Strategies employed for LHC software performance studies

Andrzej Nowak, CERN openlab
February 2010

Executive Summary

The objective of this work is to collect and assess the software performance related strategies employed by the major players in the LHC software arena: the four main experiments (ALICE, ATLAS, CMS and LHCb) and the two main software frameworks (Geant4 and ROOT). As the software used differs between the parties, so do the directions and methods in optimization, and their intensity. The common feeling shared by nearly all interviewed parties is that performance is not one of their top priorities and that maintaining it at a constant level is a satisfactory solution, given the resources at hand. In principle, despite some organized efforts, a less structured approach seems to be the dominant one, and opportunistic optimization prevails. Four out of six surveyed groups are investigating memory management related effects, deemed to be the primary cause of their performance issues. The most commonly used tools include Valgrind and homegrown software. All questioned groups expressed the desire for advanced tools, suitable for use by individual non-expert users, thus indirectly indicating limited will to spawn concentrated activities by experts. This paper outlines several recommendations, which (if implemented) might allow optimization efforts to be more effective.

Table of Contents

Introduction and motivation

Recent progress in microprocessor and system board technologies vastly differs in character from the advancements made several years ago. Due to certain effects in silicon and manufacturers' choices, we are no longer the beneficiaries of architectures effortlessly scaling up with the clock frequency. Due to these changes in the computing landscape, openlab is actively monitoring solutions and activities which might lead to improved performance in place of hardware upgrades, once taken for granted.

Performance optimizations can result in significant savings in hardware, and can lead to results being delivered with better latency. If the software running in the CERN computing center had been improved by about 1%, the resulting saving would reach hundreds of thousands of Swiss Francs in hardware. Thus it is not hard to understand why the Organization and its members are recommended and motivated to look favorably upon optimization.

The objective of this work is to collect and assess the software performance related strategies employed by the major players in the LHC software arena:

- The ALICE experiment,
- The ATLAS experiment,
- The CMS experiment,
- The Geant4 collaboration,
- The LHCb experiment,
- The ROOT collaboration.

This analysis involves performance monitoring and optimization habits and strategies as well as a high-level overview of the type of related efforts undertaken by the software teams behind the entities mentioned above. This paper is not a comprehensive in-depth study meant to examine each individual aspect of software optimization or the work done, but rather a general summary. As the software used differs between the parties, so do the directions and methods in optimization and so does the intensity of those efforts. The goal, however, is common: better overall software performance, yielding lower cost, higher throughput and optimized latency. Thus, a secondary objective of this work is to help to reach this goal and to possibly spawn additional or unified performance optimization efforts on behalf of the interested parties.

While openlab acts mainly as an observer and a provider of competence in certain areas, it also has the capacity to provide tools and hardware meant to facilitate studies. This work is meant to reinforce this effort, and provide input to future decisions concerning activities and directions, especially considering the shifts in the mainstream perception of performance monitoring.

The survey approaches two major areas: the process and the tools. It has been split into several parts dealing with particular aspects of the aforementioned issues. In order to establish a clear and – more importantly – up to date image of techniques and requirements for performance studies, interviews have been conducted with all major LHC software providers. These include the representatives of four major experiments – ATLAS, CMS, LHCb and ALICE, as well as the representatives of the two major software frameworks: Geant4 and ROOT. In addition, one needs to take into account the generic operations and requirements imposed by High Energy Physics (HEP) and CERN, as well as those imposed by openlab for internal studies.

Current situation

General remarks

This section is split into four parts which describe in general terms the combined approach from all parties.

The prevalent feeling shared by many is that software performance is not a top priority, and that maintaining it at a constant level is a satisfactory solution, given the resources at hand. Many frameworks haven't yet fully transitioned to modern compilers and 64-bit architecture, and the performance gain from such a transition might in some cases be equivalent to optimizations introduced by hand in the code. It should be explained that in certain cases a constant level of performance comes with the benefits of improved detail or accuracy, and then the performance is considered as improved.

Significant bottlenecks

There are situations in which even a complex application has a clearly defined bottleneck. One case is that of "low hanging fruit", which is perceived as a clear motivator for better performance. In another case, a well visible obstacle can often be complex and impossible to deal with using simple methods, but it can commonly be easily quantified with sufficient detail to start working on it.

All of the parties surveyed have indicated poor memory characteristics (such as inflated size and low usage efficiency) as their main performance worry. This is indicative of memory management issues such as suboptimal allocation and deallocation, which lead to growing memory pools and memory fragmentation. Memory bandwidth was mentioned as a current or possible future bottleneck in two cases. Memory latency was not typically perceived as a problem.

Processor and architecture related bottlenecks were said to be appearing mostly as a side effect of other issues, but have sometimes prompted the owners of the code to direct investigations in that direction.

Finally, many I/O related bottlenecks have recently been tackled with good results by developers from ROOT and the experiments. These changes introduced through cooperation have benefited all experiments. However, several groups indicated that even further optimizations in this area still might be possible.

Performance optimization priorities

In most cases, memory layout and usage patterns were mentioned as the top items currently investigated or marked for investigation. Thus, while optimization priorities differ vastly across camps, there is one common group of activities that stands out, with the common objective of mitigating the various side effects of memory usage.

Memory fragmentation, leaks, allocation and abuse are leading to pressure and non-locality, and have been widely cited as the cause of many performance issues. While in some situations these problems were a result of coding errors, it should

be noted that such symptoms are not always caused by mistakes in code and in some cases have a lot to do with the environment – the compiler and relevant libraries, including those managing memory. These effects are currently being investigated in detail by 4 out of 6 surveyed groups – this includes both initial and perpetual investigations. ALICE has claimed to have already finished such a phase and has declared the code stable in that respect, and LHCb is preparing to investigate.

Difficulties with memory management have naturally paved the way for the evaluation of multi-core and multi-processing technologies, which give the promise of improving locality by running a single code base and of saving memory through the involvement of shared structures. However, given that multi-threaded development is often considered to be an order of magnitude more difficult and time consuming than developing single-threaded applications, memory issues alone are not typically considered to be a sufficient justification for more widespread and systematic activities on this front. In practice, multi-threading activities are scarce, often unadvanced and are considered to be even less important than the efforts directly related to performance. In particular, it is often the case that for good threading scalability a fundamental review of the codebase is needed, and due to various reasons such efforts are very limited or inexistent.

CMS and LHCb are also investigating microarchitecture-related optimization directions as a secondary activity. These include microarchitectural investigations, platform investigations and the usage of advanced hardware-level analysis of the software. CMS has already had interesting results, and a more general framework is being built to aid with this kind of investigations, as well as to facilitate the interpretation and dissemination of PMU-based performance monitoring results. In addition, ATLAS and the Geant4 PH/SFT team are working on reviewing the ATLAS simulation with focus on processor efficiency, following a joint recent assessment paper.

Performance monitoring processes and strategies

The survey has shown that the approach to performance optimization differs a lot from case to case. In general, despite some organized efforts, a less structured approach seems to be the dominant one, and opportunistic optimization prevails.

ATLAS, CMS, Geant4, ROOT have a regular performance regression check in place, while LHCb and ALICE are currently working on implementing similar measures. Not surprisingly, in some cases the focus is shifted towards maintaining a constant level of performance rather than improving it. CMS and LHCb have designated people whose time is dedicated mainly to optimization, not only in the domain of memory issues. ROOT performance seems to be actively managed by a group of senior programmers, however some potential areas of improvement identified earlier may not be approached without additional manpower. Geant4 conducts periodic code profiling and reviews which proceed in cycles, and allow the recovery of CPU time allocated for improved physics modeling – such activities are reported to have yielded significant improvements. In the case of ATLAS, occasional centralized code reviews are reported to allow for sizeable gains in performance. In addition, ATLAS, Geant4 and ALICE also depend on best efforts from their numerous programmers for everyday optimization.

The two mentioned activities started by CMS and LHCb are aimed at developing ways of improving the software through a combined and comprehensive approach. Thus, processor usage efficiency and platform related factors are also considered in addition to memory related issues. One of the expected results of these efforts is a higher-level strategy for performance optimization, as well as general guidelines and principles at some point in time. In the case of LHCb, this effort is especially important, as the group concerned seems to lack a current strategy for efficiency improvement.

Tools and requirements

Currently used tools

There are several tools actively and commonly used by the groups surveyed:

- The Valgrind suite (valgrind, callgrind, cachegrind, etc) – 6 groups
- Own tools (depending on the project) – 4 groups
- Google performance tools (i.e. tcmalloc) – 2 groups + 1 experimenting
- Perfmon2 (pfmon) – 2 groups
- Intel tools (VTune/PTU) – 1 group experimenting

There is a trend of heavy reliance on the Valgrind suite and on in-house developed tools. While the former stems from the memory management issues described earlier, the latter might be indicative of the lack of proper tools in the environment. Usage of tools offered by openlab, such as perfmon2 and the Intel tools, is not widespread.

Basic functional requirements

Current typical functional requirements for tools include:

- Memory related statistics
    - Allocations and deallocations (usage patterns, allocation patterns, pressure, layout)
    - Categorize by calling stack
    - Tracking down leaks
- Call graph building
- Event based sampling
    - Per-function
    - Per-module

Additional functional requirements

All of the surveyed groups indicated some additional functionality would be welcome. These wishes could be summarized in the following way:

- Being able to track I/O bottlenecks easily
- Being able to specify how much processing time is attributed to specific segments of code
- Enhanced memory statistics and memory allocation effects; in particular:
    - Object layout on the heap
    - Page sharing amongst processes
    - Usage histogram
- Event based sampling with stack traces

It should be noted that some of these requirements are fulfilled by existing tools (for example memory locality tools in the Intel Performance Tuning Utility) or are being addressed in current developments, such as for example I/O tracking in ROOT.

Non-technical guidelines for tools

In addition, the following non-technical guidelines have crystallized:

- It is rather important that the tools are intuitive to use and produce easily understandable results when needed, preferably supplemented by graphics
- The tools should work in a stable and reliable fashion even with complicated software frameworks
- It is better that the tools are open source, but this is not a strict requirement
- The tools should not be proprietary, but this is not a strict requirement
- Portability is a concern, but it's not a strict requirement
- The tools should be made available free of charge
- It is important that the tools are accessible without superuser privileges past the installation phase
- The tools should be easy and convenient to use by non-expert physicists and programmers

It should also be noted that in some groups there is opposition to external tools, and a strong preference exists for developing in-house solutions. These often prove effective and tailored to the specific needs of the group, however their development and maintenance requires additional resource expenditure.

Recommendations and directions

General remarks

It has already been mentioned that optimization is rarely seen as a key activity. This might be attributed to several factors. One is the chronic shortage of funds and manpower, which forces the assignment of manpower to other tasks. Another reason might be the indirect benefit from optimizations, and the lack of certainty of achieving good results. Finally, optimization is seen as a difficult and tedious task. That is not only due to the lack of adequate tools, but also due to the frequent need of deep understanding of the software and the underlying hardware.

Analysis

The global image obtained suggests that there is not much organized, regular and concentrated performance optimization work going on. In certain cases finding appropriate representatives to interview or extracting accurate information proved to be challenging. In some groups the activities already performed would benefit greatly from better organization and additional communication with entities from different projects.

CMS and ROOT have a conscious effort focused on performance, the LHCb one just started, and a similar new project is being founded within Geant4, in addition to code reviews held to date. However, these efforts are not always high-profile, as in the case of LHCb for example. Out of the surveyed six, ATLAS maintains low involvement in performance oriented community work due to focus on issues that are different from the ones typically discussed.

The results obtained from this survey suggest strongly that any tools employed should provide output which is comprehensible for a non-expert user and hassle-free to set up. Unfortunately, this particular requirement conflicts with the complex nature of computers and the software which is driving them. On the other hand, there certainly are improvements that could be made to currently used tools, so that the data gathered is analyzed more thoroughly than the current standard might suggest, and displayed using less technical content. Another issue mentioned in this context was the stability of tools – many of them seem unfit for the type of software used at CERN.

There have been numerous concerns about the viability and usefulness of hardware-level analysis. This is understandable, given the level of obscurity and steep learning curve often associated with the topic. However, being blessed with a relatively homogenous platform (the PC) as the major vehicle for LHC-era software, one should not ignore the rich built-in performance monitoring capabilities of the platform, even if the learning curve is steep.

Several groups have used an approach in which every user is responsible for their own code, and in such a case automated facilities typically exist, which are designed to minimize performance regression. In this kind of scenario the best performance one can achieve without additional efforts could be described as “adequate”. This strategy maximizes productivity at a low cost, but at the same time it minimizes the potential benefits and often isolates expertise. It is not unreasonable to expect that a more combined effort would be required to get better results. Such is the case of ATLAS, where the notion of code ownership is very strong, but concentrated efforts yield good results and lasting improvements.

Teams which have chosen to take a managed and regular approach to the subject were in control of their software’s performance and have succeeded in moving optimization efforts forward despite manpower problems.

Recommendations

Based on the data and opinions gathered, the recommendations are as follows:

- Strategies
  - Experience shows that a unified and coordinated performance optimization strategy will always yield better results than uncoordinated efforts. This process works best if there is an entity driving the effort.
  - Several projects (i.e. ATLAS, Geant4, ROOT) have either split optimization efforts into "waves" of code examination and cleanups, or have executed periodic consolidated activities. Such a tactic will yield good results and will allow for changes to be introduced without the fear of disrupting functionality.
  - Deep in-house knowledge of the code, tools and the platform is indispensable for achieving optimal results.
  - It is important that code owners have an up to date image of the performance of their code in real, production environments. Performance data can often be easily gathered without noticeable penalties in terms of throughput.
  - Focusing on hotspots is often a good tactic as well. However, a combined and fundamental approach to performance will always allow for a frame of reference, which in turn might help to define expected improvements.
- Cooperation
  - Numerous joint efforts in the HEP community have shown that excellent results are produced when software providers (such as Geant4 or ROOT) have the opportunity to cooperate closely with software users (such as the experiments).
  - Educational efforts and community events allow knowledge exchange, efficient dissemination of good coding and optimization techniques, and, less importantly, help increase the awareness of the importance of performance studies. Such efforts also allow addressing common problems (i.e. inoptimal practices) before they even arise.
- Tools
  - In some groups, select members have embraced tools which are non-standard or experimental, and are not developed by the local community. While it is true that the properties of some of those tools might bring little or no direct benefit to the projects in question, there certainly is expertise outside of HEP which could be brought in.
  - The Intel Performance Tuning Utility is a piece of software which runs with the standard Scientific Linux kernel, and allows for some of the activities put on the current wish lists. Other tools which after some work might be able to help with the problems mentioned, are PIN, SystemTap and utrace. Good knowledge of the "market" will certainly aid in conducting an efficient optimization process.

As far as the activities of openlab are concerned, there are some areas which might benefit from a minor re-alignment:

- One is active work on closing the gap between the programmer and the hardware. Knowledge dissemination and the regular computing courses organized every 3 months are definitely a step in the right direction.
    - Action taken: In addition to regular openlab workshops, a new type of irregular workshop for experts has been founded in cooperation with Intel.
    - Action taken: openlab will also work towards other ways of disseminating knowledge related to combining high and low level performance tuning data.
- Another commonly mentioned problem was that of the stability and reliability of the tools. Openlab has brought several performance optimization tools to the table, Perfmon2 and the Intel Performance Tuning Utility amongst them. However, even though the mentioned tools often provide functionality sought by developers, they might lack reliability when working with HEP software. In addition, they would benefit from more credibility and proven success stories.
    - Action taken: this feedback has been carefully considered and will impact future openlab activities in this domain. A special program starting soon in cooperation with Intel will bring better tools.

Conclusions

Obviously, HEP communities seem to focus heavily on code feature sets and correctness, but, disappointingly, not very much on performance. This, amongst other factors, could be attributed to decades of hardware improvements and clock scaling taken for granted. Times have changed, and so has the hardware development model. A continued journey along the current path of single threaded, non-optimized processing will incur unnecessary costs both in terms of resources and time. Thus, in order to achieve good results with the limited resources at hand, it is important that:

- efficiency and performance optimization come back to the table as regular topics for discussion,
- optimization efforts are concentrated and well organized,
- multi-threading is explored as a solution to many of the described problems,
- expertise reaches involved parties and is shared and disseminated in a timely fashion.

In conclusion of the tool survey, it appears that performance related tools should be made more accessible and more reliable. At the same time additional efforts are needed to close the gap between the programmer and the hardware.

Acknowledgements

I would like to thank Sverre Jarp from openlab and Jeff Arnold from Intel for their valuable comments and insight.

I would also like to thank the following people for their input:
John Apostolakis (Geant 4), Gerhard Brandt (ATLAS), Rene Brun (ROOT), Paolo Calafiura (ATLAS), Gabriele Cosmo (Geant 4), Peter Elmer (CMS), Vincenzo Innocente (CMS), Daniele Francesco Kruse (CMS), Karol Krużelecki (LHCb), Stefan Lohn (ALICE), Zachary Marshall (ATLAS), Fons Rademakers (ROOT), David Rousseau (ATLAS), Matevz Tadel (ALICE) and others.