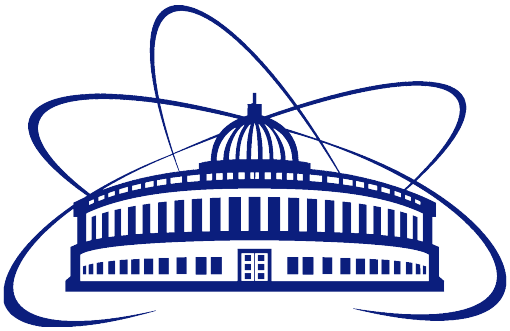

JOINT INSTITUTE FOR NUCLEAR RESEARCH
Veksler and Baldin Laboratory of High Energy Physics

FINAL REPORT ON THE SUMMER STUDENT PROGRAM

Implementation of vectorization in the BmnRoot software package

Supervisor : Sergei P. Merts
Student : Anastasiia A. Iufriakova
Russia, Saint Petersburg State University
Participation period: 31 January - 12 March

Dubna, 2022

Abstract

The BM@N (Baryonic Matter at the Nuclotron) is one of the experiments at the NICA (Nuclotron-based Ion Collider fAcility) accelerator complex. The BmnRoot framework is developed and regularly upgraded to simulate and reconstruct events, to process the data obtained from the detectors of the experiment. This software package must be constantly modified and optimized to meet the needs of the experiment. In this regard, the performance of the BmnRoot was analyzed, the most resource-intensive parts of the code were identified and the decoding module was vectorized.

Contents

Introduction

The mega-science NICA Project (fig.1.1) is being created on the basis of the Joint Institute for Nuclear Research (Dubna, Russia) to study the properties of baryonic matter under extreme laboratory conditions [1]. The enormous amount of data obtained during the experiments of this complex needs to be processed and analyzed.

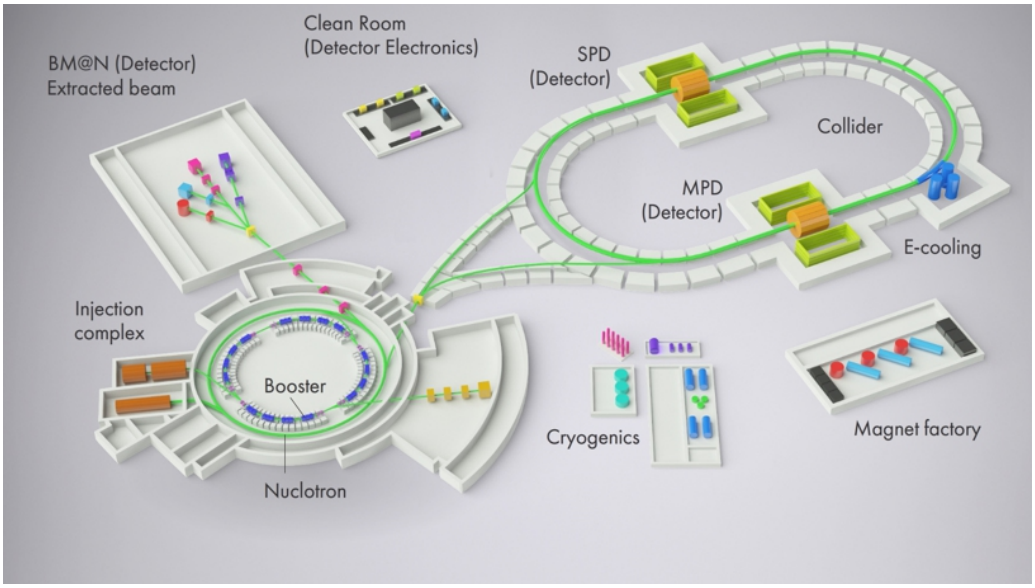

Figure 1.1: The scheme of the NICA accelerator complex

BM@N is the first experiment at the NICA accelerator complex, where physical launches with a collection of experimental data were carried out [2]. At the moment, the research group, which consists of professors, students and graduate students of the Department of Computational Physics at the Faculty of Physics, St. Petersburg State University,

is a part of the BM@N collaboration. One part of this group's work is a modification and optimization of the BmnRoot framework. This software is created to model, reconstruct, and analyze the data obtained from the detectors of the experiment. The task of efficient processing of large amount of data is one of the key tasks for successful realization of the experiment. In order to increase efficiency, it is necessary to consider all the steps: from problem statement to the choice of numerical method and algorithm, taking into account the peculiarities of programming language and compiler. One of the options for code optimization is vectorization, which can be used to achieve significant speedup of the framework.

Vectorization

In the 1970s, Michael Flynn proposed the classification of computer architectures based upon the presence of the parallelism in command and data streams:

• Single instruction, single data stream (SISD) — a single instruction stream on scalar data.

• Single instruction, multiple data streams (SIMD) — multiple data streams against a single instruction stream to perform operations which may be naturally parallelised.

• Multiple instruction, single data streams (MISD) — many processors in the architecture processing the same data stream.

• Multiple instruction, multiple data (MIMD) — many functional units perform different operations on different data [3].

SIMD computing is also known as vector processing, because the basic data unit of SIMD is a vector. A vector is a row of individual numbers or scalars. A regular CPU operates on scalars, one at a time. A vector processor, on the other hand, lines up a whole row of these scalars, all of the same type, and operates on them as an unit.

Vectors are usually represented in packed data format. Data is grouped into bytes (8 bits) or words (16 bits) and packed into a vector to be operated on. The vector size defining the number of scalars processed in parallel is one of the most critical design aspects of SIMD implementations. For instance, using a 4-element, 128-bit vector one can do four-way single-precision (32-bit) foating-point calculations in parallel.

SIMD extensions were introduced in the x86 architecture to increase the processing speed of streaming data. These extensions are called SSE (Streaming SIMD Extensions) [5]. SSE instructions are the same as

pure assembler instructions and programming languages with any higher abstraction level can not handle them directly. However, there is no need to code in pure assembly language to use them. The compiler developers have already made built-in wrappers, which are called intrinsics.

Analysis of the decoding module in BmnRoot

1.1 Intel® VTune™ Profiler

Intel® VTune™ Profiler is a performance optimization and application profiling tool. This software is a part of the Intel® oneAPI Base Toolkit [4]. It identifies the sections of code of the analyzed application which consume the largest amount of resources, so called hotspots. As a result, the profiler generates various characteristics of the application including the execution time of each module. This is an essential information for further optimization problematic code sections.

1.2 Identifying decoding module's hotspots

The events decoding program of the BmnRoot framework was analyzed in order to find the most resource-intensive places in the code. The decoder was run under the profiler with certain initial parameters. The number of events is equal to 10000. The processed data are experimental data from one of the earlier sessions of the BM@N experiment (argon beams). BmnRoot was built in release mode.

Function / Call Stack	CPU Time ▼	Module
BmnAdcProcessor::CalcEventMods	47.065s	libDecoder.so.0.0.0
BmnAdcProcessor::PrecalcEventMods	12.527s	libDecoder.so.0.0.0
__read	12.094s	libc.so.6
BmnGemRaw2Digit::ProcessAdc	8.779s	libDecoder.so.0.0.0
func@0x18e5d4	5.198s	libc.so.6
__write	4.632s	libc.so.6
func@0x18650	4.622s	libpq.so.5
operator new	4.574s	libstdc++.so.6
deflate	4.000s	libz.so.1
TBufferFile::ReadFastArray	2.896s	libRIO.so
frombuf	2.821s	libRIO.so
BmnSiliconRaw2Digit::ProcessAdc	2.752s	libDecoder.so.0.0.0
func@0x999c0	2.492s	libc.so.6
BmnAdcProcessor::RecalculatePedestalsAugmer	2.248s	libDecoder.so.0.0.0

Figure 1.2: List of the first decoder's hotspots

The most resource-intensive parts of the program are functions CalcEventMods and PrecalcEventMods from the class BmnAdcProcessor. This class is responsible for digitization the data obtained from the detectors and it is used in the monitoring of events during sessions, which must work online. So, it is very important to process the data from the detectors as fast as possible. The first hotspot takes approximately 29 percent of the execution time of the event decoder and the second takes about 8 percent of this time. These functions contain arrays and mathematical operations, that is why they were chosen for further optimization.

ADC strip decoder vector optimization

The parallelisation scheme of the decoder is simple: corresponding parameters of four or eight signals from the channels are packed into a vector. Previously, fitter track vectorization was already implemented in the framework [6]. In order to make the code readable, overloading of basic operators was written. The implementation of this class of overloads in the decoder was the first step in vector optimization.

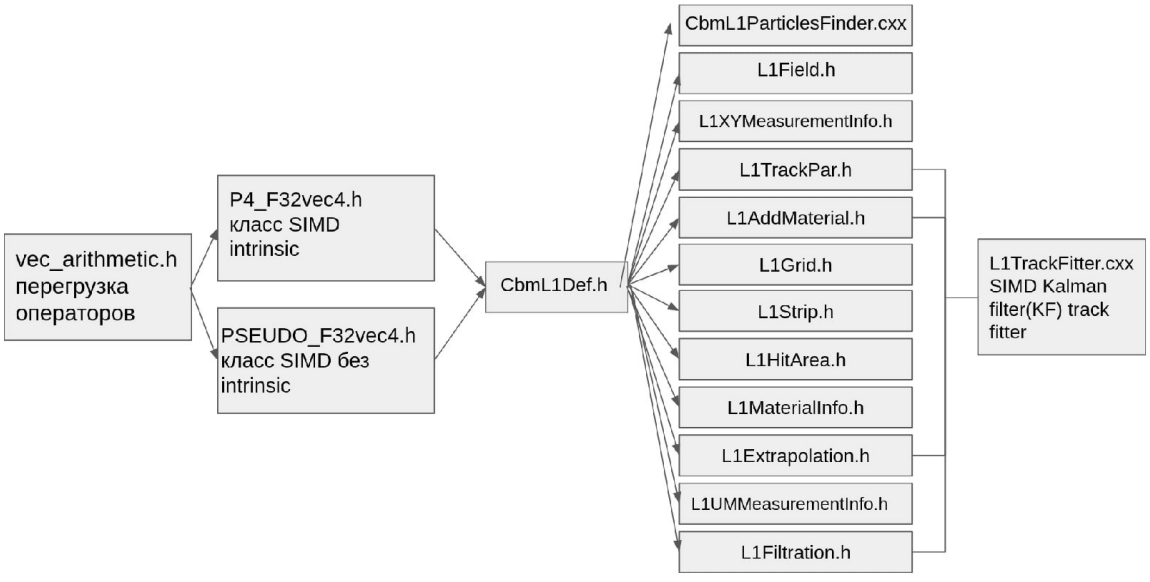

Figure 1.3: Vectorization scheme of the track fitter

1.3 SSE vectorization

To be able to use SSE instructions all the arrays of double, bool and integer types from the class BmnAdcProcessor were replaced with arrays of float type. Then the code was vectorized using the class of operator overloads. Corresponding signals from the channels on the chips were packed into a vector of four. The change in the code did not affected the results and their accuracy, but the time taken for the function CalcEventMods of the BmnAdcProcessor class has been reduced by half. The vectorization of the function PrecalcEventMods reduced its execution time by a factor 1.25.

Next, the decoder was analyzed again and the following results were obtained:

Function / Call Stack	CPU Time ▼	Module
BmnAdcProcessor::CalcEventMods_simd	18.422s	libDecoder.so.0.0.0
__read	12.991s	libc.so.6
BmnAdcProcessor::PrecalcEventMods_simd	8.572s	libDecoder.so.0.0.0
BmnGemRaw2Digit::ProcessAdc	8.490s	libDecoder.so.0.0.0
operator new	7.913s	libstdc++.so.6
func@0x18e5d4	5.601s	libc.so.6
func@0x18650	4.876s	libpq.so.5
__write	4.638s	libc.so.6
func@0x999c0	4.386s	libc.so.6
deflate	4.226s	libz.so.1
BmnSiliconRaw2Digit::ProcessAdc	2.938s	libDecoder.so.0.0.0
TBufferFile::ReadFastArray	2.850s	libRIO.so
frombuf	2.784s	libRIO.so
BmnAdcProcessor::RecalculatePedestalsAugr	2.210s	libDecoder.so.0.0.0
fread	1.992s	libc.so.6

Figure 1.4: List of the first decoder's hotspots after SSE vectorization

As a result, 15 percent speedup of the decoder program was achieved by implementation of SSE vectorization.

1.4 AVX vectorization

Today, it is possible to pack eight values into a single vector. AVX (Advanced Vector Extensions) expands most commands to 256 bits and introduces new instructions [5]. The existing wrapper for overloading operators was improved with AVX instructions. Then, this modified class was connected to the decoder. Therefore, even greater acceleration could be achieved.

Function / Call Stack	CPU Time ▼	Module
__read	12.588s	libc.so.6
BmnAdcProcessor::CalcEventMods_simd	11.606s	libDecoder.so.0.0.0
BmnAdcProcessor::PrecalcEventMods_simd	8.673s	libDecoder.so.0.0.0
BmnGemRaw2Digit::ProcessAdc	6.361s	libDecoder.so.0.0.0
func@0x18b554	5.236s	libc.so.6
__write	4.444s	libc.so.6
operator new	3.923s	libstdc++.so.6
frombuf	3.142s	libRIO.so
TBufferFile::ReadFastArray	3.020s	libRIO.so
BmnSiliconRaw2Digit::ProcessAdc	2.846s	libDecoder.so.0.0.0
func@0x96870	2.614s	libc.so.6
BmnAdcProcessor::RecalculatePedestalsAugmented	2.560s	libDecoder.so.0.0.0
TStreamerInfo::ReadBuffer<char**>	2.539s	libRIO.so
TBufferFile::WriteFastArray	1.738s	libRIO.so

Figure 1.5: List of the first decoder's hotspots after AVX vectorization

The function CalcEventMods now takes only 8 percent of the execution time of the decoding program and the function PrecalcEventMods takes about 6 percent, although in the original code, these functions together took up to 37 percent of the execution time. The acceleration of the entire decodig program is more than 20 percent.

Summary

During the student practice, important tasks were completed, necessary for the physical analysis. With the Intel Vtune Profiler program the code of the BmnRoot software package was analyzed. The most resource-intensive parts of the BmnRoot software package were identified.The BmnAdcProcessor module of BmnRoot software package was vectorized.The efficiency of the event decoding program increased in release mode by 20 percent on average. The modified code was tested for correctness with positive results.

Bibliography

[1] Official website of the NICA project [Electronic resource] URL: http://nica.jinr.ru/ru (Accessed 23.03.2022).

[2] Official website of the BM@N experiment [Electronic resource] URL: https://bmn.jinr.ru/ (Accessed 23.03.2022)

[3] Flynn M. J. Some computer organizations and their effectiveness // IEEE Transactions on Computers, 1972, 21 (9). — P. 948—960.

[4] Official Intel oneAPI Toolkits website [Electronic resource] URL: https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html (Accessed 23.03.2022)

[5] Intel Intrinsics Guide [Electronic resource] URL: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html (Accessed 23.03.2022)

[6] S. Gorbunov, U. Kebschull, I. Kisel, V. Lindenstruth, and W.F.J. Mueller, Fast SIMDized Kalman Filter based track Fit, Comp. Phys. Comm., vol. 178, no. 5, pp. 374 - 383, Mar. 2008.