Adaptive track scheduling to optimize concurrency and vectorization in GeantV

J Apostolakis[1], M Bandieramonte[2], G Bitzes[3], R Brun[1], P Canal[4], F Carminati[1], J C De Fine Licht[5], L Duhem[6], V D Elvira[4], A Gheata[1], S Y Jun[4], G Lima[4], M Novak[1], R Sehgal[7], O Shadura[8], S Wenzel[1]

[1] European Organization for Nuclear Research (CERN) - Geneva, Switzerland
[2] University of Catania and INAF (IT)
[3] University of Athens (GR)
[4] Fermi National Accelerator Laboratory (US)
[5] University of Copenhagen (DK)
[6] Intel Corporation
[7] Bhabha Atomic Research Center (IN)
[8] National Technical University of Ukraine, "Kyiv" Politechnic Institute

Andrei.Gheata@cern.ch

Abstract. The *GeantV* project is focused on the R&D of new particle transport techniques to maximize parallelism on multiple levels, profiting from the use of both *SIMD* instructions and co-processors for the CPU-intensive calculations specific to this type of applications. In our approach, vectors of tracks belonging to multiple events and matching different locality criteria must be gathered and dispatched to algorithms having vector signatures. While the transport propagates tracks and changes their individual states, data locality becomes harder to maintain. The scheduling policy has to be changed to maintain efficient vectors while keeping an optimal level of concurrency. The model has complex dynamics requiring tuning the thresholds to switch between the normal regime and special modes, i.e. prioritizing events to allow flushing memory, adding new events in the transport pipeline to boost locality, dynamically adjusting the particle vector size or switching between vector to single track mode when vectorization causes only overhead. This work requires a comprehensive study for optimizing these parameters to make the behaviour of the scheduler self-adapting, presenting here its initial results.

1. GeantV project in the simulation context of LHC

One of the striking facts for the LHC computing was that more than half of the available CPU resources of the experiments were used for producing simulated samples. Assuming a linear dependency of simulation needs with the acquired data, the implications from the perspective of the LHC upgrade schedule become sizeable as far as computing resources are concerned. As the integrated luminosity will increase by factors in the near future and by orders of magnitude for the high-luminosity LHC era, we are looking at a tremendous CPU power to cover future simulations.

The "why" part of the observation above is very important for understanding what can be done in the future besides just scaling up the resources, but also using them more efficiently. This is what

triggered the investigation and prototyping work that started in 2011 and which is the base for the current R&D behind the *GeantV* project [1]. Looking at the main reasons for the low number of instructions per cycle of about 0.8 in particle transport simulations and HEP code in general, we can identify a large range of sources, starting with abusing object-oriented design in data access patterns and ending with poor data and code locality caused by the sequential nature of the classical particle transport approach.

This simulation optimization initiative has grown during the last years into a project looking at many different performance dimensions, which can potentially speed up simulation by important factors on the architectures available today. The main idea behind the first version of the *GeantV* prototype was changing the basic work unit for transport simulation from a single particle to a vector of particles fulfilling geometry locality criteria. This was motivated by the observation that most of the simulation time is spent in only 5-10% of all logical volumes of a HEP experiment, and in addition geometry locality means material locality, which partially implies physics locality. Adding multithreaded concurrency on top of this model allowed building the initial version of the prototype which sets the basis for exploring new features like: SIMD vectorization, fine grained parallelism, transparent usage of resources (CPU/GPU) or re-design of the code in terms of specialization and re-usability.

The project is evolving in three main directions which are complementary enough to allow for independent development and which share core data structures and features into a single simulation engine, which currently is still in the prototyping phase.

- The geometry modeling development aims to re-design the main features of the standard geometry modelers such as the *Geant4* [2] and *ROOT* [3] ones using techniques like template specialization and generic programming, supporting vectorization at library level and generating optimized code for both CPU and GPU. This is evolving into the next generation geometry tool, triggered by the *GeantV* project but usable well beyond it. This geometry development, known as *VecGeom* [4], recently started to be merged with the *USolids* library [5], which was already crossbreeding the features of *ROOT* and *Geant4* shape primitives.
- The physics development aims to refactor the *Geant4* physics engine and algorithms into vectorized kernels usable on both CPU and GPU, using the same specialization techniques whenever possible and allowing for "fast" features like generalized tabulated cross sections or inclusion of fast simulation techniques.
- The kernel and scheduler development aims to globally optimize the flow of the simulation application, managing concurrency and deploying vectors of tracks matching appropriate locality criteria to both physics and geometry algorithms. The major challenge of the scheduler is to keep a positive balance between the gains from locality and vectorization and the overheads coming from the extra data management required by the vector approach. The following sections of the paper will focus more on these areas.

2. The data model: vectors of tracks

One of the important requirements allowing compilers to vectorize the code is the contiguity and alignment of the data to be processed. Any indirections using pointers or indexing represent major overheads or even blockers for vector architectures, besides the obvious negative impact on caching. Since the presence of a virtual function table for a type and the use of virtual function generally prevent vectorization, plain old data like types are to be favored in a vector approach, as well as the usage of structures of arrays (*SOA*) versus the classical arrays of structures (*AOS*). As rule of thumb, arranging the data in a way that avoids collecting at run-time from many places improves performance but is generally hard to achieve.

In *GeantV*, we implemented a custom *SOA* approach, which can automatically handle vectors of tracks expanding them into internally managed aligned arrays corresponding to each track data member. Memory allocation is performed at initialization time and expanded as needed. This structure

(called *GeantTrack_v*) can pass pointers to arrays of positions, momenta or other fields to any vectorized client. Due to the size of the track object (currently 192 bytes), an *SOA* approach can have important caching overheads, so we are also investigating a hybrid *AOSOA* (array of structures of arrays) to alleviate this effect.

2.1. Reshuffling overheads

Providing a vector container is far from being enough to percolate vectors down into non-trivially branched algorithms. Conditional code is harmful for both SIMD and GPU processing but in many cases can be partially avoided by using masking techniques [6] at the level of single algorithms. On the other hand, successive calls of algorithms that involve track selections or which are changing individual track states require container reshuffling or compacting operations.

A simple example is propagating a vector of particles in a single volume. Some of the particles will reach the volume boundary or the point where a physics process was sampled earlier than others, leaving "holes" in the container. Continuing the propagation in vector mode requires compacting the fields of the remaining tracks in contiguous memory and possibly moving the stopped tracks to a different vector container. Another common technique is to apply a vector operation to a subset of the tracks, which requires reshuffling those matching the criteria for the vector operation into a contiguous block at the beginning of the container.

All these operations involve costly memory copying, which is an overhead compared to a scalar approach. The challenge is to minimize these overheads as much as possible in order to keep them below the performance gains due to locality and vector processing.

3. Track scheduling model

The *GeantV* scheduler controls the simulation workflow. In the current implementation, a single thread controls the flow, monitoring the work queue and other critical parameters such as the vector size to trigger the appropriate corrective actions.

The prototype transports particles in a continuous flow, which is fed either from an external generator invoked on demand, or from the particles transported by a previous propagation step. The picture is very different from the standard sequential particle stack approach and mixes together tracks coming from different events. The *GeantV* scheduler design allows for an arbitrary number of filter modules to select and group together from the input flow all particles matching certain locality conditions, as shown in the top part of figure 1.

Locality means in our terminology the repeated execution of a sequence of instructions for all (or most of) the vector of particles, without any logical branching that could interrupt this sequence. The local code creates the premises for better caching and parallel execution, and is allowed to call inline functions or methods supporting a vector input signature on their turn.

The locality criteria are strongly dependent on the tasks to be executed within the track propagation (stepping) procedure, presented in the bottom of figure 1. For example, computing the distance to the next object boundary becomes local when called for a vector of particles contained by the same logical volume, allowing computing the distance in a vectorized fashion without any extra data gathering. The corresponding geometry filter will therefore group particles by this criterion. The physics sampler on the other hand requires particles of the same type and within the same energy range to become local. In the current version of the prototype, geometry locality is the only implemented filter, but we are extending the concept to other use cases as well.

An important future feature of *GeantV* will be the possibility to combine full and fast parametric simulation in the same session, in an arbitrary proportion. The use of fast simulation models will be triggered by user-defined filters, which would group particles by the criteria allowing these models to be applicable. Using the same filtering idea, additional resources such as general-purpose graphics processing units can be used transparently by grouping particles for which a given transport task would take profit from running on such devices.

Every locality filter fills its current vector of tracks up to a threshold, and after generates a *basket*, which is put in a common work queue. The baskets produced by a given filter are only used by the computing task they are filtered for. For example, a geometry filter generates baskets of tracks located in the same volume, which are processed by the *GeantV* vector propagator.

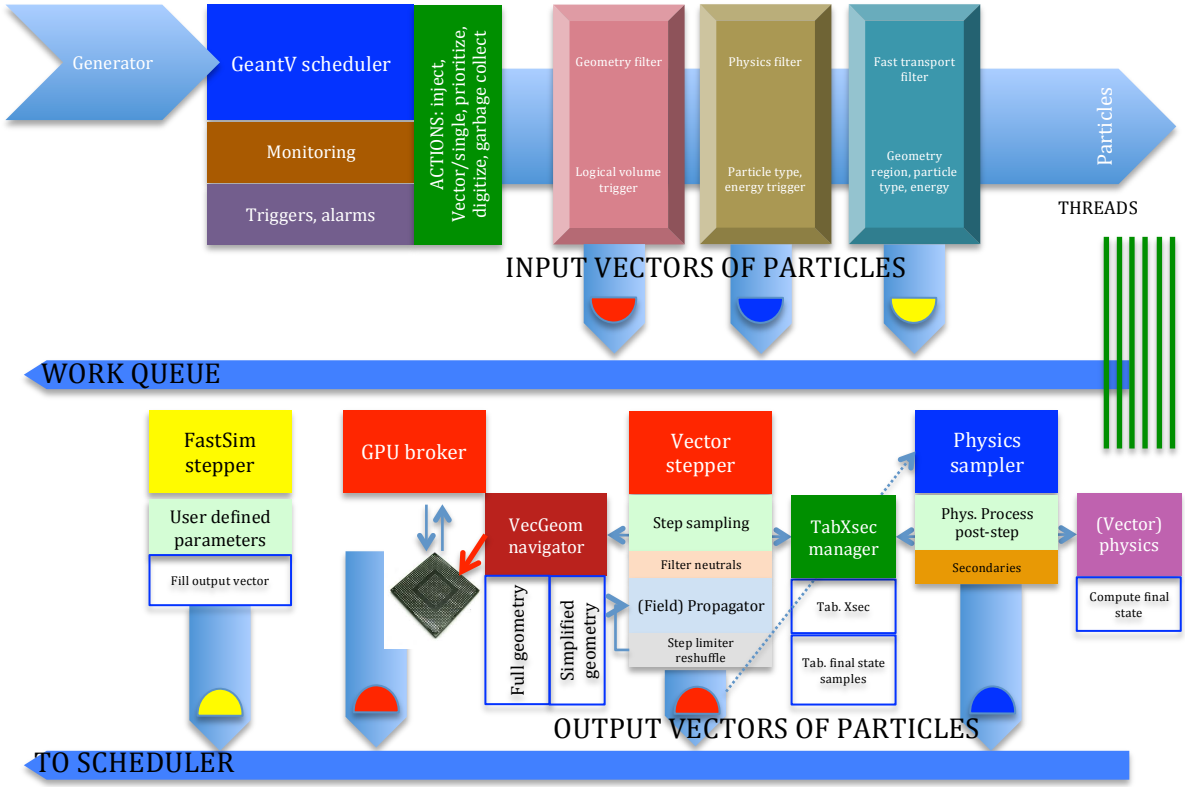

Figure 1. The *GeantV* scheduler aims to improve code locality and allow for SIMD vectorization in a concurrent environment by dispatching vectors of tracks to be transported. The design is based on a set of mutually exclusive filters grouping particles according certain locality criteria, that can be exploited in a vector manner by the different tasks executed in the propagation procedure, shown in the bottom part of the picture.

While the vector stepper operates on a basket, the input vector gets updated with the proposed steps sampled by a physics manager. This uses cross sections tabulated from a *Geant4* physics list for all the processes involved by the simulation. The vectorized geometry navigator queries the geometry for the geometrical step limits for each track. After invoking the field propagator, the track vector gets gradually smaller as particles reach their proposed steps, so the compacting mechanism has to be called several times for the vector container. The code can automatically switch to a scalar approach when the vectors become inefficient for vector processing. An important mechanism implemented in the scheduler is the possibility to postpone single particles (not yet fully propagated) to be regrouped later with others matching the same locality criteria. Postponing is triggered systematically when geometry locality can be easily fulfilled (e.g. after firing a new event), but much less in the "sparse" regime when particles are spread in the full detector, having a wide range of energies. At the end of a propagation step, all particles (primaries as well as generated secondaries) are copied in the output vector of the basket.

3.1. Workload balancing and queue control

An important concurrency feature is automatic workload balancing. In *GeantV* this is achieved by using a single queue where any worker thread can retrieve the next basket to be processed. In this approach, an empty queue implies wasted idle resources waiting for work, but an over-populated queue points to inefficient vector size and additional overheads due to inter-thread communication in the "basketizing" phase. The scheduler has therefore to constantly monitor the work queue load and take actions to correct the "bad" work regimes.

The parameters used to control the queue sanity are the low and high watermarks for the number of contained baskets, which are subject to optimization. These are now chosen in a more or less arbitrary manner, and reaching them triggers specific actions. The low watermark triggers a special garbage collection by event range, to transport with priority tracks belonging to "old" events. Once an event and its corresponding data structures get flushed and digitized, the generator produces a new event, increasing the queue size. The high watermark triggers a re-evaluation of the thresholds used by the locality filters to "basketize" tracks, generating larger vectors and therefore decreasing the queue size at the profit of vectorization.

The algorithm for choosing the "appropriate" vector size is an important scheduling parameter to optimize. This depends on many factors, like the filter type, physics, or geometry complexity. These dependencies and correlations with other scheduling parameters are not well understood yet, but we are currently doing scans of the performance for different parameter values. In figure 2 we can observe that the performance can vary by more than 30% when changing the value for the particle vector size, with an optimal value of 64 tracks per basket. This corresponds however to a very simple geometry setup and just proves the principle. One possible approach that we are considering for optimizing such critical model parameters is based on genetic algorithms [7], as described in section 5.

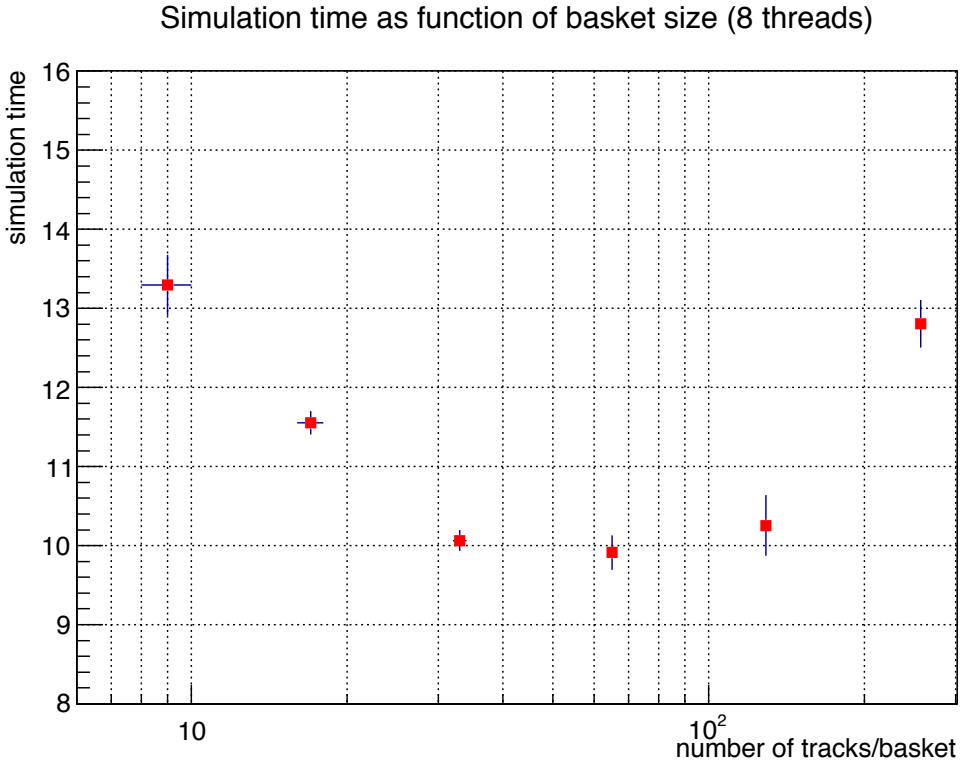

Figure 2. The simulation time has an optimum against the track vector size. Small vectors imply inefficient vectorization and dispatching overheads, while long vectors are penalized by frequent garbage collections in the "sparse" regime when locality is hard to enforce. The plot was produced on X86_64 apple-darwin13 Core i7 supporting AVX2.

4. Scalability and performance

Scalability is a very important requirement for the *GeantV* scheduler in order to minimize the memory footprint and maximize cache coherency. This is difficult to achieve due to the fine-grained parallelism, which requires concurrent data scatter and gather between propagation steps.

The first non-vectorized version of the prototype only needed to copy track pointers to new baskets, which was a lightweight operation that could be done for up to 12 workers by a single scheduling thread. The current vectorized version has to copy the full track data, which can easily bottleneck when executed by a single thread. An improved version of the scheduler performs now concurrent re-basketizing, but still suffers from an important serial bottleneck due to concurrent copy operations of track data onto a single receiving track container. To alleviate this, we are investigating an approach with multiple filtering threads managing alone the copy operations for subsets of the total basket types.

We are constantly monitoring the scalability of the vector prototype using the simple geometry setup presented in figure 3a, imported from one of the *Geant4* novice examples. This allows testing and comparing regularly different approaches for concurrency. Most of the tests are performed on an Intel® Xeon® CPU E5-2695 v2 @ 2.40 GHz machine with 24 native threads supporting AVX2 instructions. It is clear that more work needs to be done to improve this property, and the current goal is to achieve a good scalability up to at least 12 workers. One of the important milestones of the project is verifying that the new transport scheduling approach brings indeed performance improvements compared to the classical approach. This requires the ability to reproduce in a realistic way the physics (at least electromagnetic) for a complex setup and compare the performance with a standard *Geant4* run. Since we do not have yet all the ingredients to do that, we were aiming for a more simple check to hint for performance improvement or degradation.

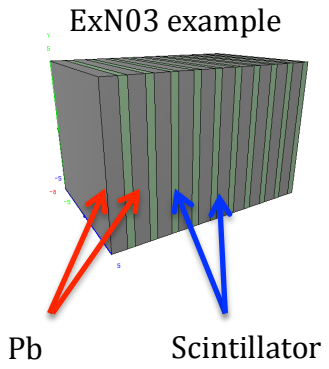

Figure 3a. Simple *ExN03* setup used for the nightly scalability tests run on.

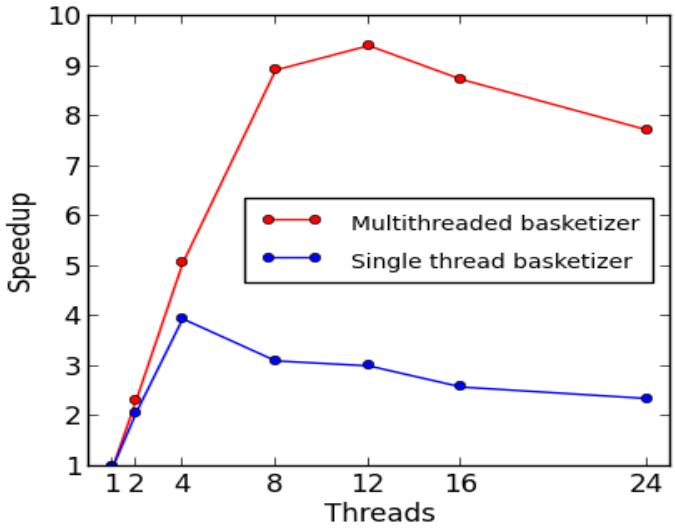

Figure 3b. Scalability plots of the speedup against number of worker threads. The blue line corresponds a single thread performing track re-basketizing between tracking steps. The red line corresponds to the current concurrent basket filling.

To reproduce the physics using a standard *Geant4* physics list we tabulated the cross sections for all the processes, particles and elements involved in the simple setup from Figure 3a. We have also tabulated final states coming from these processes, and implemented a special tabulated cross section process to have the exact same behavior in *Geant4*. Without entering into the full details of this procedure, we were able to compare the results for the following three configurations: *Geant4* simulation using the tabulated cross section process, *GeantV* using the scalar *ROOT* geometry for navigation, and *GeantV* using the vectorized *VecGeom* geometry to check the impact of vectorization.

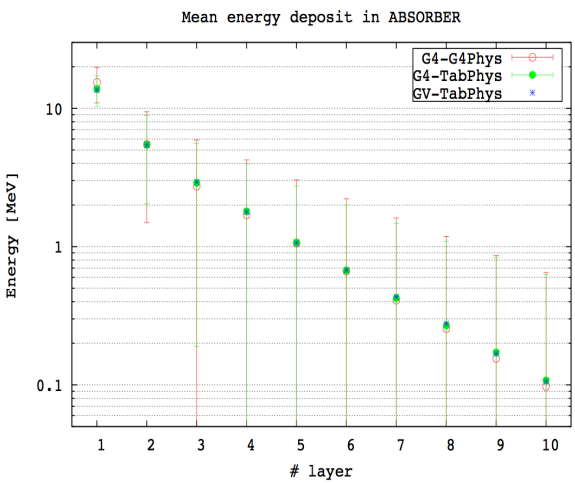

Figure 4a. The *GeantV* simulations based on tabulated physics and the corresponding native *Geant4* simulation produce compatible physics results (here energy deposit per layer in the *ExN03* setup)

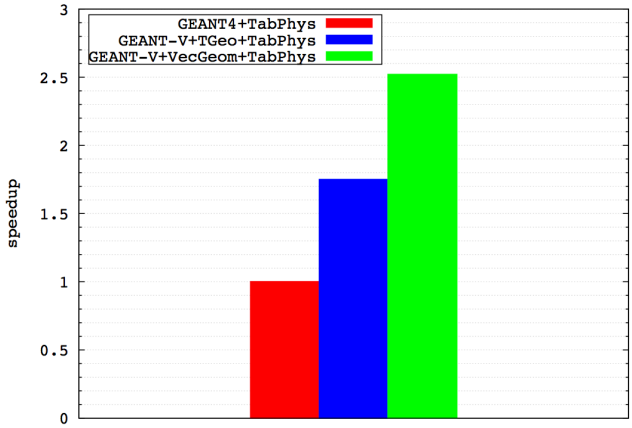

Figure 4b. Performance profile of the vector prototype for the *Geant4 ExN03* example. The performance improvements in case of tabulated physics are mainly due to geometry vectorization and better caching, despite the data gather overheads.

While getting statistically compatible results for the energy deposit and track length in the scintillator layers for incident electrons with energies ranging from *30 MeV* to *30 GeV*, we can observe a sizeable improvement of the total transport time even for this very simple setup compared to the *Geant4* + tabulated physics case. While an important contributor appears to be the vectorized geometry, the caching gains due to vector treatment have clearly a major impact. We will soon test a more complex setup, where we hope that the increased computational complexity of the vectorized code will enhance the benefit of the vectorized treatment of tracks.

5. Plans for optimising scheduling parameters

The *GeantV* scheduling strategy is based on a model that has a certain complexity and a considerable number of critical parameters that need to be tuned. The tuning procedure itself is not obvious and requires the understanding of single parameter behaviors, work that has already started.

We are currently investigating the possibility of using genetic algorithms as one possible tool for the optimization procedure. This will use as model "chromosomes" the states or ranges of the current model parameters, like the work queue thresholds, the vector sizes per locality filter or the number of worker threads. The evolution of the population of chromosomes will be done by mutation or crossover of these properties, and aim to improve the total simulation run time while keeping memory under controlled limits.

Since it is likely that the phase space for this procedure repeated for different geometry and physics setups can hardly be covered by a single set of parameters for all cases and even from the beginning to the end of the same simulation, we are foreseeing an adaptive behavior of the *GeantV* prototype, based on a very short training time of the model and a parameterization extracted from the genetic algorithm procedure.

References

[1] Apostolakis J, Brun R, Carminati F and Gheata A 2012 *J.Phys: Conf. Ser.* **396** 022014 http://iopscience.iop.org/1742-6596/396/2/022014

[2] S Agostinelli et al 2003 *Nuclear Instruments and Methods* **A 506** (53pp)

[3] http://root.cern.ch
[4] Apostolakis J, Brun R, Carminati F, Gheata A and Wenzel S 2014 *J. Phys.: Conf. Ser.* **513** 052038
http://iopscience.iop.org/1742-6596/513/5/052038
[5] Marek Gayer et al 2012 *J. Phys.: Conf. Ser.* **396** 052035
http://iopscience.iop.org/1742-6596/396/5/052035
[6] Bilk A, Girkar M, Grey P and Tian X 2002 *Int. J of Parallel Programming* **30**, No. 2
[7] Goldberg G 1989 Genetic Algorithms in Search, Optimization, and Machine Learning **ISBN-13**: 078-5342157673