EPJ Web of Conferences **214**, 02007 (2019) https://doi.org/10.1051/epjconf/201921402007
CHEP 2018

Recent progress with the top to bottom approach to vectorization in GeantV

Guilherme Amadio[1], Ananya, *John* Apostolakis[1], *Marilena* Bandieramonte[1,2], *Shiba* Behera[3], *Abhijit* Bhattacharyya[3], *René* Brun[1], *Philippe* Canal[4], *Federico* Carminati[1], *Gabriele* Cosmo[1], *Vitaliy* Drohan, *Daniel* Elvira[4], *Krzysztof* Genser[4], *Andrei* Gheata[1,*], *Mihaela* Gheata[1,5], *Ilias* Goulas[1], *Farah* Hariri[1], *Vladimir* Ivanchenko[1,6], *Przemislaw* Karpinski, *Gulrukh* Khattak[1], *Dmitri* Konstantinov[1,8], *Harphool* Kumawat[3], *Guilherme* Lima[4], *Jesús* Martínez Castro[7], *Patricia* Mendez[1], *Aldo* Miranda Aguillar[7], *Katalin* Nikolics[1], *Mihaly* Novak[1], *Elena* Orlova, *Kevin* Pedro[4], *Witold* Pokorski[1], *Alberto* Ribon[1], *Dmitry* Savin, *Ryan* Schmitz, *Raman* Sehgal[3], *Oksana* Shadura[1], *Shruti* Sharan, *Sofia* Vallecorsa[1], *Sandro* Wenzel[1], and *Soon* Yung Jun[4]

[1]European Organization for Nuclear Research (CERN), Switzerland
[2]University of Pittsburgh, 4200 Fifth Avenue, Pittsburgh, PA 15260, USA
[3]Bhabha Atomic Research Centre (BARC), India
[4]Fermi National Accelerator Laboratory (FNAL), USA
[5]Institute of Space Science (ISS), Romania
[6]Tomsk State University, Tomsk, Russia
[7]Centro de Investigación en Computación (CIC-IPN), Mexico
[8]NRC Kurchatov Institute (IHEP) Protvino, Russia

Abstract. SIMD acceleration can potentially boost by factors the application throughput. Achieving efficient SIMD vectorization for scalar code with complex data flow and branching logic, goes however way beyond breaking some loop dependencies and relying on the compiler. Since the refactoring effort scales with the number of lines of code, it is important to understand what kind of performance gains can be expected in such complex cases. We started to investigate a couple of years ago a top to bottom vectorization approach to particle transport simulation. Percolating vector data to algorithms was mandatory since not all the components can internally vectorize. Vectorizing low-level algorithms is certainly necessary, but not sufficient to achieve relevant SIMD gains. In addition, the overheads for maintaining the concurrent vector data flow and copy data have to be minimized. In the context of a vectorization R&D for simulation we developed a framework to allow different categories of scalar and vectorized components to co-exist, dealing with data flow management and real-time heuristic optimizations. The paper describes our approach on coordinating SIMD vectorization at framework level, making a detailed quantitative analysis of the SIMD gain versus overheads, with a breakdown by components in terms of geometry, physics and magnetic field propagation. We also present the more general context of this R&D work and goals for 2018.

[*]e-mail: andrei.gheata@cern.ch

1 Introduction

Due to the physical constraints preventing frequency scaling, parallel computing has become the dominant paradigm in modern computer architectures. Throughput-increasing parallelism techniques are particularly relevant for scaling up the computing performance with the transistor density. Most modern processors are superscalar, being able to deliver more than one instruction per clock (IPC). Instruction level parallelism is based on scheduling multi-stage instruction pipelines to multiple execution units in the processor. As long as instructions have no data dependencies, they can be executed out of order by the different execution units. Several speculative techniques are used for minimizing the idle time of the existing execution units, such as: branch prediction, speculative prefetching and execution, or cache hierarchy.

A different approach is to leverage the natural data parallelism of a computational problem by mapping multiple data to similar operations executed in parallel. The most successful implementations of this concept are SIMD (Single Instruction Multiple Data), and SIMT (Single Instruction Multiple Threads). In SIMD, elements of short vectors are processed in parallel using special vector registers and an extended instruction set, while in SIMT, instructions of several threads run in parallel. Both approaches are broadcasting the same instruction to different execution units, the main differences coming from the different degrees of flexibility versus efficiency. Arguably, the SIMT model, which is very popular for GPU's, is more flexible and well adapted for massive data-parallel problems, while the SIMD approach is more difficult to program but more efficient in case of more confined data loops.

While the benefit of SIMD and/or SIMT was demonstrated for applications featuring massive data parallelism, such as linear algebra or graphics, we are trying to develop vectorization techniques that can preserve these benefits in case of code with large complexity and branching. Particle transport simulation represent a very demanding HEP application having many non-SIMD friendly features like: sparse memory access over large data structures, deep embedded conditionals branching to a high number of algorithms per data unit (track). While exploring SIMD/SIMT optimization techniques in the context of GeantV R&D [1], many of our results can be extrapolated to other similar HEP workflows, including some areas of reconstruction or analysis.

This paper describes the techniques used in GeantV for enhancing the data parallelism by transforming typical single-track non-vectorizable algorithms into multi-track SIMD-aware ones. The following sections are presenting a critical view on the gains versus overheads for such an approach, hinting that there is a complexity threshold for the transformation to be beneficial, but also presenting some ways to push this threshold to higher values. In a final section we try to illustrate this on concrete examples that use specific measurements.

2 Vectorizing on track data

From a data driven perspective, particle transport simulation can be seen as a sequence of algorithms operating on a single track state. A particle would enter a given material, travel in fields and perform a stochastic physics process in a sequence of complex calculations altering its state and possibly producing new secondary particles. This can be seen as a processing pipeline that takes one track as input and outputs a variable number of tracks, as in Figure 1. The complete workflow in a standard Geant4 [2] application would stack all new tracks and then execute such pipeline (called stepping loop) repeatedly for each of them until the stack is exhausted. In reality, the picture is more complex due to the presence of conditionals that may branch the follow-up algorithms, but in all cases such scalar workflow limits the opportunities to use the natural track-level parallelism.

GeantV transforms the scalar workflow into a vector one. Instead of handling one track at a time, algorithms can operate on collections of tracks, called *baskets* in the GeantV jargon. Once a basket is injected in the algorithm, the vectorization problem is reduced to transforming all scalar operations on track data into vector operations on basket data. To generate efficient SIMD instructions, the basket data needs to be transformed from an array of track structures (AOS) to a structure of arrays of track data (SOA), to be fitted directly into vector registers. This copying is only necessary for the part of the track data needed by the algorithm. The vectorization technology we chose is based on VecCore [3], an abstraction header library on top of a number of vectorization back-ends.

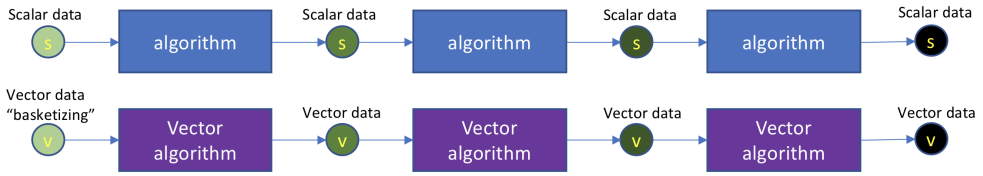

Figure 1. Transformation of a pipeline scalar workflow into a vector one. This implies propagating data containers (baskets) through the pipeline, but also transforming scalar operations inside the algorithms into vector ones.

We need to emphasize that most of the work required for migrating an algorithm from a pure scalar version to the basketized one is concentrated on refactoring the algorithm to work with vector types. The process allows to preserve the scalar version and even in most cases coalesce it with the vector one by templating the implementations on the data type. The GeantV scheduler actually makes use of both scalar and vector versions to optimize its workflow, as described in a previous paper [4].

Doing a back-of-the-envelope calculation of the performance of a single algorithm in the context of a basketized data flow can be seen in the following way: We assume T_s to be the time taken by the scalar version of the algorithm to process a given amount of tracks, ignoring any dependency on their state. The corresponding run time for the vectorized version of the same algorithm will be T_s/S_i. The SIMD intrinsic speedup S_i can be measured in a benchmark test for the given algorithm, outside of the actual GeantV workflow. This will depend only on intrinsic algorithm features such as complexity or amount of memory versus floating point operations. There is a price to pay for performing vector operations due to the need to copy scalar track information into structures of arrays, then scatter them back after getting the results. This overhead T_o depends on the amount of track state needed by the algorithm, but also on the data sparsity which can generate data cache misses. Note that after the SOA gather operations both data and instructions become more local so the data restructuring may be beneficial. A further analysis of this is done in the next section.

In a pipeline workflow where all tracks will be processed by all algorithms, we can reach almost full efficiency to execute a basket workflow, having to treat as scalar only the tails of the data passing through. In reality, the simulation workflow is more complex (as in Figure 2) and many algorithms are selected only for subsets of all tracks. This would still be efficient if there was an infinite source of tracks feeding the simulation stepping procedure. Baskets would still eventually get filled even for the very "unpopular" algorithms at a smaller rate than others. In reality there are memory constraints limiting the number of events/tracks in flight, so a starvation point will be reached when no tracks from new events can be pushed into the stepping loop until completing at least one of the events already being transported. Since the remaining tracks at this point are held by incomplete baskets, we can only refill the stepping loop by flushing those baskets. This will allow popular baskets to be refilled, but the less pop-

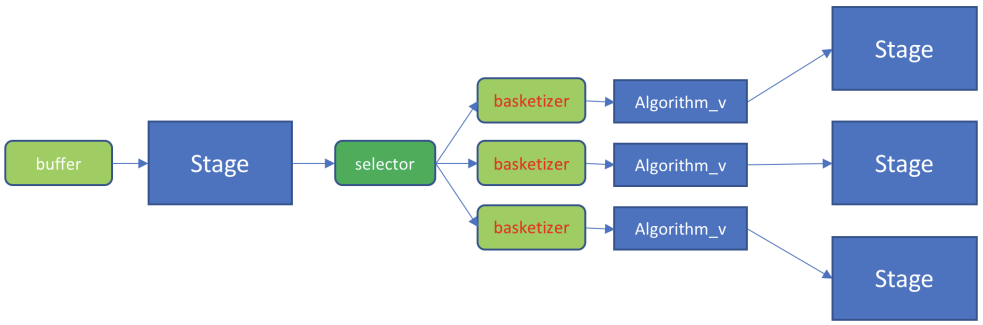

Figure 2. Generic workflow for GeantV stepping. The stepping procedure is split in stages containing one or more algorithms specialized for specific tasks like: geometry querying, physics model sampling or field propagation. A vector of tracks is pushed through the stage buffers, dispatching each track to the appropriate algorithm. A basketizer may be used to hand-over vectors instead of single tracks.

ular ones will have to be regularly flushed while executing the corresponding algorithms in scalar mode. This mechanism decreases the basketization efficiency b_e (percentage of tracks processed in basket mode) and depends on many factors such as: algorithm popularity, size of the event buffer or event type, but also physics settings or geometry setup. For example, a calorimeter shower will self-refresh populations in most baskets after each stepping loop due to the high rate of secondary generation, which is not the case within a tracker. Taking the above into account, we can express the time taken by the basketized and vectorized version of the given algorithm within the given workflow as:

$$T_v = T_o + (1 - b_e)T_s + b_e\frac{T_s}{S_i} \tag{1}$$

From equation 1 we can easily derive the observed algorithm speedup $S = T_s/T_v$ when plugged in the complete simulation workflow, taking $b_o = T_o/T_s$ as relative basketizing overhead:

$$S = \frac{S_i}{b_e + S_i(1 - b_e + b_o)} \tag{2}$$

Note, that for a pipeline workflow $b_e \approx 1$ with negligible basketizing overhead $b_o \approx 0$ we benefit from the full intrinsic algorithm speed-up, while in case of a complex workflow with an algorithm basketized only 50% ($b_e = 0.5$) we get a maximum speedup of 60% even if the SIMD algorithm is $4x$ faster than its scalar version. We can extract from equation 2 the minimum intrinsic SIMD speedup of the algorithm to have a visible benefit $S > 1$ from basketized vectorization:

$$S_i > \frac{1}{1 - \frac{b_o}{b_e}} \qquad \text{in case} \quad b_e > b_o \tag{3}$$

Note that equation 3 stands only if $b_e > b_o$, otherwise no speed-up is possible. Assuming that we can measure these numbers, equation 3 gives a quick recipe for deciding if an algorithm is worth basketizing. If the ratio overhead over efficiency is too large even an ideally vectorized algorithm will not benefit from basketizing. It hints to few rules of thumb: to have a small overhead, the number of copy operations should be smaller than the number of vectorized operations, while the algorithm needs to be rather popular in the execution flow.

3 Efficiency versus overhead

To measure basketizing efficiency per algorithm, it is enough to count how many tracks follow the basketized path compared to how many tracks had to be flushed from baskets and processed by the scalar version. This can be easily done using counters inserted in the workflow. The more complex the workflow, the more tracks have to be transported concurrently to fill baskets of different categories. This hints to the use of large event buffers and implicitly more memory to enhance performance. Table 1 gives a summary of the basketizing efficiency for several different algorithms in GeantV in case of a complex example using the CMS geometry and EM physics, for the most important simulation stages: geometry (computation of distances and particle relocation), propagation in field, multiple scattering (MSC) and final state generation (PostStep) for all physics processes involved. In the current implementation we observe overheads larger than efficiencies for basketizing geometry and final state generation, which according to equation 3 cannot produce speed-up for these stages. An interesting observation is that basketizing efficiency for a given algorithm implicitly depends on the "health" of the data flow. The starvation point tends to occur earlier in a fully basketized configuration and will reduce the overall efficiency. After flushing partial baskets and triggering scalar stepping, the tracks may not generate enough new work and the starvation regime will generate a complete scalar transport. This is an undesired effect that is unavoidable when having a too small number of tracks in flight. It makes the bulk of the events being transported in vector mode, while the tails in scalar mode.

Stage	Geometry	Field	Multiple scattering (MSC)	Phys. models final state sampling
b_e	0.20	0.75	0.43	0.20
b_o	0.63	-0.02	0.01	0.42

Table 1. Basketizing efficiency and overhead for different simulation stages for 100 GeV electrons in the CMS simulation benchmark

The basketizing overhead is difficult to measure based on scalar versus basketized profiling information, due to better caching in basket mode. GeantV allows measuring this in a special mode: a given basketizer can be activated, but the basket will be processed still in scalar mode by looping over tracks (scalar dispatch). So far, we performed only integral measurements in GeantV, summing the overheads of all basketizers in a given stage.

The current measurements show an important overhead for basketizing based on geometry and physics models final state sampling. This is due to the need to copy rather large fraction of the track state in SOA form, but also due to the limited amount of floating point operations involved in some of these models. The relative use of different physics models is very much dependent on the simulation type: beam energy, particle type and detector setup. Even along different phases during the event propagation, particles will change model affinity as the initial beam energy is deposited throughout the detector. In case of geometry, we know that currently the algorithm of reducing the number of basketizable volumes is not very efficient. We want to emphasize that the basket efficiencies and overheads are not absolute performance markers but they represent a snapshot of performance for a given set of simulation conditions.

As the particle flow distribution to algorithms in different stages changes during simulation, basketizing efficiency may evolve and make the apparent speed-up for a given algorithm less than 1. To deal with this, GeantV monitors the activity for certain basketizers and can dynamically switch on/off basketizing to cut off some unnecessary overheads. This can happen for example in geometry, where central tracker volumes are popular at the beginning of

event propagation, but much less popular during shower development in calorimeters. In the cases where GeantV uses large event buffers mixing many tracks from different events, different event stages coexist at a given moment so the configuration of active basketizers reaches asymptotically some stability.

4 Benchmarks and ongoing optimizations

GeantV evolved from the alpha tag late 2017 to a version embedding vectorized versions for all electromagnetic physics models and magnetic field propagation. We are currently in the process of understanding the model behavior and performance, to both reduce the remaining scalar bottlenecks and introduce heuristic optimizations, such as adapting the thresholds for turning on/off basketizing per model.

The current benchmarks used for tuning this behavior are examples with very different complexity, starting from a simple box with configurable dimensions and material, a simplified sampling calorimeter built up from boxes with arbitrary number of layers, to full LHC detectors such as CMS or LHCb. We are using a custom GeantV physics list including standard EM processes for e^- (ionization, bremsstrahlung, Coulomb scattering), e^+ (ionization, bremsstrahlung, Coulomb scattering) and γ (photoelectric, absorption, Compton scattering, e^+e^- pair production). Note that energy loss fluctuation and e^+ annihilation are not included in the physics list. Each of these examples have a counterpart implemented as a Geant4 application, using a physics list containing the same set of processes and model-level settings. We are scoring inclusive observables, such as track length or energy deposited per primary track, particle multiplicities or number of steps, allowing to perform MC validation at below 1 per mil level against the corresponding Geant4 application.

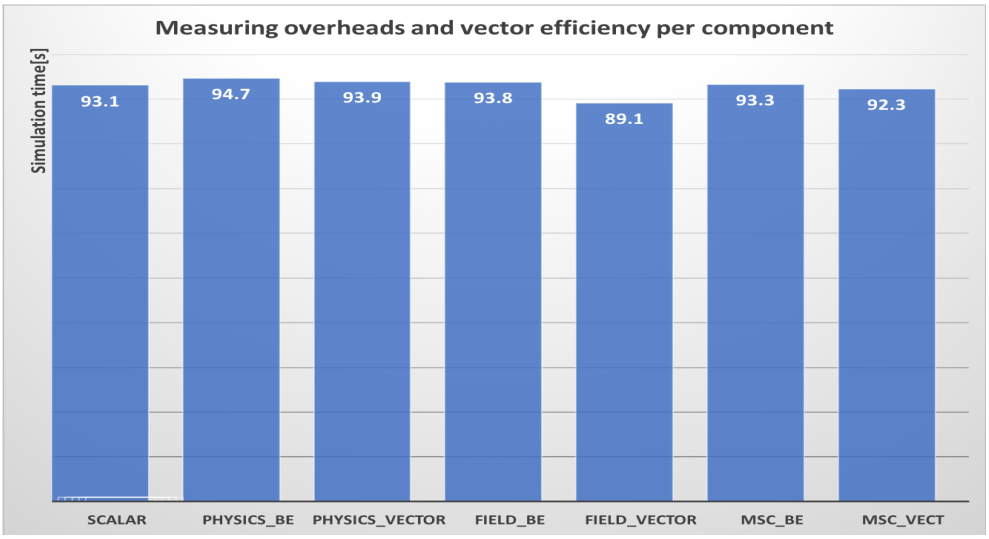

Figure 3. Simulation time measurements for different GeantV configurations for the CMS benchmark. The scalar mode provides the reference. Basketization can be switched on for a single simulation stage (here reported only physics final state sampling, field propagation and multiple scattering - MSC). The baskets can be dispatched to either the scalar version (BE) or the vector version of the stage algorithms, allowing to measure the basketization overheads and the observed speedup. The basketization efficiency is reported for each basketized stage at the end of the run.

We benchmark several different configurations of GeantV to measure the basketizing efficiency and overheads per module. For example, Figure 3 shows several such simulation time

measurements. As general feature, in the current stage of the implementation we observe that only field propagation and multiple scattering basketization bring visible benefit of up to 15% of the total run time, while geometry and post stepping physics actions have both a small deficit compared to their scalar version.

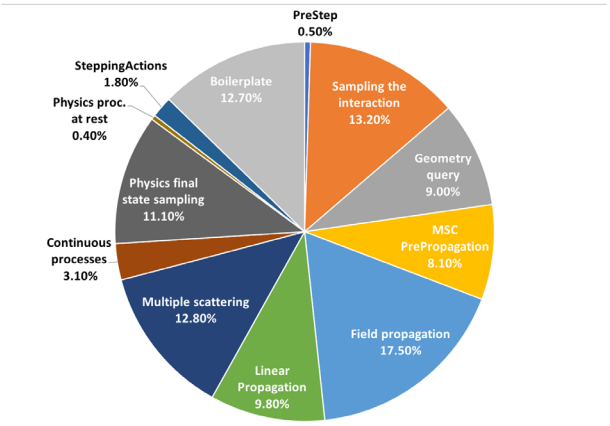

Figure 4. Profile for the CMS simulation benchmark extracted using *gperftools* in scalar mode. The relative fraction of the total time taken by a given simulation stage can be used to estimate the potential speed-up by vectorizing the stage. The relative fractions modify when using the basketized vector version of the code.

In Figure 4 we present profiling information extracted using *gperftools* [5] for the CMS example. The pie chart shows the fractions of the total run time taken by the different simulation stages. Correlating this with the bottom-up view of hot-spots, we extract relevant information about existing scalar bottlenecks. From a preliminary analysis, we identified two important optimization areas for geometry and physics respectively. Geometry calculations for relocating tracks after crossing volume boundaries are dominated by scalar matrix multiplication, adding up to 5% of the total simulation time. Physics basketization is currently very inefficient since a large fraction of the time is spent in AOS to SOA conversion. These areas are critical for the overall efficiency of the basketizing method, being currently under investigation.

An important benchmark for the GeantV prototype is the single thread CPU performance compared to Geant4 (Figure 5). This comparison and a detailed performance analysis of both GeantV and Geant4 applications can reveal where the gains are coming from, which are the weak points and possible optimization areas. At this stage we observe a speedup of 60-65% when activating basketizing for magnetic field and multiple scattering process. There are several potential geometry and basketizing optimizations being addressed, targeting a reduction of 15-20% of the run time.

5 Conclusions

GeantV investigates a top to bottom approach to vectorizing the particle transport simulation. Basketizing particles for the next processing stage allows vectorizing on a much larger class of algorithms than the single particle mode. The basketizing dynamics depends on the complexity of the workflow and on state parameters, such as number of tracks in flight, particle production budget, or percent to completion for a given event. There are workflow constraints limiting the maximum number of events being transported concurrently, that impose flushing partially filled baskets and dealing with scalar spills that reduce the overall efficiency. This affects mostly the “unpopular” algorithms and requires dynamic basketize activation/deactivation.

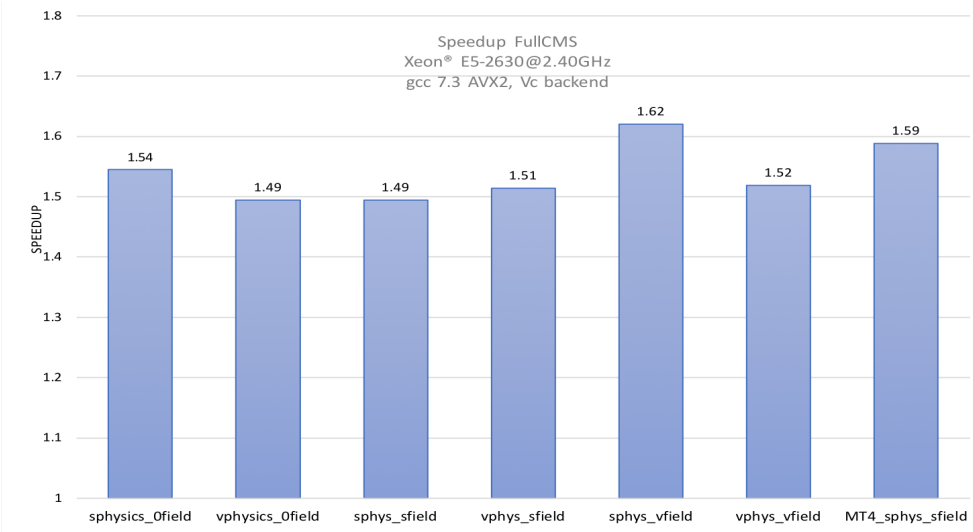

Figure 5. Speed-up for the CMS example compared to the Geant4 equivalent for different GeantV configurations: magnetic field on/off, scalar/basketized physics, scalar/basketized field, multithreading on (4 threads).

In the current version, the GeantV prototype provides vectorized implementations for a large fraction of the algorithms involved in the transportation procedure, including geometry, field propagation and physics simulation. We are in a phase where the basketizing dynamics is well understood and we are performing detailed performance analysis to optimize specific components and reduce the existing scalar bottlenecks.

In the current phase of the GeantV R&D, most of the components needed for a complete demonstrator of a vectorized EM shower simulation engine have been developed. After the ongoing optimization procedure we plan tagging a beta version and making a realistic projection of the performance figures within reach in a short time scale.

References

[1] G. Amadio et al., *J. Phys: Conf. Ser.* **664**(2015) 072006
[2] J. Allison et al., *NIM* **A 835** (2016), 186-225
[3] G. Amadio, P. Canal, D. Piparo and S. Wenzel, *J. Phys.: Conf. Ser.* **1085** (2018) 032034
[4] G. Amadio et al., *J. Phys.: Conf. Ser.* **1085** (2018) 032037
[5] C. Silverstein and D. Chappelle, *Great Performance Tools* (2012)
http://code.google.com/p/gperftools/