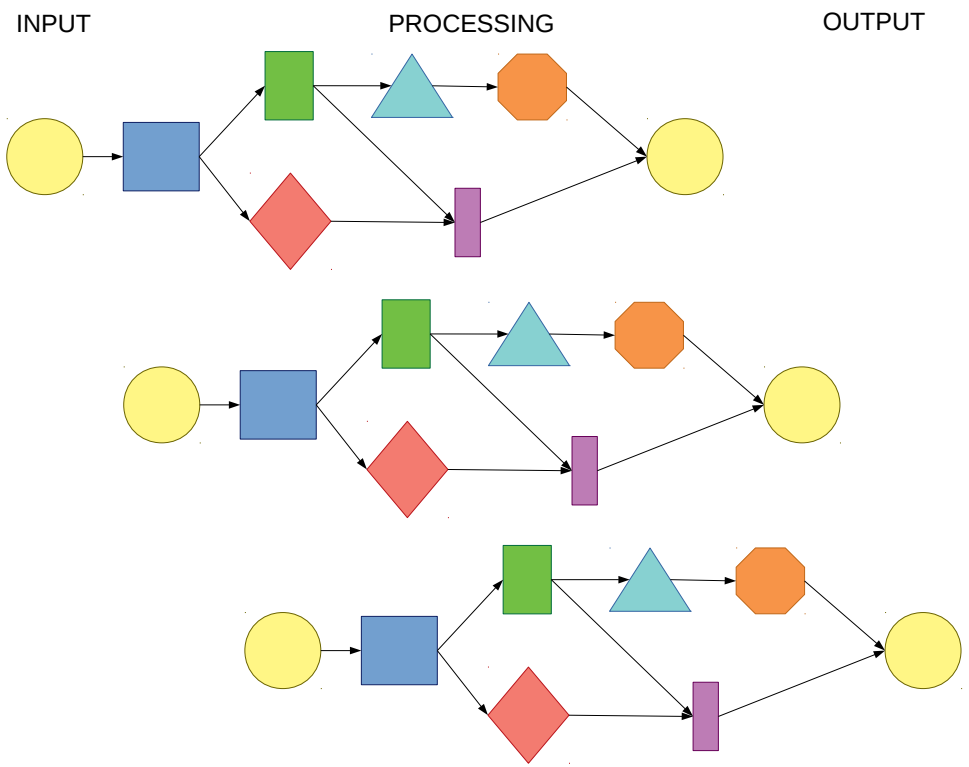


Event processing pipeline in HEP

... to superscalar sequences

Build an Directed Acylic Graph based on task data dependencies.
Parallelism at task (intra) and event (inter) level.

- Complex task scheduling
- Architecture adopted by LHC experiments (e.g Gaudi)

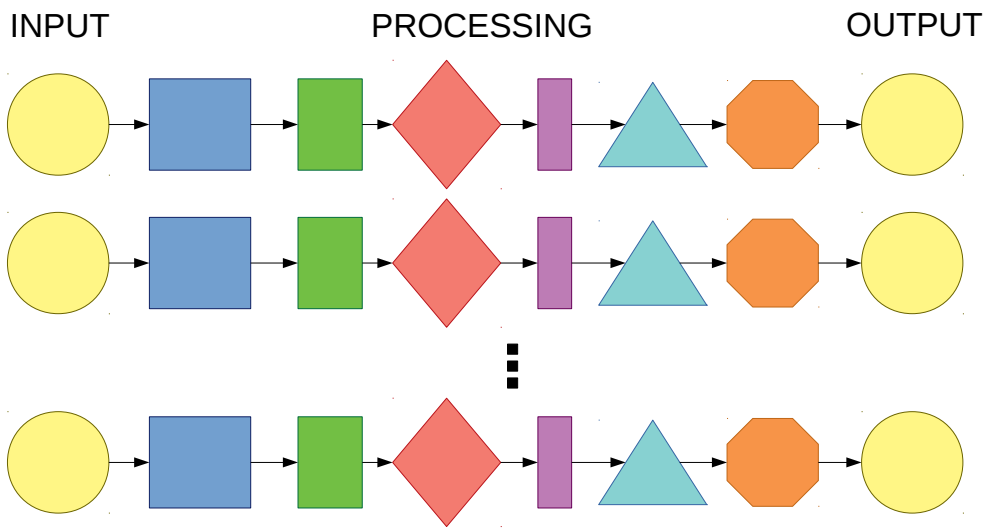


MarlinMT

Chosen solution: fully parallel pipeline

Simply duplicate the full pipeline for each thread

- N events at a time
- **Best and most natural scaling**
- Hard requirements: **Each task must to be thread-safe**



MarlinMT

First parallelization strategy

Parallelization over LCIO input files

Current application worflow:

- Create N task threads with each a processor chain
- For M LCIO files, divide them among threads
- Initialize application (`Processor::init()`)
- **For each thread:**
  - **Open LCIO files with a `LCReader`**
  - **Process events**
- Join threads
- Terminate the application (`Processor::end()`)

A running scenario:
processing 12 files in 3 threads, each with 4 files

MarlinMT

Pre-implementation stage

MarlinMT repository:
→ Github: **rete/Marlin**, branch `marlin-mt`
Will be merged in iLCSoft/Marlin, branch marlin-mt.

Investigating thread safety in dependencies:

- ✔ LCIO: event data model (IO + EDM)
  → Github: iLCSoft/lcio, branch `lcio-thread-safe`
- ✔ streamlog: main logging library
  → Github: iLCSoft/iLCUtil, branch `streamlog-thread-safe`
- ✔ Geometry: Gear (deprecated), DD4hep
  → const access, so thread-safe by definition
- ✗ RAIDA: histograming interface with ROOT implementation
- ✗ LCCD: conditions data handling, one of the most tricky part

LCIO thread-safety

SIO layer

- A lot of global variables in SIO library !
- IO layer re-factored
  - can use different SIOReader/SIOWriter instances in different threads
- SIO can be used independently from LCIO:
  $\rightarrow$ `find_package( LCIO COMPONENTS sio )`
- Could possibly now implement IO layer in PODIO with SIO ...

LCIO thread-safety

LCIO interface

- A few minor changes, **no interface breaking !**
  - LCIO object extension mechanism (introduced overhead but thread safe)
  - LCObject uid assignment is now atomic
- LCReader issue: unclear `LCEvent` ownership in MT environment
  - `LCReader::readNextEvent()` deletes the previous event
  - `LCEvent` ownership unclear for LCEventListeners
- Added new classes for MT use to solve the problem
  - `MT::LCReader`: provide callbacks with `shared_ptr<LCEvent>`
  - `EVENT::LCEvent` deleted with reference counting
- Added MT examples in LCIO repository as demonstrator

streamlog thread-safety

Issues listing and re-implementation

- Global logger issues
  - Dataraces when using `streamlog_out()` macro
  - Uses cout without lock (no datarace but printout not ordered)
    → introduced mutexes, configurable at compile/run-time
    → `streamlog_out` changed but backward compatible
- Marlin verbosity level setting
  - Change of Processor verbosity level from different threads
    → datarace !
  - Introduced local logger instance in Processor class
- streamlog extension:
  - Can log in different *sinks* (console, file, ...) with the same/different logger

```
auto logger = logstream::createLogger( "mylogger", {
  logstream::console(), logstream::simpleFile("logfile.log") });
// log in console and log file
logger->log<MESSAGE>() << "Hello world !" << std::endl ;
```

MarlinMT framework

Geometry issues/solutions

DD4hep geometry loading:

- Loaded by a processor (`InitializeDD4hep`)
- Duplicating processor chains = duplicating geometry loading...
  → Introduced GeometryManager class and GeometryPlugin
  - Unique geometry loading
  - Unique geometry access via const method

```
auto handle = geoManager->geometry<dd4hep::Detector>();
```

Gear geometry loading:

- GearMgr instance stored in static Global::GEAR (removed now)
- Gear is deprecated and will be removed (kept for the exercise...)
- Same solution as for DD4hep: geometry loading encapsulated in a `GeometryPlugin`

```
auto handle = geoManager->geometry<gear::GearMgr>();
```

MarlinMT framework

Processor manager / application management

ProcessorManager singleton class

- Accessed everywhere by almost every class in the framework
- Too much responsabilities:
  - processor plugin,
  - application configuration,
  - logging setup,
  - scheduling, ...
- Class removed and split in different classes
  - PluginManager: remaining singleton. Loads plugins from DLL
  - ProcessorChain: runs a set of processor
  - LoggerManager: configuration/access/creation of *streamlog* loggers
  - Application: configure an application logic (ST/MT)

MarlinMT framework

Current status

First compiling and running version available on Github !

- Implemented a simple number crunching test processor
- Running 9 Marlin processors on N threads with N files
  $\rightarrow$ scales perfectly !
- Very simple test setup but encouraging start for the project

Currently investigating:

- Histograming component:
  - ROOT `TH1::Fill` with locks in an AIDA wrapper ?
- Data conditions in MT environment:
  - Possible MT solutions: LCCD, DD4hep, others ?
- LCIO output writing
  - Currently: LCIOOutputProcessor (not MT suitable)
  - Output event pool + defered writing ? Split & merge strategy ?
- ILD reco chain issues in MT environment

Summary and outlook

Conclusion

- Parallel version of Marlin investigated
- First parallelization: MT over input LCIO files
- Investigated thread safety issues in dependencies:
  - ✔ LCIO: SIO layer, interface
  - ✔ streamlog: thread safe logging available
  - ✔ Geometry: geometry loading (DD4hep/Gear)
- Implemented a first working version of MarlinMT
  - ✔ Tested with simple processor

Outlook

- Missing ingredients to be integrated
  - ✗ LCIO output data management
  - ✗ Histograming component
  - ✗ Conditions data handling
- Will investigate ILD reco chain bottlenecks

Thanks !