Optimization of selected components in MPD Root: Capabilities of distributed programming techniques

A. Fatkina, O. Iakushkin[a], O. Gasanova, L. Tazieva

Saint Petersburg State University, 7/9, Universitetskaya emb., Saint Petersburg, 199034, Russia

E-mail: [a] o.yakushkin@spbu.ru

The article analyses the prospects of optimizing the architecture and the execution logic of selected scripts available in MPD Root project. We considered the option of porting the scripts to allow execution on massive parallel architectures. We collected and structured large data illustrating the project's work:

- the project's dependency tables were drawn up with regard to the support of parallel and concurrent computing;
- the source code database was indexed to identify cross dependencies among architectural entities;
- the code profiling measurements were made at launch time of the scripts in question, the call sequences were analyzed, the execution time of the scripts was evaluated.

The study evaluated the prospects of using various libraries and platforms: CUDA, OpenMP, OpenCL, TBB, and MPI. The obtained measurements and the analysis of the best practices of the software under consideration allows to make recommendations for modifying MPD Root in order to optimize:

- vectorization of loops;
- transfer of continuous computing segments to co-processing architecture;
- source code segments whose operation can be represented as call graphs;
- source code segments that can be subject to load allocation between computing nodes.

Keywords: CUDA, STL, MPD Root, Parallel and Distributed Computing.

The work was supported by RFBR research project № 16-07-011113 and SPBU projects 0.37.155.2014 and 9.37.157.2014.

Introduction

MPD (Multi Purpose Detector) is a part of NICA (Nuclotron-based Ion Collider fAcility). MPD includes a large number of detectors employed to study the results of particle collision. The particles studied by MPD have different masses and range from protons to gold ions.

MPD Root is a framework designed to simulate experiments conducted on MPD and to analyze the resulting data. The framework uses ROOT and FairRoot components, which are also used in other experiments in nuclear physics. These components are tools for modelling and analyzing detector events that have already been proven effective.

According to the MPD design, raw data collected by the detector will amount to about 30 PB per annum, which corresponds to 19 billion events. For example, a single Au + Au collision can produce up to a thousand of secondary charged particles, which necessitates high speed data processing.

This paper presents research results that allowed to put forward recommendations on speeding up the performance of the framework being developed. The proposed optimizations are based on parallel data processing: specifically, vectorization, and distributed and multi-core computing (including using graphics accelerators).

Performance analysis of MPDxRoot framework tests

The MPD Root framework was analysed against three main criteria:

- Processor time spent executing algorithms; this parameter directly describes the application's runtime performance.
- Component dependencies; this parameter shows independent parts of the framework which may be executed in parallel.
- Computational capabilities of external libraries used by the system, with regard to the parallel and distributed programming technologies.

The following algorithm was employed in the analysis:

1. Doxygen utility was used to generate reports. We built dependency graphs for files, classes, and functions. Thus, we drew up the framework's structure, as well as a source code navigation system;
2. Valgrind CallGrind profiler was used to model graphs of mutual function invocations. They show the time spent by the CPU and the number of calls when running the unit tests.
3. Profiling data allowed us to identify framework segments meeting the following criteria:
    a. Their single invocation does not take long, but the aggregate time spent on all their calls is distinguishingly long compared to the call time of other methods invoked during the application's runtime;
    b. The time required by the processor to execute this code segment is longer than the time required to perform other functions called in each particular test;
    c. They have optimization potential, which does not affect the algorithm outcome.
4. We drew up a dependency table for the libraries used (Table 1) with regard to the potential for parallel and distributed MPD Root technologies. It included ROOT and FairRoot dependencies.

Technologies				Is included as a dependency		
Library	**Multicore CPU**	**GPGPU**	**Cluster**	**ROOT**	**Fair Root**	**Mpd Root**
MesaOpenGL	OpenCL	CUDA, OpenCL		+	+	+
FFTW	OpenMP		MPI	+		+
GSL	OpenMP	CUDA	MPI	+	+	+
PLUTO	OpenMP		MPI	+		+
GEANT3	OpenMP				+	+
GEANT4	Intel TBB, OpenCL	CUDA, OpenCL	MPI	+	+	+
Boost	OpenMP, OpenCL, Intel TBB (Thrust)	OpenCL,CUDA (Thrust)	MPI		+	+

The contents of the table include:

- MesaOpenGL is a graphics library. It is used within EVE (EventVisualizationEnvironment) to visualize modelling results and event reconstruction.
- FFTW library is used in the digitizing algorithm of the TPC detector in TVirtualFFT class, a Fourier transformation shell in the ROOT framework. FFTW library provides interfaces for OpenMP and MPI, but it does not support graphic co-processors.
- GNU Scientific Library provides mathematical functions, such as: solving differential equations, linear algebra algorithms, and others.
- PLUTO is a simulation framework for reactions in nuclear physics.
- GEANT is a tool simulating the passage of particles through matter.

Boost is a collection of general-purpose libraries used by other MPD Root dependencies.

Proposed optimizations

We analyzed the results of MPD Root package macros profiling and reviewed the source code of the methods used. This allowed us to select several segments of the framework that fit the aforementioned criteria and can be expedited:

- STL functions;
- Runge-Kutta method for solving ordinary differential equations;

The conclusions on optimization capacity are based on the results of studies published over the past few years, and on the multi-core data processing capabilities in the package algorithms.

Optimization of dependencies

Co-processors allow to considerably optimize the systems using massive parallel processing [Holmen, Humphrey, Berzins, 2015; Bogdanov, Degtyaryov, Khramushin, 2014; Iakushkin, Degtyarev, Shvemberger, 2014; Abrahamyan, Balyan, …, 2016]. Co-processors are supported by the OpenCL open standard implemented in NVIDIA, AMD, and Intel devices [Shichkina, Degtyarev, …, 2016; Iakushkin, 2015]. There are a number of open source libraries which use OpenCL. clFFT library, for instance, is designed to work with Fourier Transforms. GSL-CL provides an alternative to

GSL for general mathematical calculations, while Boost.Compute library may be used to implement internal project algorithms.

STL

Macro/mpd test profiling showed frequent use of standard library functionality, including the search feature. For example, FairMCApplication::Stepping() function of runMC macros calls find() function over 5 million times. This number exceeds the calls of Stepping() function itself by 25 times.

The new STL C++ standard provides data processing tools that make it possible to perform the search function using parallel computing units. In addition, it includes patterns frequently used in parallel treatment of vectors: transform, reduce, scan, foreach, etc. Thus, the move to the current STL standard will allow for parallel processing of vectorized data without specialized libraries.

Runge-Kutta Methods

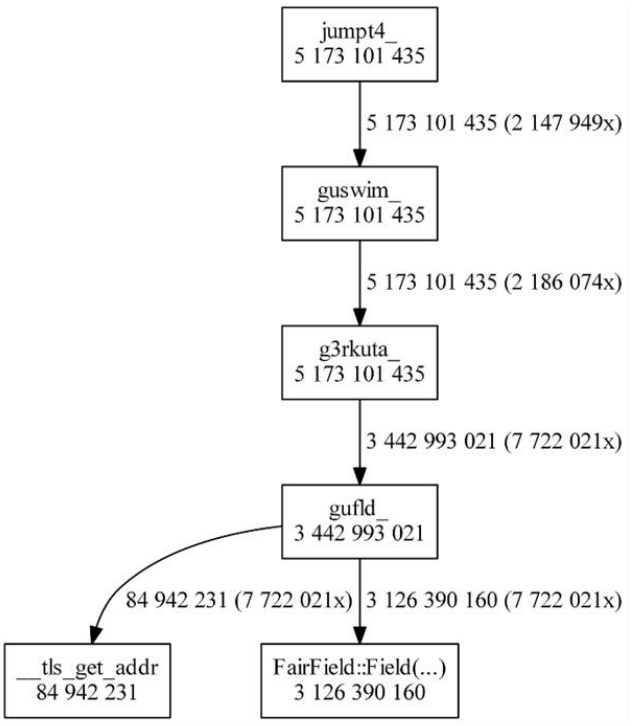

Figure 1. Profiling data—Runge-Kutta methods

Figure 1 contains a fragment of profiling data for the RunMC test. It shows that the Runge-Kutta method for solving ordinary differential equations was invoked over 2 million times. Furthermore, it is one of the most processor-intensive functions.

[Al-Turany, Uhlig, 2010] provided an example of Runge-Kutta methods optimization using CUDA technology. The paper was presented at the ACAT (Advanced Computing and Analysis Techniques) Conference in 2010 and described the method's optimization for the PANDA@FAIR (Anti-Proton Anihilation at Darmstadt, Facility for Antiproton and Ion Research) experiment. Runge-Kutta methods implemented in Geant3 are optimized both in MPD Root and in the PANDA@FAIR experiment. The tables published in [Al-Turany, Uhlig, 2010] show the effectiveness of this approach, which allows for more than threefold acceleration, depending on the graphics processor and the volume of input data.

Conclusion

The paper lays out MPD Root framework optimization criteria. It shows an algorithm to identify optimizable code segments. We give a number of recommendations on modifying certain methods of the package in terms of parallel and distributed programming on multi-core CPUs and GPUs. The next steps may include development of prototypes that allow load testing of the proposed MPD Root optimizations using the equipment of the JINR LIT heterogeneous computing cluster.

References

Iakushkin O.O., Degtyarev A.B., Shvemberger S.V. Decomposition of the modeling task of some objects of archeological research for processing in a distributed computer system // Computer Research and Modeling. — 2014. — Vol. 7, No. 3. — P. 533 537.

Al-Turany M., Uhlig F. Applying CUDA computing model to event reconstruction software. — PoS, 2010. — P. 014.

Богданов А.В., Дегтярев А.Б., Храмушин В.Н. Высокопроизводительные вычисления на гибридных системах: будут ли решены «задачи большого вызова» // Компьютерные исследования и моделирование. — 2015. — Том 7, № 3. — С. 429 438.

Bogdanov A.V., Degtyarev A.B., Khramushin V.N. High performance computations on hybrid systems: will "grand challenges" be solved? // Computer Research and Modeling. — 2015. — Vol. 7, No. 3. — P. 429 438 (in Russian).

Holmen J., Humphrey A., Berzins M. High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches. // Elsevier Inc. — 2015.

Iakushkin O. Intellectual scaling in a distributed cloud application architecture: A message classification algorithm // In "Stability and Control Processes" in Memory of V. I. Zubov (SCP), 2015 International Conference. — IEEE, 2015. — P. 634 637.

Abrahamyan S., Balyan S., Muradov A., Korkhov V., Moskvicheva A., Jakushkin O. Development of M-Health Software for People with Disabilities // International Conference on Computational Science and Its Applications. — Springer International Publishing, 2016. — P. 468 479.

Shichkina Yu., Degtyarev A., Gushchanskiy D., Iakushkin O. Application of Optimization of Parallel Algorithms to Queries in Relational Databases // International Conference on Computational Science and Its Applications. — Springer International Publishing, 2016. — P. 366 378.