EPJ Web of Conferences **214**, 02029 (2019) https://doi.org/10.1051/epjconf/201921402029
CHEP 2018

A scalable and asynchronous detector simulation system based on ALFA

Sandro Wenzel[1,*]

[1]CERN, Route de Meyrin, 1211 Geneva, Switzerland

Abstract. In the context of the common online-offline computing infrastructure for Run3 (ALICE-O2), ALICE is reorganizing its detector simulation software to be based on FairRoot, offering a common toolkit to implement simulation based on the Virtual-Monte-Carlo (VMC) scheme. Recently, FairRoot has been augmented by ALFA, a software framework developed in collaboration between ALICE and FAIR, offering portable building blocks to construct message-based and loosely-coupled multiprocessing systems.
We will report here on the implementation of a scalable and asynchronous detector simulation system which is based on ALFA. The system offers parallelization at the primary-track level, going beyond the usual inter-event parallelism of Geant4-MT, and the possibility to asynchronously and simultaneously process simulation data for the purpose of digitization and clusterization. Core advantages of our implementation are an ideal reduction of the processing time per event as well as a reduction of the memory footprint, allowing us to make significantly better use of opportunistic resources, such as HPC backfills. Moreover, the track-level parallelism opens up the interesting possibility to use different simulation engines (such as Geant4 and Fluka) concurrently, based on simple selection filters on the primary particles. The integration of fast MC processes, such as machine learning kernels running on a dedicated GPU, are a natural extension to the system.

1 Introduction and Motivation

Detector simulation, transforming computer generated collisions into simulated hardware/readout signals, is a pillar of any software stack in high-energy physics (HEP) experiments. At the same time it is one of the major consumers of compute time and potentially memory, depending on the use case. Logically, there is an ever increasing interest to make these simulations scalable and deployable on all sorts of computing equipment including high-performance computing (HPC) facilities, next to the focus of tuning/optimizing the algorithm themselves. In short, this work describes a generic and scalable system achieving these goals based on message passing and which is obtained through easy migration of our existing code.

Typically in HEP, detector simulation is built around the Geant4 toolkit [1] natively or using the so called Virtual Monte-Carlo (VMC) layer [2]. The latter enables the possibility to also use other simulation engines such as Geant3 [3] and Fluka [4]. ALICE and FAIR

*e-mail: sandro.wenzel@cern.ch; On behalf of the ALICE collaboration

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

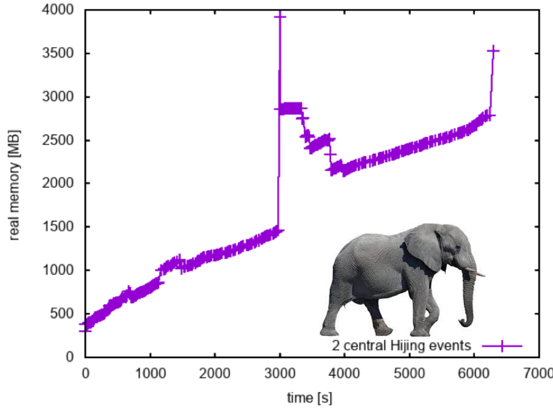

Figure 1. Illustration of the resources required to simulate two heavy ion collisions, which can be very demanding both on CPU time and memory. Numbers only indicating a typical scale. Due to the resources needed for each event, we can think of them as "elephant" tasks.

experiments are using the VMC approach which typically consists of describing the detector geometry using TGeo/ROOT [5] and implementing the sensitive detector response using API and hooks offered by VMC. For LHC Run3, ALICE and FAIR are using common software components, in particular FairRoot [6], reducing much the boilerplate code to setup a VMC detector simulation.

For an experiment studying heavy-ion collisions, like ALICE, one of the major challenges in simulation is that these events can be very complex, with thousands of primary particles emerging from a single interaction. In contrast to the much smaller pp events, this can lead to a very long CPU time required to fully obtain the detector response for any single event, which can be on the order of an hour (see figure 1 for an illustration) – for simplicity we consider here only simulation up to the "hits" level. Similary, for instance with the standard IO and data model implemented by FairRoot, a lot of memory might be required to keep the simulation products for one event in memory before flushing.

One of the reasons for this is the fact that most simulation engines and/or the frameworks typically treat an event as an *atomic unit of work*. Next to requiring considerable resources to treat such a workable chunk, this is typically bad or sub-optimal for scheduling in case of large events. Moreover, it is currently impossible to access and benefit from any opportunistic computing resource that offers a time window considerably smaller than what such a single atomic task needs.

1.1 Goal and Outline

This work originated from the motivation to improve upon the situation described above. Most importantly, the primary goal was to gain the possibility of accessing opportunistic HPC or high-throughput resources. These typically provide lots of CPU cores on which to scale out but in a very short or constrained time window. Secondly, we would like to be able to do so while keeping the VMC simulation approach.

The first goal is not compatible with the current approach to simulate an event as an unbreakable unit. The very large resources needed per event – both in time and memory – make scaling out at least a challenge. Therefore, one clearly needs to provide a way to split up events into smaller work chunks that can be treated each at a time. Indeed, since in modern HEP Monte-Carlo and transport codes (like Geant4), simulated and/or transported particles do not interact with each other, and material properties are not affected by deposited energy, it should be allowed to split event tracks in chunks and process them separately and independently. Such sub-events consisting of a group of primary particles, can then be simulated in

a fraction of the resources compared to a full event. By tuning the size of these sub-events, essentially even small CPU time windows can be dispatched to and utilized. It is clear that this requires additional logic and book-keeping to split the collisions and to reassemble the results into an accumulated output for the whole event.

The second goal to scale out on a many-core node *and across* typically requires some combination of multi-threading (MT) and multi-processing. Within the VMC ecosystem, MT is only offered by Geant4 and it is not feasible to work on multi-threading support for other engines. Scaling out with multi-processing, on the other hand, seems to be a very viable, convenient and universally applicable option (especially in cases where multiple threads don't touch the same data structures, as is typically the case).

In outline, this work describes a scalable multi-processing architecture with components collaborating through messaging to achieve both of the primary goals as well as many secondary emerging advantages and possibilities as discussed below.

2 System Description

2.1 Messaging and Asynchronous Computing with FairMQ/ALFA

For LHC Run3, ALICE is developing common software components together with the FAIR experiments within the so-called ALFA project [7]. FairRoot for example, with its toolkit for VMC simulation, is one essential part of ALFA.

Other essential parts are FairMQ [8] and the Dynamic Deployment System (DDS) [9], which provide an easy to use API to implement asynchronously processing components interacting via message queues as well as workload distribution/deployment on compute farms. Development of these components was largely influenced by the ALICE-O2 project [10], targeting an integrated online-offline computing architecture and workflow based on the experience with the multi-process design and data flow in the ALICE High-Level Trigger [11].

In essence, FairMQ makes it possible to program and setup a system of heterogenous and asynchronous actors (processes) that communicate via sending and receiving messages. Individual actors (or "devices" in FairMQ-speak) can be setup in only a few lines of code and

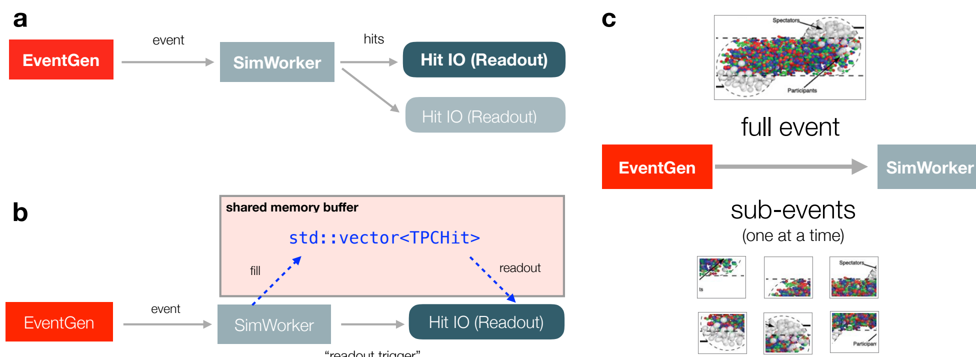

Figure 2. Main components and ideas of the simulation system. (a) Splitting simulation into multiple actors of event generator/server, simulation worker and one or potentially more IO processes. (b) Actors can share memory buffers to reduce copying/transporting data. (c) The event generator can distribute whole events or sub-events consisting of groups of primary particles which can be transported individually.

building reactive systems is made easy by offering a callback mechanism which is invoked automatically whenever an actor receives incoming data. ROOT serialization can be used to exchange complicated C++ objects between the actors.

FairMQ is implemented on top of well known transport layers such as ØMQ [12], nanomessage [13], shared memory and others but hides these details to the user. As such, a FairMQ topology (the group of all actors and communication links) is easily scalable from a single node to a complex – potentially heterogeneous – cluster of CPUs.

Even though FairMQ was not designed with detector simulation in mind, it was realized that it gives us a *convenient layer* to solve our goals without much effort or complicated refactoring. However, nothing below depends fundamentally on FairMQ and the general ideas could be implemented with many other messaging systems.

2.2 System Architecture

Here we describe the high-performance detector simulation system based on an elastic diamond architecture composed of FairMQ devices/actors with specialized concern.

2.2.1 Basic Components

Taking the initial monolithic simulation process – handling everything from event generation to IO – we start progressively by setting up three independent actors: an event generation process; a simulation worker; and a hit collector / IO process (see figure 2(a)). Each of them is a FairMQ device, and as their name indicates they perform dedicated tasks. The event generator produces collisions and forwards them to the simulation worker – when the worker request so in a request-reply pattern. The worker performs the actual detector simulation using a VMC engine and forwards the generated hits to the hit merger. The latter one simply takes care of writing them to a ROOT file. Possibly, one can choose to have multiple IO actors, each taking care of hits of a particular sub-detector. Figure 2(c) highlights the fact that the event server can distribute events or sub-events.

By virtue of this splitting, all these tasks will already happen in parallel and asynchronously in the transformed system. Note also that deploying these actors on the same node or not is a matter of a simple configuration. One could for instance easily promote the event generator to a real event server that could serve all simulation jobs on the computing grid.

By default, communication between devices happens through the network, by using the TCP/IP stack. However, when actors are deployed on the same node, the shared memory system of FairMQ can be used [14]. For demonstration purposes, we have developed a variant where the hits are put into a shared memory buffer and consumed directly from there, as outlined in figure 2(b). Such a system has analogies to a typical data acquisition scheme.

2.2.2 Parallelism

In a next step, we introduce event and sub-event *parallelism* into the simulation by scaling up the number of simulation workers. This means that multiple *independent* worker devices are instantiated at the same time as indicated in figure 3(a). Each of those actors asks the event server for tasks to process, where a task is either a full event or a sub-event. Hence the system is able to process multiple events in parallel *or* to collaborate on the simulation of a single event concurrently.

In case of full events, this setup achieves the same functionality offered by Geant4-MT (in its current version) but goes beyond it when sub-events are used. Moreover, since the

https://doi.org/10.1051/epjconf/201921402029

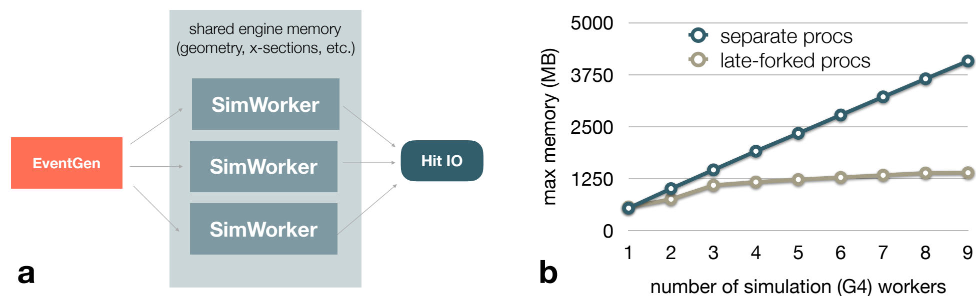

Figure 3. Simulation parallelism can be introduced by spawning multiple worker actors as shown in (a). If the process forking happens after the initial setup stage, the read-only memory such as geometry and x-sections are shared between all workers. (b) Memory comparison of the late-forking approach versus plainly instantiating multiple Geant4 processes. The scaling of the forked approach is very flat and substantially better compared to the naive process instantiations approach.

simulation worker can be any VMC engine, the parallelism is available even with Geant3 or Fluka, providing a new perspective to these engines.

The internal data structure of single simulation process can occupy a considerable amount of memory, for instance due to the detector geometry, magnetic field tables, and tables for physics cross-sections. In our case, a single Geant4 process needs around 500 MB of initial memory, not yet counting data produced during simulation. Carelessly spawning multiple such processes would hence lead to a significant memory usage, which is often taken as one argument in favour of multithreading. However, by using a particular strategy based on late forking, this can easily be avoided. In this approach, a single VMC engine is fully initialized at the beginning and only thereafter the other workers are created by repeatedly forking off from the first one, targeting a use of the copy-on-write mechanism of the operating system. Typically the actual FairMQ actor instance and its properties are only created and set thereafter.

Figure 3(b), demonstrates that this works very well. Using the late-forking approach, the total memory consumption, including simulation products for a few simple pp events, scales very flatly as the number of simulation engines increases. Each engine in addition barely adds on the order of a few MB and memory is not a limiting factor to deploying the system on large nodes on HPC hardware. Note that due to sub-event capabilities, also the memory-spikes seen earlier will no longer occur.

2.2.3 Timing Evaluation

Other than evaluating the scaling of overal memory, we looked into how the messaging system affects the wall-time needed to process collisions. In the first instance, it is interesting to compare the time needed to process a few events in the original simulation executable with the time needed to do the same in the actor-based system of figure 2(a). In case of simple 14 TeV pp collisions, we see that even a few percent ($\sim$ 5%) of wall-time is instantly gained. This is solely due to the introduction of the asynchronous and concurrent nature of event generation, simulation and IO and that we see that the gain is (in our case) larger than the overhead from transporting data between processes over the network.

Next, in order to determine the parallel (strong-scaling) speedup, the detector response to a few large realistic heavy-ion collision was simulated with sub-event splitting enabled (500

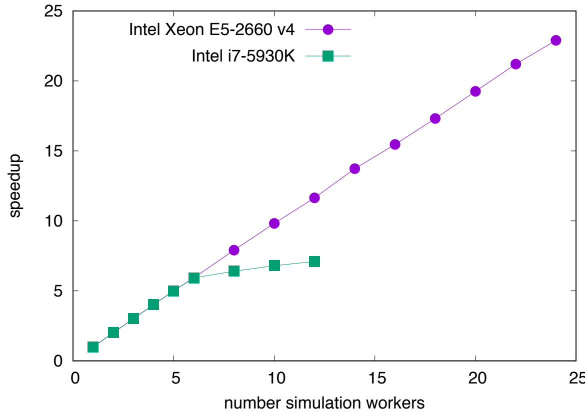

Figure 4. Strong-scaling speedup to simulate 5 large heavy-ion collisions as a function of the number of simulation workers. We see perfect scaling up to the number of physical cores on a simple desktop machine (Intel i7) as well as on a multi-node server (Xeon E5). Note data points for Intel i7 include the hyperthreading region (flattening of curve) which was not available on the Xeon machine.

primaries was chosen as the sub-event size). This study was performed both on a desktop machine with 6 physical cores as well as on a 2-node server with 24 cpu cores in total. The speedup, as shown in figure 4, was determined by comparing the wall-time of N workers with the time needed for just one worker. In essence, the numbers show that we have almost perfect strong-scaling up to 24 CPU cores on the server node (The serial coefficient in Amdahl's law is smaller than 0.5%). Currently, for yet more workers we have indications that the IO process may become a bottleneck. In such a case, one needs to refine the IO components to be more hierarchic or to scale them out to a different node. The scaling study was currently limited to one node. We plan to extend the study across multiple nodes in future.

In summary, we obtained a reduction of the time needed to treat a single large ion-ion collision to a few minutes compared to an hour before. Hence, we will be able to use opportunistic time windows on HPC machines which provides a considerably new quality to the simulation project.

It is worth stressing that arriving at such a fully scalable and performant system did not require touching the internals of these applications nor complicated multi-threaded development. All that was needed was interfacing existing code from within FairMQ device instances and defining some communication connections.

2.2.4 Configuration Distribution

In a distributed system such as described here, there is a natural question on how to ensure consistency of the configuration used across all actors. Currently, this is implemented via a simple request-reply distribution of the simulation parameters. For instance, at startup time, each worker asks the configuration for the current production from the event server and sets itself up accordingly. This configuration includes for instance geometry parameters, type of simulation engine, process cuts, and other values. Apart from that the system can easily be extended to query condition-database servers to fetch condition objects (such as alignment).

3 Additional Benefits and Further Ideas

The architecture described here offers a few interesting secondary benefits and possible extensions. The first one is related to the elastic nature of the system. In fact, due to the request-reply communication pattern, simulation workers can be attached and detached to the topology dynamically. This allows for things like volunteer computing, where for example some laptop with free CPU cycles can join – for a short time – a simulation production running somewhere else.

Another interesting feature offered by the system at no cost is the possibility to mix (heterogenous) simulation engines working on the same collision. In fact, different VMC engines can easily be created in the topology due to the multi-processing nature. Each of them could receive particular primary tracks, selected according to direction, type, energy and other filters. Like this, one could leverage specific strengths of each VMC simulation engine or gain speed in regions where less detailed simulation is required.

Thinking further, the system may provide a step towards a parallel system having both full and fast simulation options. We are currently exploring how to add fast simulation components to the setup. Possible options are to have fast simulation kernels as separate FairMQ actors communicating with the rest of the components and/or to have fast-simulation dispatch as an extension inside the VMC framework.

A third point, worth mentioning, is the interaction or integration with the new ALICE-O2 Data Processing Framework [15], which will also be based on FairMQ/ALFA. Here we will have the possibility to directly forward simulated hits to the data processing chain, in order to do digitization and reconstruction asynchronously to the detector simulation.

4 Discussion of Related Work

We would like to turn to a short discussion of other developments that take place in parallel to this work or are planned for the future.

First of all, the current Geant4-MT development plan foresees support for sub-event parallelism, at least within one compute node, as requested by multiple users. This will bring similar benefits as described here (in particular speeding up treatment for a single event) to native Geant4 users and could simplify out our scheme a bit (for instance for bookkeeping). In this regard the recent progress in enabling the Geant4-MT mode in the VMC layer and FairRoot is a fundamental requirement [16]. We plan to evaluate the combination of the FairMQ actor model and G4-multithreaded mode and support this if beneficial.

The idea of sub-event parallelism is not completely new. It has been used in various forms for instance by the ATLAS Athena framework (see, e.g., [17]. The extension of sub-event parallelism to track-level parallelism, in order to benefit from SIMD vectorization, is being studied in the GeantV R&D activity [18].

5 Summary

In summary, this proceedings presented a system to perform detector simulation based on message passing and independent processing actors. The system achieves sub-event parallelization with strong scaling for any VMC simulation engine. There is no decisive memory disadvantage in comparison to multi-threaded solutions. The system can easily be scaled across multiple compute nodes and can be deployed on HPC architectures. A key advantage is the low development effort with which this was achieved using FairMQ.

We would like to thank the FairMQ developers, notably Dennis Klein, Alexey Rybalchenko, and Mohammad Al-Turany for very stimulating technical discussions and support.

References

[1] S. Agostinelli, et al, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)

[2] I. Hrivnacova, D. Adamova, V. Berejnoi, R. Brun, F. Carminati, A. Fasso, E. Futo, A. Gheata, I.G. Caballero, A. Morsch, CoRR **cs.SE/0306005** (2003)

[3] R. Brun, F. Bruyant, M. Maire, A.C. McPherson, P. Zanarini, *GEANT 3: user's guide Geant 3.10, Geant 3.11; rev. version* (CERN, Geneva, 1987), `https://cds.cern.ch/record/1119728`

[4] G. Battistoni, T. Boehlen, F. Cerutti, P.W. Chin, L.S. Esposito, A. Fassa, A. Ferrari, A. Lechner, A. Empl, A. Mairani et al., Annals of Nuclear Energy **82**, 10 (2015), joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms

[5] R. Brun, F. Rademakers, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **389**, 81 (1997), new Computing Techniques in Physics Research V

[6] M. Al-Turany, D. Bertini, R. Karabowicz, D. Kresan, P. Malzacher, T. Stockmanns, F. Uhlig, Journal of Physics: Conference Series **396**, 022001 (2012)

[7] M. Al-Turany, P. Buncic, P. Hristov, T. Kollegger, C. Kouzinopoulos, A. Lebedev, V. Lindenstruth, A. Manafov, M. Richter, A. Rybalchenko et al., Journal of Physics: Conference Series **664**, 072001 (2015)

[8] *FairMQ*, `https://github.com/FairRootGroup/FairMQ`

[9] *DDS*, `https://github.com/FairRootGroup/DDS`

[10] P. Buncic, M. Krzewicki, P. Vande Vyvre, Tech. Rep. CERN-LHCC-2015-006. ALICE-TDR-019 (2015), `https://cds.cern.ch/record/2011297`

[11] S. Acharya et al. (ALICE) (2018), `arXiv/1812.08036`

[12] *zmq*, `zeromq.org`

[13] *nanomsg*, `https://nanomsg.org/`

[14] A. Rybalchenko, M. Al-Turany, D. Klein, T. Kolleger, presented at CHEP2018 (Sofia) - same proceedings

[15] G. Eulisse, P. Konopka, M. Krzewicki, D. Rohr, S. Wenzel, presented at CHEP2018 (Sofia) - same proceedings

[16] I. Hrivnacova, presented at CHEP2018 (Sofia) - same proceedings

[17] G.A. Stewart, J. Baines, T. Bold, P. Calafiura, A. Dotti, S.A. Farrell, C. Leggett, D. Malon, E. Ritsch, S. Snyder et al., Journal of Physics: Conference Series **762**, 012024 (2016)

[18] G. Amadio, A. Ananya, J. Apostolakis, A. Arora, M. Bandieramonte, A. Bhattacharyya, C. Bianchini, R. Brun, P. Canal, F. Carminati et al., Journal of Physics: Conference Series **762**, 012019 (2016)