

# Framework extensions The MarlinMT effort.

AIDA 2020 final meeting 2019

[Rémi Ete](#), F. Gaede, H. Grasland, J. Benda

DESY

April 23, 2020

**HELMHOLTZ**  
RESEARCH FOR GRAND CHALLENGES



**AIDA**<sup>2020</sup>



# The Marlin framework

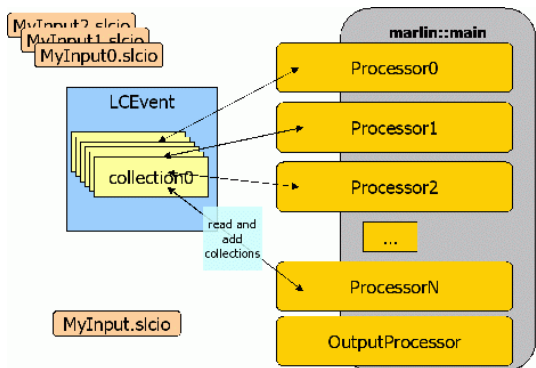
 <https://github.com/iLCSoft/Marlin>

Standard HEP event processing framework

- **LCIO** as event data model
- **LCCD** for data condition handling
- **AIDA** for histogram handling

Main components

- Processor: Main user module
- Application steered via XML files
- Plugin mechanism
- Standard event processing pipeline.
  - **All sequential**
  - **Parallelization at process level**



# The MarlinMT framework

 <https://github.com/iLCSoft/MarlinMT>

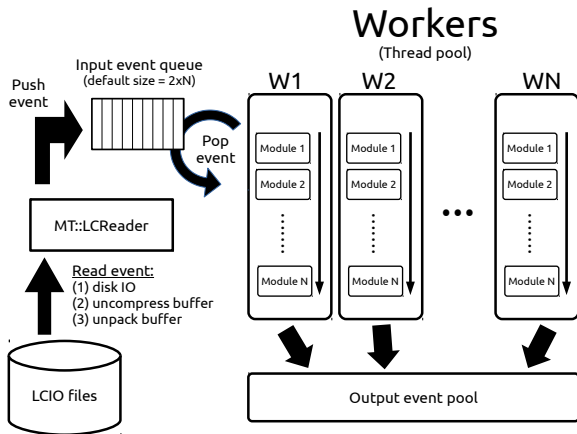
Choice of parallelism ?

- **Event parallelism only**, one thread per event
- No task parallelism
  - Introduces thread-safety additional requirements at user/framework level
  - Not mandatory for a first prototype
- Use only STL library for multi-threaded scheduler
  - Keep implementation simple
  - Keep dependencies minimal
  - May use TBB later...
- Processing pipeline replicated for each thread
  - Modules are cloned or shared
  - Depends on memory / thread-safety requirements



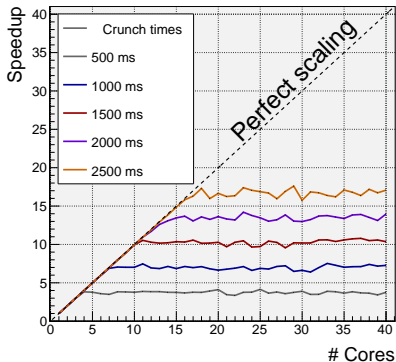
# The PEP scheduler

## Scheduler architecture



# The PEP scheduler

## First scaling performances



AMD EPYC 7451 24-Core (2 x 24 x 2 threads)

CPU crunching: crunch numbers for  $n$  ms

**Amdahl's law:**

$$S_{latency} = \frac{1}{1 - p + \frac{p}{s}}$$

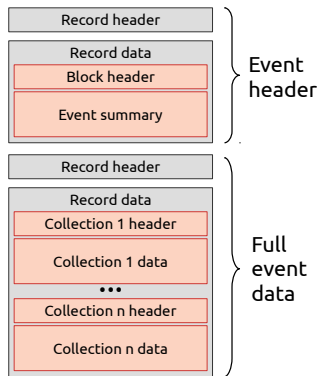
- Parallel code is CPU crunching  
→ Limited by sequential code fraction
- Main sequential code:  
→ LCIO file reading !
- **Need to reduce sequential code**



# The SIO library

<https://github.com/iLCSoft/SIO>

- SIO is the IO layer of LCIO
- Re-implemented with thread-safety in mind
- Features:
  - Schema evolution (block versioning)
  - Pointer chasing (e.g linked-list)
  - Fast random access of records
  - Endianness agnostic (always write in big endian)
- LCIO event written in two records:
  - *Event header*: event summary (simple metadata)
  - *Event record*: all event collections



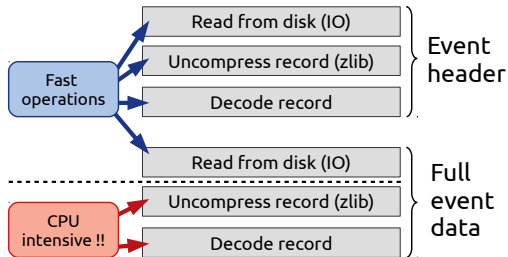
LCIO event structure



# LCIO lazy unpacking with new SIO

## LCIO lazy unpacking

- 1 Extract event record in a buffer
- 2 Move the buffer in LCEvent
- 3 Trigger full event decoding on first collection access



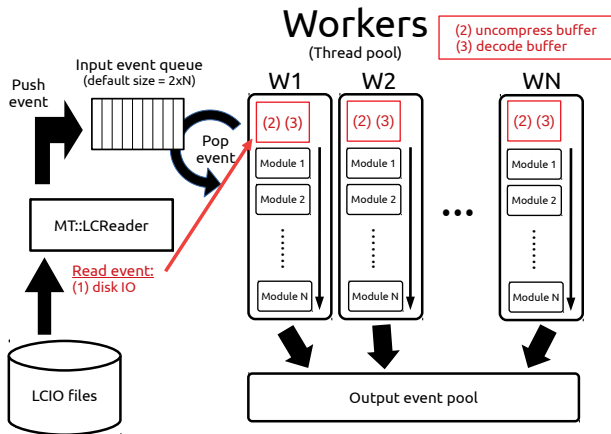
LCIO event decoding

**Trigger CPU intensive decoding only on demand!**



# MarlinMT scheduler

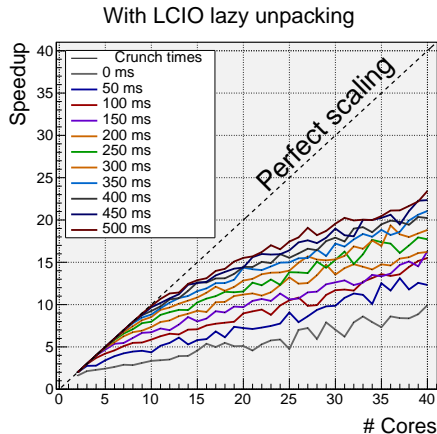
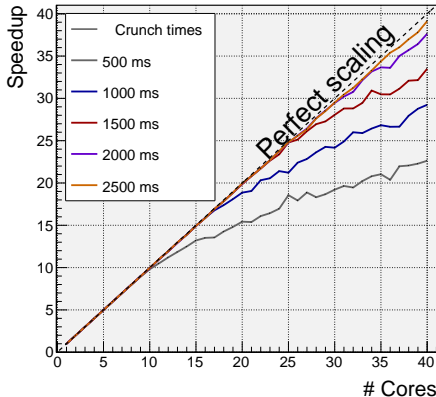
Parallel event processing - **With lazy unpacking**





# MarlinMT scheduler

Scaling performances with lazy unpacking



AMD EPYC 7451 24-Core (2 x 24 x 2 threads)



# Histogramming and multi-threading

Common dilemma of histograms in multi-threaded frameworks:

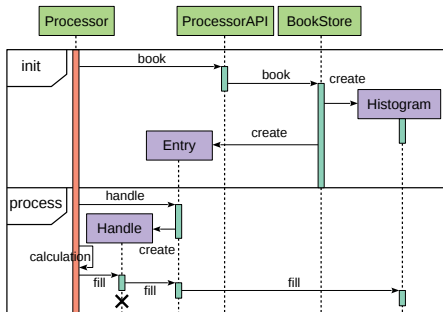
- One histogram in N threads ? Nice for memory ! Killing CPU !
- N histogram copies in N threads ? Nice for CPU ! Killing memory !

MarlinMT::Book:

- Histogramming component of MarlinMT
- Use ROOT::RHist for backend
- Additional templated layer for MT management
- Support for both shared and multiple copies of histogram (runtime)

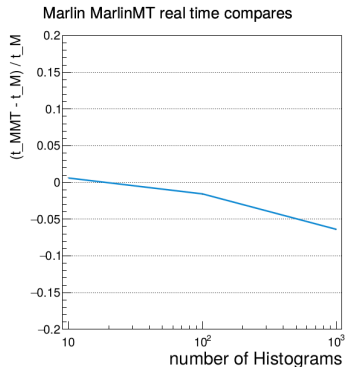
Performances:

- Overhead ?
- Impact on scaling?

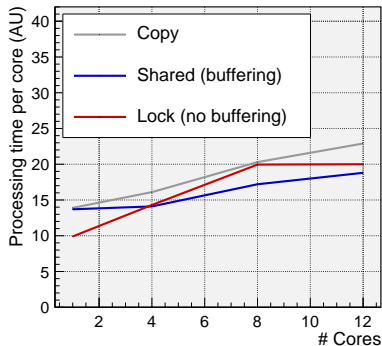


# Histogramming and multi-threading

MarlinMT::Book performances



Single threaded mode.  
Marlin VS MarlinMT



Comparable performances for copy,  
shared AND standard locking



# Conclusion and outlook

## Conclusion

- A prototype of MarlinMT has been developed:
  - Schedule parallel event processing in worker threads
  - Benchmarking doesn't show big issues → keep this implementation for now
- New SIO implementation
  - Full control on read/write steps: disk I/O, un/compression, en/de-coding
  - Allow for lazy data en/de-coding → Speeds up MarlinMT !
  - New SIO implementation used in other projects: see talk by F. Gaede on PODIO
- MarlinMT: :Book for histogram management in MT mode

## Outlook

- Re-furbish user API, write documentation
- Integrate new iLCUtil and LCIO versio in master
- Start porting exisiting reconstruction code



# Backups



# LCIO reading performance

Comparison of extract and decode times

		Extract time	Decode time	Read fraction	Event size
SIM	Single photon	2.583	33.116	12.821	1.34
	uds 200 GeV	9.399	79.45	8.453	3.05
	4f WW had	20	171	8.55	5.85
REC	Single photon	1.916	26.25	13.7	2.5
	uds 200 GeV	9.749	156.5	16.053	1
	4f WW had	20.1	331.7	16.502	55.6
DST	Single photon	0.085	0.088	1.035	0.0016
	uds 200 GeV	0.179	1.51203	8.447	0.052
	4f WW had	0.334	3.118	9.335	0.1

**Table:** LCIO file read performances (single thread). Read times in *milli-seconds*. Event size in MG. Read fraction defined as decode / extract. Results may vary on different machines.



# Benchmarking machine

- Architecture: x86\_64
- Byte Order: Little Endian
- CPU(s): 96
- Thread(s) per core: 2
- Core(s) per socket: 24
- Socket(s): 2
- NUMA node(s): 8
- CPU family: 23
- Model name: AMD EPYC 7451 24-Core Processor
- CPU MHz: 1200.000
- CPU max MHz: 2300.0000
- CPU min MHz: 1200.0000
- L1d cache: 32K
- L1i cache: 64K
- L2 cache: 512K
- L3 cache: 8192K

