Multi-threaded simulation for ATLAS: challenges and validation strategy

Marilena Bandieramonte[1,2,*], *John Derek* Chapman[1,3], *Justin* Chiu[1,4], *Heather* Gray[1,5], and *Miha* Muskinja[1,5]

[1]CERN, EP Department, Meyrin, 1211, Switzerland
[2]University of Pittsburgh, Pittsburgh, PA 15260, USA
[3]Cavendish Laboratory, University of Cambridge, Cambridge, CB2 1TN, UK
[4]University of Victoria, Victoria, BC V8P 5C2 Canada
[5]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Abstract. Estimations of the CPU resources that will be needed to produce simulated data for the future runs of the ATLAS experiment at the LHC, indicate a compelling need to speed-up the process to reduce the computational time required. While different fast simulation projects are ongoing (FastCaloSim, FastChain, etc.), full GEANT4 based simulation will still be heavily used and is expected to consume the biggest portion of the total estimated processing time. In order to run effectively on modern architectures and profit from multi-core designs a migration of the Athena framework to a multi-threading processing model has been performed in the last years. A multi-threaded simulation based on AthenaMT and GEANT4MT, enables substantial decreases in the memory footprint of jobs, largely from shared geometry and cross-section tables. This approach scales better with respect to the multi-processing approach (AthenaMP) especially on the architectures that are foreseen to be used in the next LHC runs. In these proceedings we will report about the status of the multi-threaded simulation in ATLAS, focusing on the different challenges of its validation process. We will demonstrate the different tools and strategies that have been used for debugging multi-threaded runs versus the corresponding sequential ones, in order to have a fully reproducible and consistent simulation result.

ATL-SOFT-PROC-2020-030
14 March 2020

1 Introduction

High-energy physics (HEP) experiments at the Large Hadron Collider (LHC) are preparing for the next LHC runs that are respectively scheduled to start in 2021 (so called Run 3) and at the end of 2027 (so called Run 4 or High-Luminosity phase - HL-LHC), performing and planning major upgrades to their detectors. For example, during Run 3 the ALICE experiment will increase the collision detection rate to 50kHz, while the number of collisions that the LHCb experiment will have to process will grow to more than 40 times of what it does today. The transition to the HL-LHC phase will involve substantial changes for the ATLAS

*corresponding author e-mail: marilena.bandieramonte@cern.ch

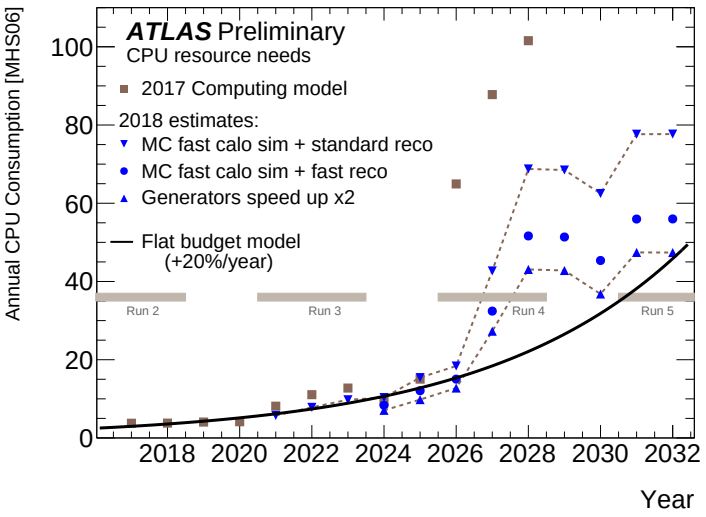


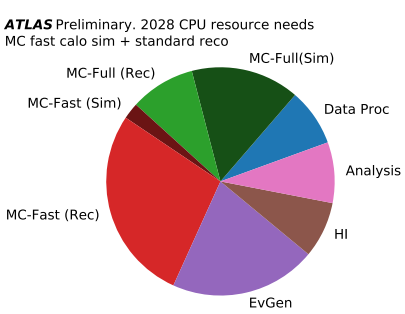
Figure 1: Estimated CPU resources (in MHS06) needed for the years 2018 to 2032 for both data and simulation processing. The plot updates the projection made in 2017 (which was based on the Run 2 computing model) with updated LHC running conditions and revised scenarios for future computing models [2].

[1] and CMS experiments. With HL-LHC the instantaneous luminosity will become 5 times greater than its nominal value $L_0 = 10^{34}\text{cm}^{-2}\text{s}^{-1}$ (and up to 7.5 L_0 as the ultimate limit), by operating in a "leveled" mode, i.e. with constant luminosity over a significant length of time, allowing to increase tenfold the integrated luminosity : 3000fb^{-1} at the end of life of the HL-LHC, around 2037. The increase in the intensity of the beam will allow for collisions where the number of proton-proton interactions (pile-up) will increase from the current value of 60 to 200. All of this implies necessary changes to the detector hardware and software. In particular, both experiments will modify their trigger systems to improve their quality and speed, with the aim of recording $5 - 10$ times the number of current events. As mentioned, an increased luminosity implies a bigger number of events and therefore an increase in the statistics collected which enables new physics studies and discoveries. However it should be borne in mind that, in general, the amount of data that experiments will be able to collect and process in the future is limited by constraints in terms of available budget for computational and software resources and by the efficiency with which these resources are exploited.

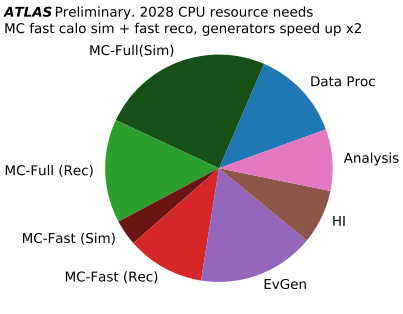
Figure 1 shows ATLAS numbers that are particularly interesting in that they estimate CPU resources (in MHS06) needed for the years 2018 to 2032 for both data and simulation processing. These are compared to the projection (solid line) of what is expected to be available in a flat budget scenario that assumes an increase of 20% per year, in light of the technology trends at the start of 2018.

The brown squares show what would be the ATLAS computing needs if we would keep the same computing model parameters from 2017. As can be seen, the gap between needs and bare technological advances is significant. The blue points show what kind of improvement is achievable in three different scenarios, which require a non negligible development effort: (1) the top curve assumes that fast calorimeter simulation (FastCaloSim) is used for 75% of the Monte Carlo simulation; (2) the middle curve assumes that, in addition, a faster version

of reconstruction will be used, which is seeded by the event generator information; (3) the bottom curve, adds a speedup in the event generation assuming that the time is halved, either by software improvements or by re-using some of the events. In Run 3, ATLAS plans to run at least 50% of simulation with fast techniques (the aim is to reach 75%), but full GEANT4 [3–5] simulation will be heavily used regardless. If we look at how this computational time would be split into components in two of the different envisaged scenarios, the (1) represented in Figure 2a and the (3) in Figure 2b, we can ultimately see that even in the best scenario case the Full Simulation will be the largest CPU consumer and, together with the Fast Simulation and the Fast Reconstruction will account for $\simeq$ 40% of the total expected CPU consumption.



(a) CPU consumption if using FastCaloSim for 75% of the Monte Carlo simulation and standard reconstruction.



(b) CPU consumption using faster version of reconstruction, which is seeded by the event generator information, and assuming event generation is sped up by a factor of two.

Figure 2: Fraction of CPU resources needed in 2028 at the end of Run 4 for different processing workflows.The "MC-Full" section in green is related to the fraction of time spent on the full AtlasG4 simulation and divided in a simulation part "(Sim)" for the GEANT4 simulation and a reconstruction part "(Rec)" accounting the time spent reconstructing the events. Similarly, the "MC-Fast" section in red shows this distribution for the time spent running the FastCaloSim simulation [2].

It is clear that any performance optimization that could be done on the Full Simulation would have a big impact on the overall picture. In this context, one of the biggest project in which the ATLAS collaboration is concentrating its effort is the migration of the ATLAS software framework, Athena [6, 7] to a multi-threaded design.

2 Motivation for AthenaMT

The reader is probably familiar with the plot in Figure 3 that shows the evolution trend of CPUs over the past 50 years. It can be seen that faster single-threaded CPU performance broke more than 10 years ago and while the number of transistors kept growing, a stagnation of the CPU clock speed aroused, being overcome by an increase in the number of low-power cores per CPU, that share a smaller pool of memory.

In order to run effectively on modern architectures and profit from multi-core designs a Multi-Threaded (MT) design is needed and the MT approach is critical for heterogeneous architectures (e.g. GPU, HPCs). Furthermore, if we concentrate on the ATLAS simulation, it has to be noticed that the amount of Monte-Carlo that can be produced already limits many

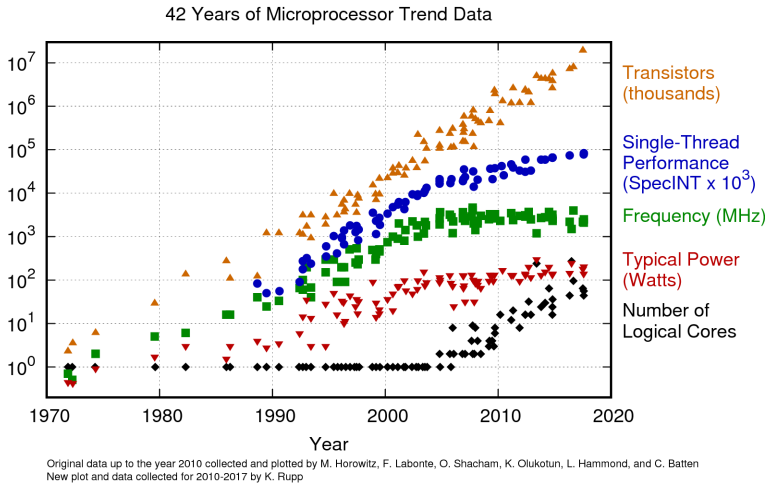


Figure 3: History of Intel chip introductions by clock speed and number of transistors over the past 50 years. [8]

physics analyses and with the increased luminosity much higher statistics of simulated events will be needed, worsening the situation. The current ATLAS model, AthenaMP [9] shown in Figure 4, relies on Linux's copy-on-write mechanism for sharing memory pages between forks: this approach allows for memory saving but it won't scale for Run 3 and beyond. An MT approach would instead scale better than the existing one especially on the architectures that are foreseen to be used in the next LHC runs. This is why the ATLAS collaboration undertook a migration activity of its computing model to a new multi-threaded design, called AthenaMT [10, 11]. The goal is to move to a finer-grained task parallelism, with a better scaling in terms of memory footprint, that can leverage new architectures easing the investigation of heterogeneous computing architectures (e.g. use GPUs, FPGAs, etc.). Simulation, Digitization and Reconstruction are moving to an MT paradigm using the AthenaMT/GaudiHive [12] infrastructure. Production ready AthenaMT Simulation is considered critical for Run 3 and a blocker activity for Run 4.

2.1 AthenaMT and Geant4MT

AthenaMT is based on GaudiHive, a multi-threaded, concurrent-execution extension to Gaudi [13]: its concurrency model is based on Intel® Threading Building Blocks library (TBB) [14] and the computation is broken down into tasks (building blocks) that can run in parallel. The scheduling is driven by data-flow and events are processed in multiple threads. However, Geant4 has its own approach to parallel processing, adopting a master-slave concurrency model, using pthreads. A multi-threaded version of the Geant4 toolkit, Geant4MT [15] has been available since the version 10 series introduced in 2014. It provides event-level parallelism and thread safety is achieved using thread-local storage. The main Geant4MT components must be thread-local. Instead GaudiHive provides task locality, not thread locality. It is not easy to pin a Gaudi component to a specific Geant4 thread: one must decouple the Gaudi components from the Geant4 core functionality. This explains why the coupling between Gaudi and Geant4 is very tricky: Geant4 requires that thread-local objects are initialized in their threads at the right time. Nevertheless, after solving some issues caused for example by

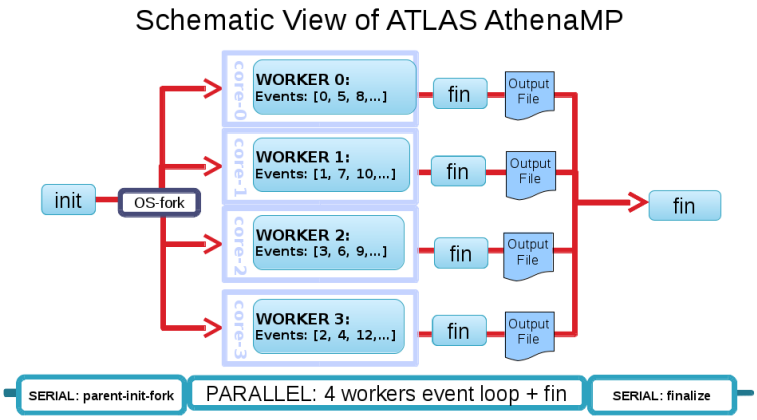


Figure 4: Schematic view of AthenaMP, based on Linux copy-on-write sharing mechanism [2].

the way that TBB sometimes spawns extra-threads after the initialization phase, GEANT4MT has been successfully integrated in AthenaMT outside and inside of the Integrated Simulation Framework (ISF), as will be described in the next section. This should enable memory saving coming from sharing the GEANT4 geometry and the cross-section tables between different threads.

3 Validation strategy and tools

Once the migration of ATLAS Simulation to a multi-threaded design was completed, debugging and validation of the multi-threaded ATLAS simulation started. The scope of this activity was to find and correct defects that arise in either GEANT4 software, other external packages, or ATLAS code. The adopted validation strategy was the following: first concentrate on differences between single-threaded simulation, and multi-threaded simulation running in single thread mode, then after correcting problems that arise already in this simple configuration, transition to debugging multiple threads in multi-threaded simulation. To help this debugging activity the powerful Intel Inspector tool [16] has been used to locate memory faults, deadlocks, and race conditions. These have brought to light a host of issues with the thread-safety of ATLAS code and externals upon which it depends.

In order to reduce the complexity of the system analysed, the first milestone was to validate the simulation outside of the ISF, with the goal of having a fully reproducible and consistent simulation result. This is possible, despite the stochastic nature of any Monte Carlo simulation, setting the seed of the Pseudo Random Number generator (PRNG) used in the simulation, to the same value. In this way the output of the simulation process, that is stored in the so called HITS files, is bitwise comparable between different runs that use the same input file.

Initially, when simulating a *ttbar* event in sequential mode and multi-threaded mode with one thread, the two simulations were diverging at some point. The divergence was observed in the *safety* value that was retrieved while being in a specific volume. The source of the discrepancy in the safety calculation between sequential and multi-threaded mode, was identified debugging the code with the GNU debugger (GDB). We found that the *smartlessness* voxel density parameter in the MT job was not being set correctly. This resulted in a different

voxelization of the volume and consequently in a different safety calculation. Once this issue has been solved the output of the sequential and single-threaded simulation resulted bitwise identical.

The second step was to test the execution of the multi-threaded simulation with more than one thread vs the corresponding sequential execution. This required some manipulation of the output files in order to split the HITS of a MT execution in different files per event, in order to compare the output of each event in the correct order and in a consistent way. This comparison highlighted issues related to race conditions on different parts of the code:

- thread-unsafety causing differences in the hits of the LAr sensitive detector (1-2%)
- thread-unsafety causing differences in the hits of the Tile sensitive detector (1-5%)
- thread-unsafety in the *CaloCalibrationHit* code (50% of Dead material hits)

All these issues have been tracked down and solved. The next step was to confirm the reproducibility of simulation with SUSY/Exotics extensions to the GEANT4 physics enabled. The following packages have been checked: Charginos (Stable and decaying), Neutralinos (Decaying), Sleptons+Gauginos (Stable, Decaying taus, Decaying light) and Monopole. The reproducibility in MT mode of the Monopole package revealed a thread-unsafety issue that has been solved. At this point we had GEANT4 running on multiple threads as part of an AthenaMT job and outside of the ISF, giving identical output to GEANT4 running in a standard Athena job. The next fundamental milestone was to validate the multi-threaded GEANT4 full simulation when running it from inside ISF. ISF is not yet thread-safe and also the initialization chain of GEANT4 simulation in MT mode needed to be re-organized and cleaned up in order to run properly. This activity followed the same concept and validation strategy steps as the first one, so after making reproducible the output when running with one thread we concentrated on the thread-safety issues arising when running with multiple threads.

This validation activity has been successfully completed and now GEANT4MT simulation can run in a thread-safe way from inside the ISF and the output is bitwise identical and reproducible w.r.t. the sequential runs.

4 Benchmarking AthenaMP vs AthenaMT

Once multi-threaded simulation was proved to be safe and functioning, we conducted a benchmark campaign to assess performance and speedup scale factors of AthenaMT versus AthenaMP. These benchmarks were obtained running a standard *ttbar* full simulation. The following plots were produced running on a local server, *pitt-buildnode-01*, with the following specs:

```
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
```

It has to be noted that the machine has 2 CPUs with 8 cores each, so 16 physical cores, 32 logical cores (in hyper-threading regime). Results are averages of 5 separate runs (from 1-32

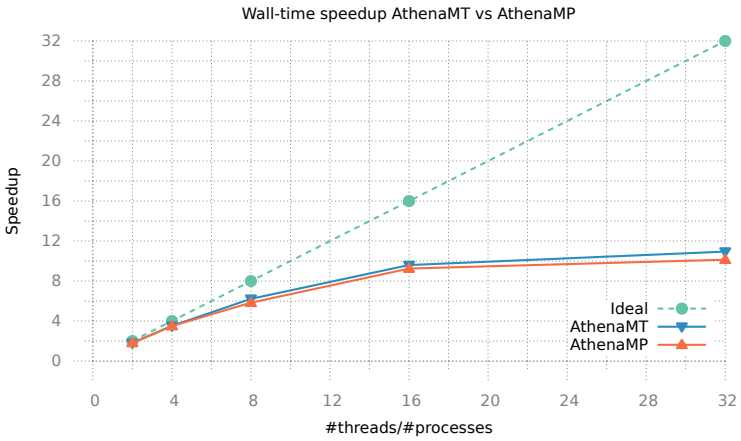


Figure 5: Wall-Time speedup of AthenaMT (blue) vs. AthenaMP (orange), compared with the ideal linear scaling.

threads/processes) and the machine was quiet all the time. The speedup is defined as follows:

$$AthenaMT - Speedup_{nthreads} = Wall - time_{th_1} / Wall - time_{th_n}$$

$$AthenaMP - Speedup_{nprocs} = Wall - time_{proc_1} / Wall - time_{proc_n}$$

4.1 Strong scaling benchmarks

The following results show strong scaling speedup factors of AthenaMT vs AthenaMP as a function of the number of threads/number of processes when keeping constant the total number of events processed to 100 *ttbar* events. Figure 5 shows the wall-time speedup of AthenaMT in blue, versus AthenaMP in orange.

As can be seen, AthenaMT scales in a very similar way with respect to AthenaMP and the speedup is almost linear up to the number of physical cores of the machine. However, in the hyper-threading regime we observe a loss of scalability and the speedup almost reaches a plateau, meaning that the code cannot profit from the number of logical cores available with the hyperthreading technology. Also we notice that both in the case of AthenaMT and AthenaMP even if up to 8 cores the code scales almost ideally, we observe a degradation of the performance when trying to use more than 8 cores. This effect might be due to the fact that 100 events are not keeping busy enough the threads/processes, and as a consequence, for higher numbers of threads/processes the initialization time might be significant with respect to the event loop time, impacting the overall results. For this reason a new benchmark campaign, that is presented in the next paragraph, was conducted to measure the weak scaling factors keeping constant the load per thread/process (i.e. 50 events).

Figure 6 shows the Proportional Set Size (PSS) that measures the portion of main memory occupied by a process and it is composed of the private memory of that process plus the proportion of shared memory with one or more other processes.

It can be noticed that the AthenaMT memory footprint scales better with the number of threads, than AthenaMP with the number of processes. In particular we see an increase in the memory occupancy of $\sim$ 27.25 MB per thread in AthenaMT while there is a $\sim$ 214.33 MB increase per process in AthenaMP. This is the most significant memory occupancy metric in

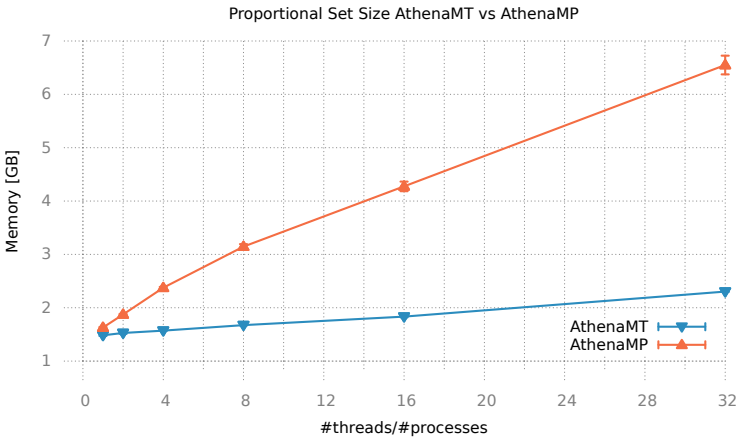


Figure 6: Memory Occupancy (Proportional Set Size) of AthenaMT (blue) vs. AthenaMP (orange), as a function of the number of threads or processes, respectively.

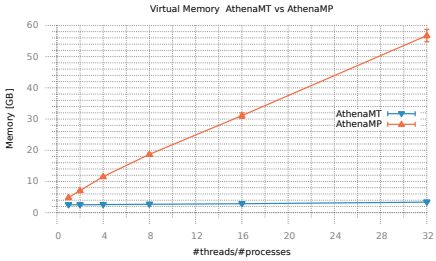


Figure 7: Memory Occupancy (Virtual Memory) of AthenaMT (blue) vs. AthenaMP (orange), as a function of the number of threads or processes, respectively.

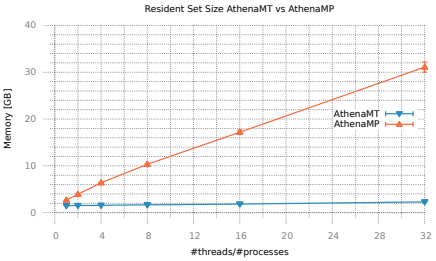


Figure 8: Memory Occupancy (Resident Set Size) of AthenaMT (blue) vs. AthenaMP (orange), as a function of the number of threads or processes, respectively.

order to compare a multi-threaded application vs. a multi-process one. In fact, the PSS takes into account both the memory that is shared between processes and between threads.

To explain the difference between the various memory occupancy metrics, Figure 7 shows the Virtual Memory (VMEM) occupancy, while Figure 8 shows the Resident Set Size (RSS) plot. The Virtual Memory metric includes all memory that the process can access, including memory that is swapped out, memory that is allocated, but not used, and memory that comes from shared libraries.

The RSS is the portion of memory occupied by a process that is held in main memory. It does not include memory that is swapped out, but includes memory from shared libraries as long as the pages from those libraries are actually in memory. It does include all stack and heap memory. However this metric doesn't consider if part of the memory occupied by a process is shared with other processes, i.e. if a portion of memory is shared between 2 processes it will be counted twice. This explains why the memory footprint numbers of VMEM and RSS are much higher for AthenaMP than AthenaMT. The latest 2 plots are

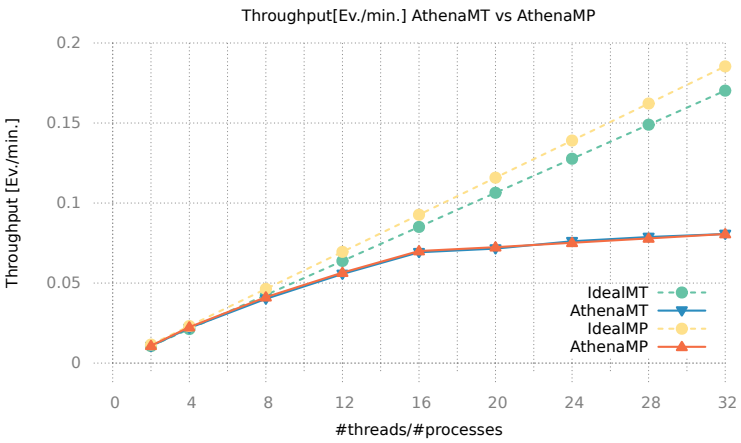


Figure 9: Wall-Time speedup of AthenaMT (blue) vs. AthenaMP (orange), compared with the ideal linear scaling in case of weak scaling.

included only for educational purposes, since they don't give a real indication of the memory occupancy in the 2 different paradigms considered in this study.

4.2 Weak scaling benchmarks

As introduced in the previous paragraph, a second benchmark campaign was conducted to assess the weak scaling performance of AthenaMT vs. AthenaMP. In this second benchmark the load of each thread/process has been kept constant to 50 *ttbar* events. In this way we ensure that each thread/process has a significant load to execute, in order to reduce the impact of the initialization time on the overall execution time. Figure 9 shows the throughput in events per minute, of the 2 paradigms as a function of the number of threads/processes and with respect to their ideal scaling curves. AthenaMT shows again to scale in a very similar way as AthenaMP which is an impressive achievement. As speculated, this time we can observe that the gap between the ideal and the real curve is significantly reduced below 16 threads/processes, revealing an almost ideal scaling behaviour. Furthermore we notice that in hyper-threading regime, the loss of performance is persistent, as expected.

5 Conclusions

With the new multi-core computing era, multi-threaded programming models became necessary to exploit modern computing architectures. This paper presented the commissioning and validation process of ATLAS multi-threaded simulation based on AthenaMT and GEANT4MT, describing the different challenges faced. We demonstrated the different tools and strategies that have been used to successfully have a fully reproducible and bitwise consistent simulation result when running in multi-threaded mode with respect to sequential mode. Benchmark campaign results demonstrated that ATLAS multi-threaded simulation enables substantial decreases in the memory footprint of jobs while keeping an almost ideal scaling behaviour in terms of speedup and throughput. This approach scales significantly better in terms of memory occupancy with respect to the multi-processing approach (AthenaMP) and this effect is expected to be more visible on the architectures that are foreseen to be used in the future LHC runs.

References

[1] The ATLAS Collaboration, Journal of Instrumentation **3**, S08003 (2008)

[2] *The ATLAS Collaboration, Computing and software public results*, https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults

[3] S. Agostinelli et al. (GEANT4), Nucl. Instrum. Meth. **A506**, 250 (2003)

[4] J. Allison et al., IEEE Trans. Nucl. Sci. **53**, 270 (2006)

[5] J. Allison et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **835**, 186 (2016)

[6] *The Athena Framework*, https://atlassoftwaredocs.web.cern.ch/athena/athena-intro/

[7] P. Calafiura, W. Lavrijsen, C. Leggett, M. Marino, D. Quarrie, *The Athena control framework in production, new developments and lessons learned*, in *Computing in high energy physics and nuclear physics. Proceedings, Conference, CHEP'04, Interlaken, Switzerland, September 27-October 1, 2004* (2005), pp. 456–458

[8] K. Rupp, *42 Years of Microprocessor Trend Data*, https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

[9] P. Calafiura, C. Leggett, R. Seuster, V. Tsulaia, P.V. Gemmeren, Journal of Physics: Conference Series **664**, 072050 (2015)

[10] P. Calafiura et al., Journal of Physics: Conference Series **664**, 072031 (2015)

[11] C. Leggett et al., Journal of Physics: Conference Series **898**, 042009 (2017)

[12] M. Clemencic, B. Hegner, P. Mato, D. Piparo, Journal of Physics: Conference Series **513**, 022013 (2014)

[13] G. Barrand et al., Comput. Phys. Commun. **140**, 45 (2001)

[14] *Intel® Threading Building Blocks*, https://software.intel.com/en-us/tbb

[15] X. Dong et al., Journal of Physics: Conference Series **396**, 052029 (2012)

[16] *Intel® Inspector*, https://software.intel.com/en-us/inspector