Vectorization of CMSSW offline software

Patrick Gartung[1]*

[1]Fermi National Accelerator Laboratory, Batavia, IL, US

Abstract. The CMS experiment has been utilizing vectorization, or SIMD, in parts of its data processing applications for over a decade. On x86 platforms the vectorization level is still SSE3. In the past attempts to use wider vector instruction sets such as AVX or AVX-512 have, in practice, not resulted in improvements in the overall event processing throughput, because the CPUs scale down their frequency when processing AVX instructions. In addition, a notable part of the global pool of CMS resources has been old systems either not supporting AVX, or where the CPU frequency downscaling impacts all cores of the CPU. CMS has nevertheless continued to vectorize more of its application code, and in this work we review profiling methods we have found effective to find out pieces of code that would benefit from vectorization, and techniques to transform those codes such that the GCC compiler is able to auto-vectorize those codes. The build system used for CMSSW, Scram, has also been enhanced to be able to build code for multiple CPU microarchitectures such that the shared libraries of desired microarchitecture level can be loaded based on the CPU of the system. This multi-microarchitecture setup is invisible to the workflow management system, which makes its deployment straightforward. We describe in detail how this multi-microarchitecture build is set up, and measure the impact of using wider vector units than SSE3 on the event processing throughput of CMS applications such as simulation and reconstruction on recent x86 CPUs

1. Profiling CMSSW with Intel's OneAPI Toolset

The CMSSW [1] application uses Intel's Thread Building Blocks (TBB) library to enable the use of multiple threads in it's framework. Intel's OneAPI Toolset [2] include a number of libraries and applications that can assist in optimizing software to use SIMD instructions. Intel Vtune [3] is used to profile the CMSSW application. Intel Vtune is the only profiler that stitches together TBB thread stack frames. Intel Advisor [4] is used to identify scalar loops with floating point operations that can potentially be replaced by SIMD instructions. Intel's C++ Compiler Classic (ICC) [5] and small vector math library (SVML) [6] can produce more vectorization with the SSE3 instruction set but are not physics validated for use in CMSSW. Intel Advisor GUI offers a Vectorization Advisor pane, a Threading Advisor pane, a Survey pane with Source view of a floating point loop with timing, and a Survey pane with Assembly view of a floating point loop with timing.

2. Methods used to auto-vectorize with SSE3 instructions

* Corresponding author: gartung@fnal.gov

Using Intel Advisor and Vtune, loops using floating point operations were identified in CMSSW application code. Modifications were made to these loops to enable the GCC compiler [6] to automatically use SIMD instructions, auto-vectorization, with the use of the SSE3 instruction set. The GCC compiler diagnostics for auto-vectorization were used to count the number of loops that were vectorized with each change. Performance was measured with Intel Vtune, before and after changes, to ensure that changes reduced time spent in floating point loops. Some of the techniques used to achieve this included the following:

- Using temporary arrays to store loop calculation results for use in later loops in a function.
- Splitting a loop with many calculations into many smaller loops for each calculation.
- Putting functions that could not be inlined in their own loop.
- Putting functions that could be inlined in their own loop and using GCC pragma *__strict__* to tell the compiler that the values of function inputs would not be changed in the function.

3. CMSSW multi microarchitecture releases

For production, CMSSW is optimized for the SSE3 instruction set, the lowest common instruction set for the mix of AMD [4] and Intel [5] CPU available on the LHC computing grid [6]. Optimizing for higher instruction sets and vector widths enables more loops to be vectorized, potentially increasing performance. The CMSSW build system was enhanced to produce three sets of libraries. These are compiled with the GCC *-march* flags *sse3*, *haswell* and *skylake-avx512*. The library set is chosen by an environment variable set by the CMSSW build configuration tool SCRAM [7] .

SCRAM uses the microarchitecture of the CPU on the computer where the application is run to set it. SCRAM then uses the environment variable to set LD_LIBRARY_PATH and PATH to the directories with library set compiled for that microarchitecture. This environment variable can be overridden to allow the use of lower vector width instruction sets for performance comparison. The size of a multi microarchitecture release is at least three to four times the size of a single architecture release.

4. CMSSW multi microarchitecture performance

A multi architecture release was build and installed on CVMFS. This release was used to run the CMSSW application on a desktop computer with an 11th Gen Intel Core i7 11700 @ 2.50 GHZ [8] with boost clocks up to 4.9 GHZ. This CPU features 8 cores was with two threads per core and the avx-512 microarchitecture. The physics event generation and simulation of particle in the detector (GEN-SIM), digitization of simulated detector hits and high level trigger (DIGI-HLT) and reconstruction of particles (RECO) processes of the CMSSW application were run using the conditions and detector geometry of the CMS detector during Run 3 operations. The number of processes and threads per process were set to occupy 8 (50%) and 16 (100%) of the available hyper threads on the CPU. The average throughput (events per second) per thread was calculated by dividing the total number of events processed by the total cpu time of all processes. The average throughput for each CMSSW process and micro-architecture is shown below for the 50% loaded node in Table 1 and 100% loaded node in Table 2.

Table 1. Throughput on 50% loaded node

Micro-architecture	GEN-SIM	DIGI-HLT	RECO
default	0.079 events per second per thread	0.087 events per second per thread	0.082 events per second per thread
haswell	0.090 events per second per thread	0.099 events per second per thread	0.077 events per second per thread
skylake-avx512	0.080 events per second per thread	0.087 events per second per thread	0.078 events per second per thread

Table 2. Throughput on 100% loaded node

Micro-architecture	GEN-SIM	DIGI-HLT	RECO
default	0.056 events per second per thread	0.053 events per second per thread	0.045 events per second per thread
haswell	0.051 events per second per thread	0.053 events per second per thread	0.053 events per second per thread
skylake-avx512	0.052 events second per thread	0.052 events per second per thread	0.052 events per second per thread

The use of higher vector widths can cause the boost clock to be lowered for thermal management[9]. For the 50% loaded RECO process, the throughput is lower for higher vector widths. This indicates that the boost clock was used but lowered. For the 100% loaded RECO process, the throughput is higher for higher vector widths. This indicates that the boost clock was not used and the improvement from higher vector widths can be seen.

5. Discussion and conclusions

Only very select algorithms in the CMSSW application have been written explicitly to auto-vectorize. CMS [12] has pursued research and development into high level trigger algorithms on GPU written in CUDA. The Alpaka [10] library was selected in order to make the HLT algorithms portable across GPU vendors. The Alpaka library also offers SIMD implementations. As more CMSSW algorithms are ported to Alpaka, multi microarchitecture builds may offer more benefits. The benefits of micro-architecture releases are minimal compared to the added cost of higher storage requirements. Therefor CMS continues to deploy builds optimized to the lowest common denominator micro-architecture, SSE3, to the grid. Discussions within CMS have recently begun to deploy builds optimized to the x86_64-v2 [11] architecture by default with fallback to builds optimized to SSE3 architecture.

This document was prepared by the CMS Collaboration using the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359.

References

1. http://cms-sw.github.io, *CMS Offline Software,* Accessed 1 May 2023
2. https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html#gs.3u6x9i, *oneAPI: A New Era of Heterogeneous Computing,* Accessed 1 May 2023
3. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html#gs.3u6tna, *Intel VTune Profiler,* Accessed 1 May 2023
4. https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html#gs.3u707m, *Intel Advisor,* Accessed 1 May 2023
5. https://www.intel.com/content/www/us/en/docs/cpp-compiler/developer-guide-reference/2021-10/overview.html, *Intel C++ Compiler Classic Developer Guide and Reference,* Accessed 1 May 2023
6. https://www.intel.com/content/www/us/en/docs/cpp-compiler/developer-guide-reference/2021-10/intrinsics-for-short-vector-math-library-ops.html, *Intrinsics for Short Vector Math Library Operations,* Accessed 1 May 2023
7. https://gcc.gnu.org, *GCC, the GNU Compiler Collection,* Accessed 1 May 2023
8. https://www.amd.com/en/processors, *AMD Processors,* Accessed 1 May 2023
9. https://www.intel.com/content/www/us/en/products/details/processors.html, *Intel Processors and Microprocessors for All That You Do,* Accessed 1 May 2023
10. https://en.wikipedia.org/wiki/Worldwide_LHC_Computing_Grid, *Wordwide LHC Computing Grid,* Accessed 1 May 2023
11. https://scram.readthedocs.io, *SCRAM documentation,* Accessed 1 May 2023
12. https://ark.intel.com/content/www/us/en/ark/products/212279/intel-core-i711700-processor-16m-cache-up-to-4-90-ghz.html, *Intel Core i7-11700 Processor,* Accessed 1 May 2023

13. S. R. Lantz, https://cvw.cac.cornell.edu/vector/performance/performance-turbo, *Turbo Boost,* Accessed 1 May 2023
14. https://github.com/alpaka-group/alpaka, *Abstraction Library for Parallel Kernel Acceleration,* Accessed 1 May 2023
15. https://en.wikipedia.org/wiki/X86-64#cite_note-47, *x86_64,* Accessed 1 May 2023
16. https://arxiv.org/abs/2309.05466, *Development of the CMS detector for CERN LHC Run 3,* Accessed 1 May 2023