Removing Inefficiencies From Scientific Code: A Case Of The ATLAS Experiment [⋆]

André Pereira[1,2], Alberto Proença[1], and António Onofre[1,2]

[1] Universidade do Minho, Portugal
[2] LIP-Minho, Portugal
{ampereira, aproenca}@di.uminho.pt, antonio.onofre@cern.ch

Abstract. This paper presents a set of methods and techniques for removing inefficiencies in scientific code. A scientific application, of the ATLAS experiment, is used as a case study. Here, we implement and test a set of optimizations to deal with performance inefficiencies in a organized approach: identification, removal at application development and at application runtime. These optimizations consider code, algorithm and data structure design, parallelization in shared memory systems, and runtime configurations for NUMA environments. These optimizations target different multithreading and multiprocess combinations, and CPU affinities. We expect that this communication will aid scientists to become more aware of common efficiency pitfalls in scientific code.

Keywords: High Performance Computing, Efficiency Optimization

1 Introduction

The European Organization for Nuclear Research (CERN) is a consortium of 20 European countries, founded in 1954, with the purpose of operating the largest High Energy Physics (HEP) experiments in the world. The instrumentation used in nuclear and particle physics research is essentially divided into particle accelerators and detectors, alongside with the facilities necessary for delivering the protons to the accelerators. The Large Hadron Collider (LHC) particle accelerator speeds up groups of particles close to the speed of light, in opposite directions, resulting in a controlled collision inside the detectors (each collision is considered an event). The detectors record various characteristics of the resultant particles, such as energy and momentum, which originate from complex decay processes of the collided protons. The purpose of these experiments is to discard or confirm the Standard Model of High Energy Physics, to discover new physics if present at the current energy scale, in addition to the Higgs boson recently discovered.

The ATLAS experiment, a key project at CERN, is conducting most of the crucial experiments on both the Top quark measurements and Higgs boson searches. Approximately 600 millions of collisions occur every second at the

[⋆] Detalhes financiamento...

LHC. Particles produced in head-on proton collisions interact with the detectors of the ATLAS experiment, generating massive amounts of raw data as electric signals. It is estimated that all the detectors combined produce 25 petabytes of data per year [11, 12]. The amount of data is expected to grow after the LHC upgrade, which purpose is to increase the amount of energy of the accelerated particle beams, producing more reactions. This data passes a set of processing and refinement until it is ready to be used to reconstruct the events by specific applications. The application varies according to the HEP theory being analysed. The ATLAS detector has a 2% experimental resolution associated with every measure, which affects the quality of the data. This can be overcome by doing an extensive search among a 2% space around every particle parameter, improving the reconstruction quality. At this stage, different research groups in the same experiment enforce positive competition to produce results in a fast and consistent way.

The growing amount of data to process increases the need of processing the analysis applications in the same time as before. The common approach is to increase the computational power of the research group, which is usually expensive. However, applications inefficiently use the available computational resources and, if properly optimized, the computational throughput could be highly boosted. This gives the research group an edge over the competition in terms of results quality and amount of events processed.

This paper is organized as follows: section 2 briefly presents the Top quark and Higgs boson decay process and introduces a short characterization of the `ttH_dilep` application used as case study; section 3 presents and characterises the inefficiencies identified in the application; section 4 shows the process of removing the inefficiencies and optimizing the application at development and runtime stages; finally, section 5 concludes the paper and mentions future work.

– CERN, LHC and ATLAS
– Research at ATLAS (mention top decay and higgs)
– What the paper proposes to achieve (relevance to the physicists and general scientific community)
– Mention the difference between performance and efficiency
– Paper organization

2 Top Quark and Higgs Decay

In the LHC, two proton beams are accelerated close to the speed of light in opposite directions, set to collide inside a specific particle detector. From this head-on collision results a chain reaction of decaying particles, and most of the final particles react with the detector allowing their characteristics to be recorded. One of the experiments being conducted at the ATLAS detector is related to the studies of Top quark and Higgs boson properties. Figure 1 presents the schematic representation of the top quark decay (usually addressed as the $t\bar{t}$ system), resultant from a head-on collision of two protons.

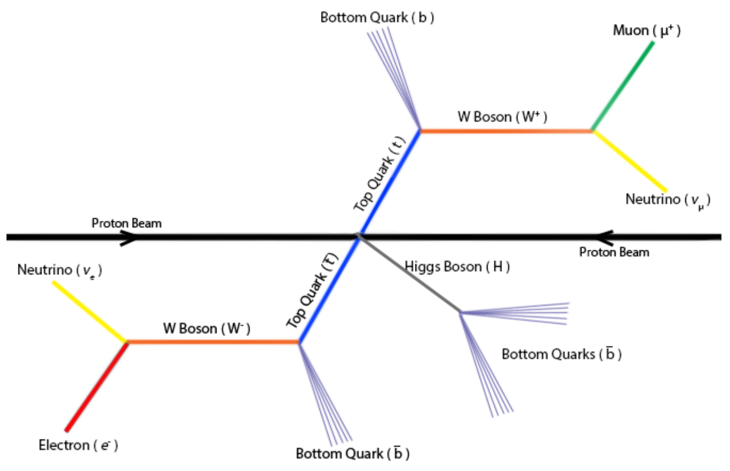

Fig. 1: Schematic representation of the $t\bar{t}$ system and Higgs boson decay.

The ATLAS detector can record the characteristics of the bottom quarks, detected as a jet of particles, and leptons(muon with a positive charge and electron with a negative charge). However, the neutrinos do not interact with the detector and, therefore, their characteristics are not recorded. Since the Top quark reconstruction requires the neutrinos, their characteristics are analytically calculated with the information of the quarks and leptons, in a process known as kinematical reconstruction. The reconstruction of the whole $t\bar{t}$ system has a degree of certainty associated, which determines its quality.

The amount of bottom quark jets and leptons detected may vary between events, due to other reactions simultaneously occurring with the top quark decay. As represented in figure 1, 2 jets and 2 leptons are needed to reconstruct the $t\bar{t}$ system, but the input data for an event may have many more of these particles associated. It is necessary to reconstruct the system for every combination of 2 jets and 2 leptons in the input data (referred only as combination). Then, only the most accurate reconstruction is chosen for each event.

The Higgs boson reconstruction is performed after the $t\bar{t}$ reconstruction, with the two jets that it decays to. This adds at least two more jets to the event information, increasing the number of possible combinations, and it is not possible to distinguish them from the $t\bar{t}$ jets. The Higgs boson reconstruction is performed with the jets that are not used in the $t\bar{t}$ system. The overall quality of the event reconstruction depends on the quality of both $t\bar{t}$ system and Higgs boson reconstructions.

The ATLAS detector experimental resolution induces an error of 2% in each measure of the particle characteristics. The reconstruction of both $t\bar{t}$ system and Higgs boson will have an error associated due to this. To improve the quality of the reconstructions it is possible to apply a random variation to the particles parameters, with a maximum magnitude of 2%, a given amount of times and chose the reconstructions that better fits the theoretical model. The quality of the reconstructions, and application execution time, are directly proportional to the amount of variations per combination performed. The goal is to perform as many variations as possibly within a reasonable time frame.

The `ttH_dilep` application was designed the reconstructions of the $t\bar{t}$ system and Higgs boson, as explained above. The application flow is presented in figure 1. The data of an event is loaded to a single global state shared among the application and the LipMiniAnalysis library, and it is overwritten every time a new event is loaded. Then the event is submitted to a series of cuts, which filter the events that are not suited for reconstruction. When an event reaches the cut 20 the $t\bar{t}$ system and Higgs boson are reconstructed in a function named `ttDilepKinFit`. If an event has a possible reconstruction it passes the final cut and its information is stored.

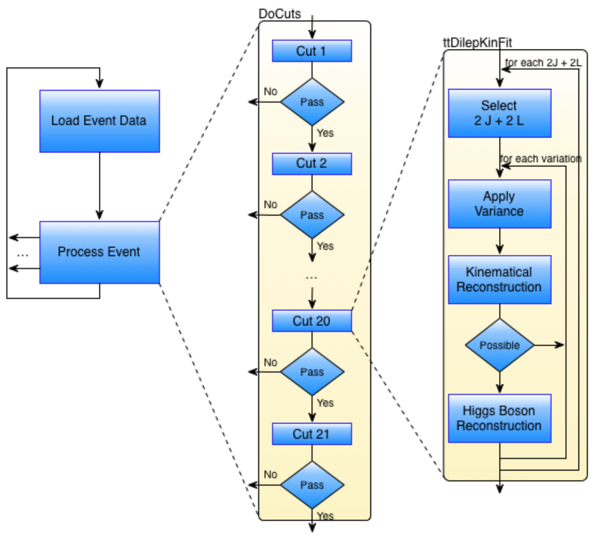

Fig. 2: Schematic representation for the `ttH_dilep` application flow.

– Top Quark and Higgs Boson decay from a physics point of view
– ATLAS detector experimental resolution
– ttH_dilep brief characterisation

3 Identification and Removal of Code Inefficiencies

Inefficiency removal is a two stage iterative process. First, the code must be analysed to identify the critical sections of the code that take the most time executing. This can process can be automatised by using third party tools, such as gprof or callgrind, which produce a report listing the percentage of time spent in each of the application functions. A more detailed analysis can be obtained using tools similar to PAPI, where hardware counters are measured to obtain cache miss rates, float point instructions, and other low level information.

The test environment used in both this section and section 4 is a dual-socket system with two Intel E5-2670v2 with 10 cores (20 hardware threads) at 2.5 GHz each, 256 KB L2 cache per core and 25 MB shared L3 cache, coupled with 64 GB DDR3 RAM.

3.1 First Iteration

In a preliminary analysis using callgrind, we identified the `ttDilepKinFit` as the most time consuming function when considering a significant number of variations, as shown in figure 3. One particularity was that the Pseudo-Random Number Generator (PRNG) seed was being reset for every variance applied to the particles parameters, consuming XXX for 512 variations.

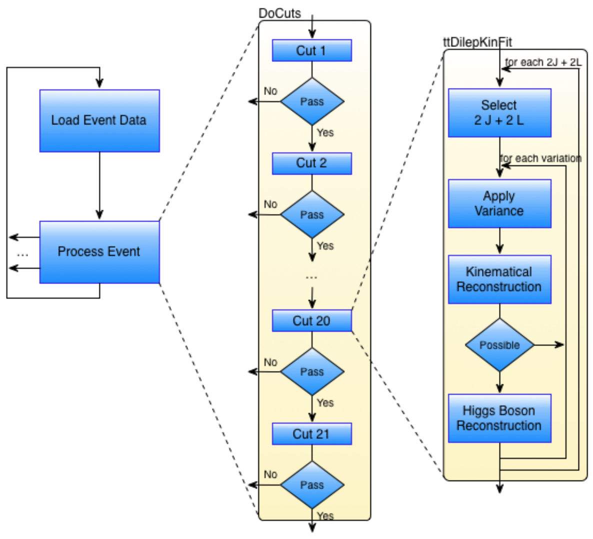

Fig. 3: Schematic representation for the `ttH_dilep` application flow.

Since the period of the Mersenne Twister, which is the PRNG used, is approximately $4.3 * 10^{6}001$ and the maximum amount of generated numbers in the input test file used, for this amount of variations, is $1.5 * 10^{9}$, it is not necessary to reset the seed. The removal of this simple inefficiency granted speedups up to 1.8, as shown in figure 4. Note that there are not tests for 1024 variations as the application execution time rendered them infeasible.

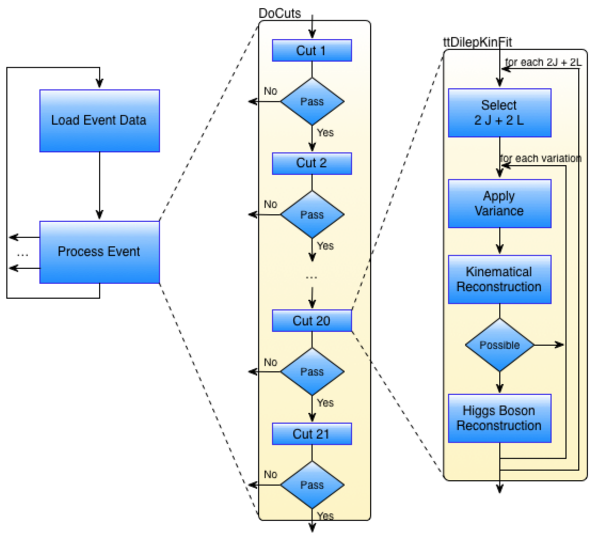

Fig. 4: Schematic representation for the `ttH_dilep` application flow.

3.2 Second Iteration

Even with the PRNG optimization the `ttDilepKinFit` remains the critical region in the application. Since there are no relevant code inefficiencies left the next step is to parallelize `ttDilepKinFit`. Keep in mind that it is not possible to parallelize the whole event processing since part of its information is stored in LipMiniAnalysis library, which we did not have permission to refactor.

Parallelization of `ttDilepKinFit` implies modifying its flow. The data of each variation is overwritten for each different combination of an event, so it is needed to create a data structure of all combinations of the event being processed. Picking the lepton/jets combination depends on all previous combinations, which serializes the construction of the data structure. Each parallel task (indivisible work segment) picks a combination with variations still to compute, varies the particles parameters and performs the kinematical and (if possible) Higgs reconstruction. Then, since only the most accurate reconstruction must be considered, a parallel merge is performed. Figure 5 presents the sequential and parallel workflow for `ttDilepKinFit`.

The parallelization was performed using OpenMP. The parallel tasks are grouped into threads, which holding the data of the best reconstruction to minimize the complexity of the best reconstruction merge by reducing through all the threads instead of tasks. The amount of tasks for each thread is balanced dynamically by the OpenMP scheduler. Each thread has a private PRNG initialized with different seeds to avoid correlation between the numbers generated.

Figure 6 presents the speedups for

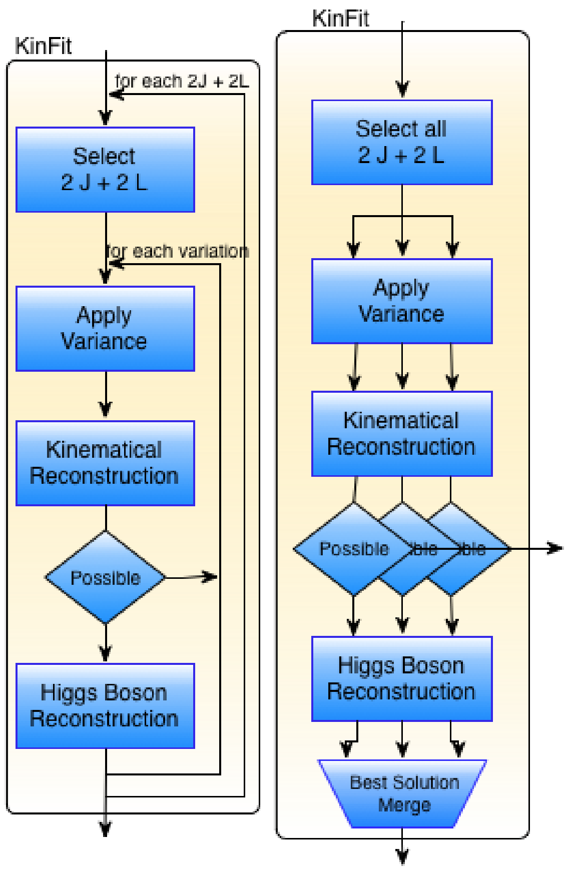

Fig. 5: Schematic representation of the `ttDilepKinFit` sequential (left) and parallel (right) workflows.

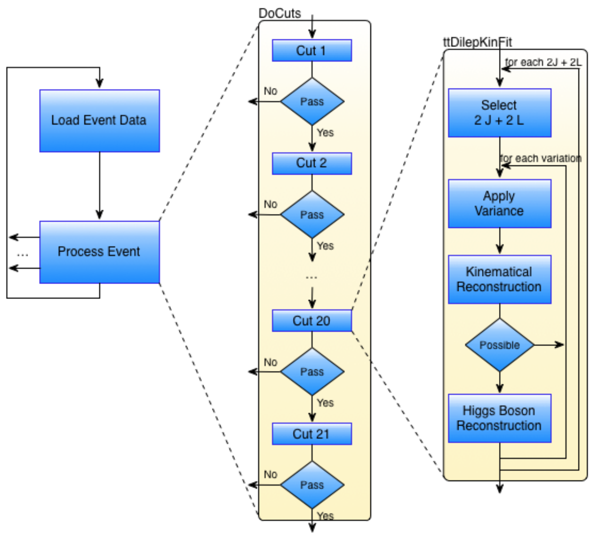

Fig. 6: Speedup for `ttH_dilep` parallel non-pointer version application.

3.3 Third Iteration

We used Intel's VTune to search for hotspots (bottlenecks) on the application, since this tool is best suited for profiling parallel applications while providing an easy to use GUI. From a preliminary analysis it was concluded that the application was spending around 20% of the time building the combination data structure for 256 variations.

A closer look to the data structure revealed that there were inefficiencies that were affecting the performance in specific situations. There is read-only control information that is being replicated in each element of the data structure. If the elements were to share a pointer to such data the overhead of constructing the data structure would be reduced. However, this adds an increase in communication, which could lead to worse cache management and added overhead in NUMA environments. Nevertheless, this optimization was implemented and tested (addressed as *pointer version*), with its speedups presented in figure 7.

As predicted, the best speedup occurs when using 8 threads, which counts for almost one device on the system. Even with 10 threads the performance of the application suffers due to the increase of concurrent accesses to the L3 cache on the system due to the shared data. This decrease in performance is even more noticeable when using both CPU devices, where the non-pointer version still scales. However, this implementation is more efficient than the former when using only one device. Note that the superlinear speedups are due to the PRNG optimization and the reduction in the data that each thread has to process, making it more suitable to be stored in the private L2 cache of each core, therefore reducing the slower accesses to L3 cache.

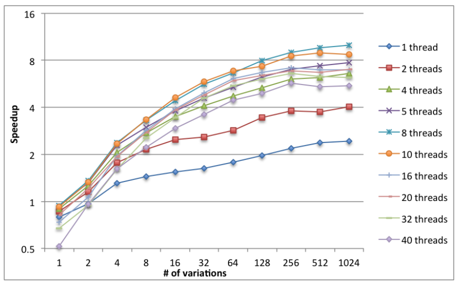

Fig. 7: Speedup for `ttH_dilep` parallel pointer version application.

– Present MT gains?
– Describe the task flow with data structure?

4 Identification and Removal of Runtime Inefficiencies

The identification and removal of inefficiencies follows the same iterative approach presented in section 3.

4.1 First Iteration

Without the sensibility provided by the tests in section 3, a scientist would incur in the pitfall of using all available cores (and even all hardware threads) on the system hoping that it would provide the best performance. While it may be true for the non-pointer implementation, it would inefficiently use the system computational resources, and using the single device highly efficient pointer implementation would provide a even greater waste.

Since the pointer implementation is the most efficient but only when using a single device, using multiple processes may provide better efficiency. As it was not possible to refactor LipMiniAnalysis to change the event information from a global to a private state, an MPI implementation was discarded due to the communication overhead necessary in each event processing.

A characteristic of particle reconstructions at CERN is that the processing is made by executing the application on a vast set of 1GB input files. The system resources can be efficiently used if it is performed a careful balancing of the input files across a set of application processes, producing the same goal of an MPI implementation but with no need for communicating between processes. A dispatcher was devised, which takes a set of input files and creates a given amount of `ttH_dilep` processes. It is then responsible for dispatching the files to the different processes in a queue-like approach, and monitors their execution. A set of 20 input files was considered for testing purposes, with different configurations of processes and threads per process.

Figure 8 presents the speedups using 2, 4, 5, 8, and 10 processes for different thread configurations to fill one or both CPU devices.

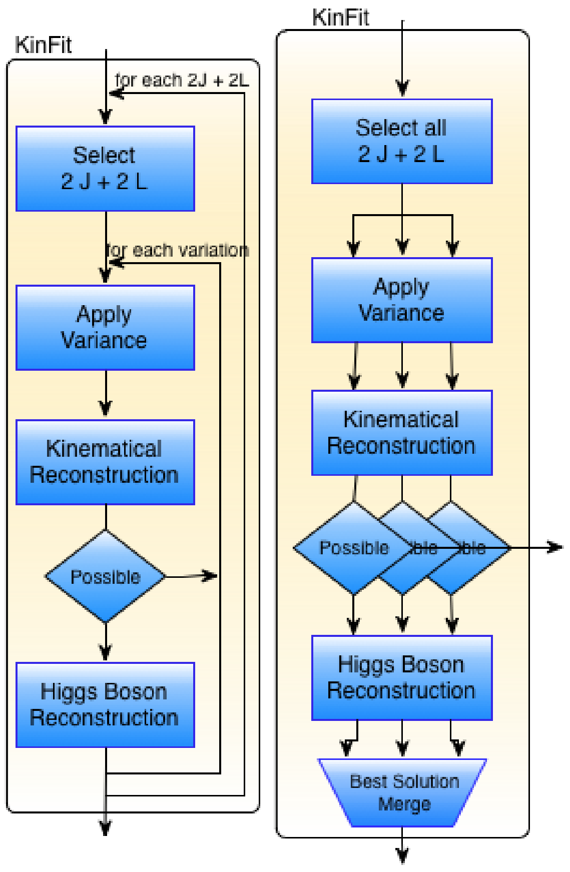

Fig. 8: Schematic representation of the `ttDilepKinFit` sequential (left) and parallel (right) workflows.

- At runtime
  - Thread affinity experiments vs standard OpenMP affinity (force bad affinity examples that may occur to compare the performance?)
  - Using all available cores is not always profitable
  - Many processes/threads combinations, also testing thread affinity

5 Conclusion

Acknowledgments:

References