Multithreaded Simulation for ATLAS: Challenges and Validation Strategy

CHEP 2019
Adelaide
5th November 2019

Marilena Bandieramonte
John Derek Chapman
Heather Gray
Miha Muskinja
Yu Him Justin Chiu
on behalf of the ATLAS collaboration
Computing complexity challenges

- In Run3, we plan to run at least 50% of simulation with fast techniques (we aim to reach ~75%), but full Geant4 simulation will be heavily used regardless.
- In Run 4, Full Simulation is expected to be the largest CPU consumers (20-25%)
  - Together with FastSim and FastReco it amounts to ~40% of all expected CPU consumption.
- Any performance optimizations of ATLAS simulation have a big impact on the overall picture.

M. Bandieramonte, University of Pittsburgh
Why multi-threading?

- Evolution trend of faster single-threaded CPU performance broken more than 10 years ago
- Increase of CPU cores and more execution units to overcome stagnation in CPU Clock Speed
  - low power core sharing a pool of memory
- We need a 'Multi-Threaded design (AthenaMT)' to run effectively on modern architectures and profit from multi-core designs
  - MT approach is critical for heterogeneous architectures (e.g. GPU HPCs)
  - This approach will scale better than the existing multi-processor approach (AthenaMP) especially on the architectures that are foreseen to be used in the next LHC runs
- Production ready MT simulation is considered CRITICAL for Run 3 and BLOCKER for Run 4
  - to exploit the HL-LHC successfully
- What about vectorization?

---

M. Bandieramonte, University of Pittsburgh
• The amount of **Monte-Carlo** that can be produced already **limits many physics analyses** and this will get **worse** with the increased luminosity expected.

• The current model, **AthenaMP**, relies on Linux’s *copy-on-write* mechanism for sharing memory pages between forks:
  • won’t scale for Run-3 and beyond

• Ongoing effort to migrate **ATLAS** computing model to **multi-threaded AthenaMT**
  • Finer-grained *task parallelism*, minimised memory footprint
  • Only execute() is concurrent
  • *Scheduler-driven*, by dependency graph

• **Simulation**, **Digitization** and **Reconstruction** moving to MT paradigm using the AthenaMT/GaudiHive infrastructure.
  • Better scaling in terms of memory footprint (leverage new architectures)
  • Easy the investigation of heterogeneous computing architectures (e.g. use GPUs, FPGAs etc)

**Why multi-threading?**
AthenaMT and Geant4MT

- **AthenaMT** is based on **GaudiHive**, a multi-threaded, concurrent-execution extension to Gaudi:
  - Concurrency model based on **Intel® Threading Building Blocks library (TBB)**
    - Computation is broken down into **tasks (building blocks)** that can run in parallel
  - Scheduling is driven by data-flow
  - Events processed in multiple threads

- **Geant4** has its own approach to parallel processing:
  - Master-slave concurrency model, using **pthreads**
  - Provides event-level parallelism
  - Thread safety achieved using **thread-local storage**
    - Main Geant4MT components must be thread-local

- **GaudiHive** provides **task locality, not thread locality**
  - Cannot easily pin a Gaudi component to a specific thread
  - Must decouple the Gaudi components from the Geant4 core functionality
  - Initialization is very tricky: G4 requires that thread-local objects are initialized in their threads at the right time
Thread coupling AthenaMT and G4MT

- Geant4MT has been successfully integrated in AthenaMT outside of the Integrated Simulation Framework (ISF)
  - Inter-event rather than intra-event parallelism:
    - memory saving coming from sharing geometry and cross-section tables between threads
- Segfaults during execution or finalization of MT jobs, due to the way TBB starts new threads:
  - During execution of a MT job:
    - **TBB can spawn new threads** even after initialization is complete
    - The simulation was aborted because the geometry was released after the initialization but it is always needed to initialize new threads
  - When finalizing a MT job:
    - TBB creates extra-threads that are not caught by the ThreadPoolSvc -> no call to G4ThreadInitTool::initThread
      - Crashes when G4ThreadInitTool::terminateThread is called for those threads
Tools to detect thread related issues

M. Bandieramonte, University of Pittsburgh
Collection of data races detected in AthenaMT simulation with **Intel Inspector**:

<table>
<thead>
<tr>
<th>ATLLASSIM-3991</th>
<th>Data race 1 - read/write</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLLASSIM-3992</td>
<td>Data race 2 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-3993</td>
<td>Data race 3 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-3994</td>
<td>Data race 4 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-3995</td>
<td>Data race 5 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-3996</td>
<td>Data race 6 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-3997</td>
<td>Data race 7 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-3998</td>
<td>Data race 8 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-3999</td>
<td>Data race 9 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-4005</td>
<td>Data race 10 - read/write</td>
</tr>
<tr>
<td>ATLLASSIM-4015</td>
<td>Data race 11 - write/read - libRIO.so and AtlasFieldSvc</td>
</tr>
<tr>
<td>ATLLASSIM-4016</td>
<td>Data race 12 - write/read - libG4AtlasAlgLib and libGaudiCoreSvc</td>
</tr>
</tbody>
</table>

**Problem:**

```plaintext
static const G4String tileVolumeString("Tile");
Was not thread-safe, substituted with:
static const char * const tileVolumeString = "Tile" ;
That is initialised before the first call
```
Lock Hierarchy Violations

- Collection of lock hierarchy violations detected in AthenaMT simulation with **Intel Inspector**:
  - It happens when two threads are trying to access and lock two critical sections in a different order. Possible deadlock.

Thread 1:

```
Description: Lock owned
Source: gthr-default.h:748
Function: __gthread_mutex_lock
Module: libAthAllocators.so
Variable: block allocated at ArenaBase.cxx:148

Code snippet:
746   {
>747    if (__gthread_active_p ()))
>748     return __gthr_(pthread_mutex_lock) (__mutex);
>749    else
750    return 0;

Call Stack:
libAthAllocators.so!__gthread_mutex_lock - gthr-default.h:748
libAthAllocators.so!allocator - ArenaBase.icc:40
libGeneratorObjectsTPCnv.so!_ZNZSG2lArenaHandleBaseAllocTINS_18ArenaPoolCv
libGeneratorObjectsAthenaPoolPoolCnv.so!createTransient - TPConverter.icc
libGeneratorObjectsAthenaPoolPoolCnv.so!PoolToDataObject - T_AthenaPoolCv
libAthenaPoolCnvSvcLib.so!createObj - AthenaPoolConverter.cxx:68
libAthenaBaseComp.so!makeCall - AthCnvSvc.cxx:565
libAthenaBaseComp.so!createObj - AthCnvSvc.cxx:261
```

Thread 2:

```
Description: Lock owned
Source: gthr-default.h:748
Function: __gthread_mutex_lock
Module: libAthAllocators.so
Variable: block allocated at ArenaBase.cxx:30
Code snippet:
746   {
>747    if (__gthread_active_p ()))
>748     return __gthr_(pthread_mutex_lock) (__mutex);
>749    else
750    return 0;

Call Stack:
libAthAllocators.so!__gthread_mutex_lock - gthr-default.h:748
libStoreGateLib.so!clearStore - SGImplSvc.cxx:309
libStoreGateLib.so!clearStore - SGHiveMgrSvc.cxx:49
libAthenaServices.so!clearWBSlot - AthenaHiveEventLoopMgr.cxx:1310
libAthenaServices.so!drainScheduler - AthenaHiveEventLoopMgr.cxx:1260
libAthenaServices.so!nextEvent - AthenaHiveEventLoopMgr.cxx:850
libAthenaServices.so!makeCall - AthCnvSvc.cxx:565
```

M. Bandieramonte, University of Pittsburgh
Case study: Differences in LAr Hits

- Differences in the LAr Hits affecting *very rarely* the energy:

  Ex:
  
  Py:diff-root INFO comparing [22] leaves over entries...
  000.LArHitContainer_p2_LArHitEMB.m_energy.6891 3484255171L -> 1336771523L => diff = [22.27205722%]
  Py:diff-root INFO Found [630943] identical leaves
  Py:diff-root INFO Found [1] different leaves

- LArG4SimpleSD instances are contained in SDWrapper, which has a common hit collection container for all SDs.

- The SDWrapper is a derived G4VSensitiveDetector, and instances of SDWrapper are contained in a thread ID-keyed map.

- Problem was likely localized to ProcessHits.
  
  LArCalorimeter/LArG4/LArG4Barrel/src/LArBarrelCalibrationCalculator.cxx
  LArCalorimeter/LArG4/LArG4Barrel/src/LArBarrelPresamplerCalculator.cxx

M. Bandieramonte, University of Pittsburgh
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads
Simulation with 10 tbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 tbar evts)
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)
Strategy for MT debugging

Simulation with 10 $t\bar{t}$bar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 $t\bar{t}$bar evts)
Simulation with 10 ttbar events, *sequential* mode vs *MT* with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)
Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

- Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)
- MT mode, nthreads=5
- Split the HITS file into n files, one per eventID
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)

Sequential mode
- LOG
- HITS

MT mode, nthreads=5
- HITS
- LOG

Split the HITS file into n files, one per eventID

HITS_n

M. Bandieramonte, University of Pittsburgh
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)

Sequential mode

MT mode, nthreads=5

Split the HITS file into n files, one per eventID

LOG

HITS

LOG

HITS

HITS

HITS

Compare on eventID basis the HITS files

M. Bandieramonte, University of Pittsburgh
Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)

MT mode, nthreads=5

Sequential mode

LOG

HITS

Split the HITS file into n files, one per eventID

HITS

LOG

HITS_n

Compare on eventID basis the HITS files

Same?
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)

Sequential mode

LOG

HITS

HITS

LOG

MT mode, nthreads=5

Split the HITS file into n files, one per eventID

HITS_n

HITS_n

HITS_n

HITS_n

Compare on eventID basis the HITS files

Same?

Yes

Go Back to the Start

M. Bandieramonte, University of Pittsburgh
Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

- Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)
- MT mode, nthreads=5
- Split the HITS file into n files, one per eventID
- Compare on eventID basis the HITS files

Same? Yes No

Go Back to the Start

M. Bandieramonte, University of Pittsburgh
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)

Sequential mode

LOG

HITS

MT mode, nthreads=5

HITS

LOG

Split the HITS file into n files, one per eventID

HITS

HITS_n

HITS

HITS_n

HITS

HITS_n

Compare on eventID basis the HITS files

Filter, split and manipulate LOG file, into n files, one per eventID

Same?

No

Yes

Go Back to the Start

M. Bandieramonte, University of Pittsburgh
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

- Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)
- MT mode, nthreads=5
- LOG
- HITS
- LOG
- Split the HITS file into n files, one per eventID
- HITS_n
- Compare on eventID basis the HITS files
- Filter, split and manipulate LOG file, into n files, one per eventID
- Same?
- Yes
- Go Back to the Start
- No
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

- Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)
- MT mode, nthreads=5
- LOG
- HITS
- LOG
- Split the HITS file into n files, one per eventID
- HITS
- LOG
- HITS
- LOG
- HITS
- LOG
- Compare on eventID basis the HITS files
- Yes
- Go Back to the Start
- No
- Filter, split and manipulate LOG file, into n files, one per eventID
- LOG
- LOG
- LOG
- LOG
- LOG

M. Bandieramonte, University of Pittsburgh
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

- Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)
- MT mode, nthreads=5
- LOG
- HITS
- LOG
- Split the HITS file into n files, one per eventID
- HITS_n
- Compare on eventID basis the HITS files
- HITS_n
- LOG_n
- Filter, split and manipulate LOG file, into n files, one per eventID
- LOG_n
- Compare Log files and cross your fingers
- Same?
  - Yes: Go Back to the Start
  - No: Go Back to the Start

M. Bandieramonte, University of Pittsburgh
Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

Increment the messaging level to DEBUG (~7GB for 10 ttbar evts)

Sequential mode
- LOG
- HITS

MT mode, nthreads=5
- LOG
- HITS

Split the HITS file into n files, one per eventID

Compare on eventID basis the HITS files

Filter, split and manipulate LOG file, into n files, one per eventID

Compare Log files and cross your fingers

Log files
- LOG_n

Same?
- Yes
- No

Cross your fingers again…

Go Back to the Start

M. Bandieramonte, University of Pittsburgh
Strategy for MT debugging

Simulation with 10 ttbar events, sequential mode vs MT with 5 threads

**EMBPresamplerCalculator**

10673650  EMBPresamplerCa... DEBUG  module,x0,y0,current0 from map 5 0.541834 0.234809 [-1.03946-] {+1.08644+}
10673651  EMBPresamplerCa... DEBUG  Energy for sub step 0.0246629
10673652  EMBPresamplerCa... DEBUG  set current map for module 5
10673653  EMBPresamplerCa... DEBUG  module,x0,y0,current0 from map 5 0.523319 0.204478 [-1.00652-] {+1.04875+}
10673654  EMBPresamplerCa... DEBUG  Energy for sub step 0.0246629
10673655  EMBPresamplerCa... DEBUG  set current map for module 5
10673656  EMBPresamplerCa... DEBUG  module,x0,y0,current0 from map 5 0.504805 0.174147 [-0.973578-] {+1.01106+}
10673657  EMBPresamplerCa... DEBUG  Energy for sub step 0.0246629
10673658  EMBPresamplerCa... DEBUG  set current map for module 5
10673659  EMBPresamplerCa... DEBUG  module,x0,y0,current0 from map 5 0.48629 0.143817 [-0.932291-] {+0.970216+}
10673660  EMBPresamplerCa... DEBUG  Hit Energy/Time [-0.175662-] {+0.179722+} 87.253

Cross your fingers again...
Compare Log files and cross your fingers
LOG_n
Filter, split and manipulate LOG file, into n files, one per eventID
LOG_n

No
Same?

Yes
Go Back to the Start
The problem & the fix

**PsMap** is a singleton and **SetMap()** method was not thread-safe:
- set the current “Current map” in its private member **m_curr**
- store the module in **m_module**.

**LArBarrelPresamplerCalculator.cxx**

```cpp
m_psmap->SetMap(imodule);
m_psmap->Map()->GetAll(x0,y0,&gap,&current0,&current1,&current2);
```

**PsMap.cxx**

```cpp
void PsMap::SetMap(int module)
{
  if (m_module==module) return;
  m_module=module;
  [...] if (m_theMap.find(code) != m_theMap.end())
    m_curr = m_theMap[code];
  else {
    m_curr=0;
  }
}
```

- **Data race** in the **LArBarrelPresamplerCalculator**:
- **SetMap** could be called by another thread before the current values were obtained.
PsMap is a singleton and SetMap() method was not thread-safe:
• set the current “Current map” in its private member m_curr
• store the module in m_module.

Data race in the LArBarrelPresamplerCalculator:
• SetMap could be called by another thread before the current values were obtained

Remove SetMap() function and m_curr and m_module members
• Implement CurrMap* GetMap(imodule) const method
PsMap is a singleton and \texttt{SetMap()} method was not thread-safe:

- set the current “Current map” in its private member \texttt{m\_curr}
- store the module in \texttt{m\_module}.

\texttt{LArBarrelPresamplerCalculator.cxx}

\texttt{CurrMap* cm = m\_psmap->GetMap(imodule);} 

\texttt{PsMap.cxx}

\texttt{CurrMap* PsMap::GetMap(int module) const}

\begin{verbatim}
{ 
 [..]
  auto it = m\_theMap.find(code);
  if (it != m\_theMap.end())
    return it->second;
  else {
    return nullptr;
  }
}
\end{verbatim}

\begin{itemize}
  \item \textbf{Data race} in the \texttt{LArBarrelPresamplerCalculator}: 
    \begin{itemize}
      \item \texttt{SetMap} could be called by another thread before the current values were obtained 
    \end{itemize}
  \item Remove \texttt{SetMap()} function and \texttt{m\_curr} and \texttt{m\_module} members 
  \item Implement \texttt{CurrMap* GetMap(imodule) const} method 
\end{itemize}

Tested with 100 ttbar with MT 10 threads \textbf{Solved!}
AthenaMT & Geant4MT validation

- Recent progress highlights:
  - **Output** validation:
    - Fixed: thread-unsafety causing difference in HITS of LAr sensitive detector (~1-2%)
    - Fixed: thread-unsafety causing difference in HITS of Tile sensitive detector (~1-5%)
    - Fixed: simulation with CaloCalibrationHit (~50% of Dead material hits)

We can now run reliably full single-threaded and multi-threaded simulations, results are fully consistent (read identical)! physics validation in progress.

- Confirmed reproducibility of simulation with **SUSY/Exotics G4Extensions** enabled:
  - We currently have six packages which add support for additional particles and physics processes to Geant4
    - **Charginos** - Stable and Decaying Charginos – OK ✔️
    - **Gauginos** - OK ✔️
    - **Neutralinos** - Decaying - OK ✔️ ✔️
    - **Monopoles** - OK ✔️
    - **Quirks** - Postponed - lack of samples (no associated physics analysis) ❌
    - **RHadrons** - waiting for samples ❌
    - **Sleptons** - Stable, Decaying taus, Decaying light - OK ✔️
AthenaMT vs AthenaMP benchmarks

- **Architecture:** x86_64
- **CPU op-mode(s):** 32-bit, 64-bit
- **Byte Order:** Little Endian
- **CPU(s):** 32
- **On-line CPU(s) list:** 0-31
- **Thread(s) per core:** 2
- **Core(s) per socket:** 8
- **Socket(s):** 2
- **NUMA node(s):** 2
- **Model:** 79
- **Model name:** Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz

Test on 100 ttbar events, with prom
Athena, r2019-09-30T2130,master

Results are AVG of 5 separate runs (from 1-32 threads/processes) - the machine was quiet all the time (me as only user)

**AthenaMT Speedup\(th_n\) = Wall-time\(th_1\)/Wall-time\(th_n\)**

**AthenaMP Speedup\(proc_n\) = Wall-time\(proc_1\)/Wall-time\(proc_n\)**

<table>
<thead>
<tr>
<th>Wall-Time [min.]</th>
<th>1 thread/process</th>
</tr>
</thead>
<tbody>
<tr>
<td>AthenaMT</td>
<td>169.6733333</td>
</tr>
<tr>
<td>AthenaMP</td>
<td>173.9166667</td>
</tr>
</tbody>
</table>
The Proportional Set Size (PSS) is the portion of main memory occupied by a process and is composed by the private memory of that process plus the proportion of shared memory with one or more other processes.
Summary

• The Athena Multi-threaded simulation with Geant4MT is fully functional

• Outside of ISF:
  • The G4 single threaded vs multi-threaded output has been confirmed to be identical
  • 100k grid test were ran with 8 cores without reported issues (physics validation in progress)

• Inside ISF:
  • After revising the Geant4 initialisation steps in MT mode, simulation runs correctly in multi-threaded mode with 1 thread and the output has been validated

• Next steps
  • Solve the thread-unsafely issues and assure that G4MT simulation works with more than one thread and that the results are reproducible
Thanks for your attention!

Marilena Bandieramonte
marilena.bandieramonte@cern.ch
Backup slides
AthenaMT vs AthenaMP benchmarks

Test on 100 ttbar events, with prom Athena, r2019-09-30T2130,master

results are AVG of 5 separate runs (from 1–32 threads/processes) – the machine was quiet all the time (me as only user)
AthenaMT vs AthenaMP benchmarks

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 32
On-line CPU(s) list: 0-31
Thread(s) per core: 2
Core(s) per socket: 8
Socket(s): 2
NUMA node(s): 2
Model: 79
Model name: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz

Test on 100 ttbar events, with prom
Athena, r2019-09-30T2130,master

results are AVG of 5 separate runs (from 1-32 threads/processes) - the machine was quiet all the time (me as only user)