A GPU offloading mechanism for LHCb

Alexey Badalov¹, Daniel Hugo Campora Perez², Alexander Zvyagin², Niko Neufeld, Xavier Vilasis Cardona¹
E-mail: alexey.badalov@cern.ch, dcampora@cern.ch

Abstract. The current computational infrastructure at LHCb is designed for sequential execution. It is possible to make use of modern multi-core machines by using multi-threaded algorithms and running multiple instances in parallel, but there is no way to make efficient use of specialized massively parallel hardware, such as graphical processing units and Intel Xeon/Phi. We extend the current infrastructure with an out-of-process computational server able to gather data from multiple instances and process them in large batches.

1. Introduction
LHCb is the LHC experiment dedicated to studying B-physics. Its primary goal is to search for indirect evidence of new physics in CP violation and rare beauty hadron decays [1]. From a computational perspective, the data produced from the experiment subdetectors is gathered and processed with a real-time system, referred to as Online, where the data rate is reduced from the bunch-crossing rate of 40 MHz to a manageable 2 kHz (or 100 MB/s) by using hardware and software selection triggers based on discarding uninteresting data. The system runs on a computer farm consisting of roughly 16,000 physical CPU cores. After the Long Shutdown 2 in 2018, the LHC accelerator is set to receive an upgrade and will increase collision energy and luminosity. In LHCb, the mixed trigger is expected to be replaced by a full software trigger [2]. As a consequence of these two factors, the amount of data needed to be processed in software will increase from tens of gigabytes to terabytes. This steep increase is our motivation for searching for new ways of processing the computation load.

The entire software infrastructure at LHCb is built around a flexible, extensible, modular framework called Gaudi [3]. This framework was conceived at the turn of the century for a computational environment, where a large number of powerful CPU cores handled a stream of data, with each core taking a piece of data from the stream and applying to it a sequence of algorithms.

A different computational paradigm has been gaining in popularity, spurred by the success of consumer 3D graphics accelerators. There is now readily available hardware for massively parallel computation, such as consumer graphics processing units, NVIDIA Tesla cards, and Intel Xeon/Phi coprocessors. Enabling the use of such hardware in the Gaudi framework has the potential to cope with increased load and enhance algorithms used in HEP experiments.

The current Gaudi framework architecture presents an obstacle to the use of massively parallel hardware. Its benefits are best realized when processing events in large batches, not one by one in independent concurrent pipelines. We contribute a natural and generic way of extending Gaudi with

¹ La Salle — Ramon Llull University
² CERN

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd
the capability of taking advantage of this hardware. The extension allows addition of new massively parallel algorithms to existing pipelines.

We have adapted the Pixel VELO pattern recognition algorithm to use the new extension. In this paper, we describe the Gaudi framework, our extension, and the modifications that were required to adopt the algorithm.

2. LHCb Gaudi architecture

Gaudi is a general data processing framework, which provides interfaces and services for HEP experiments. It is used in the data collection, processing, and analysis chain, serving to separate data from algorithms, and helping to structure computation. Pipelines are constructed by chaining together individual algorithms. Each algorithm takes data from the Transient Event Store (TES), processes it using libraries, services, and tools provided by Gaudi, and then places the output into TES.

An algorithm running on Gaudi can choose to take advantage of multiple processor cores independently of the framework, but it is limited to processing only a single piece of data at a time. It is possible to run several instances of the framework in parallel. In this case, each instance applies the same algorithms to different data.

Massively parallel hardware has been shown to achieve significant gains in efficiency by utilizing vectorization and single-instruction, multiple-data (SIMD) architectures in HEP experiments [4]. It is best applied to large datasets and especially to multiple instances of the same computation. Given the small size of individual data sets at LHCb (about 60 KB raw event size), an individual algorithm running on Gaudi cannot properly take advantage of massively parallel software. This is why we need a mechanism for computing data in batches outside of the pipeline.

3. Offloading mechanism

We address Gaudi’s limitations by taking massively parallel computations outside of the pipeline. Our approach is to create a client-server interface, in which Gaudi pipelines act as clients to a server managing the piece of hardware.

We tie each massively parallel computation unit to a server process. This process receives data from multiple Gaudi instances, processes them in batches, and then distributes the results back to senders. The client instances can be located on the same node as the server process or communicate over the network.

Gaudi pipeline instances communicate with the server through a Gaudi service. A Gaudi service is a special kind of library that can be accessed from within the pipeline. This service takes data in a GPU-ready format, sends it to the server process, and then returns the result when it is ready. Client-server communication is hidden from the algorithm implementer. Different communication strategies could be used without affecting the algorithms.

Figure 1 compares the data flow in the original Gaudi framework with our offload mechanism. In the original, multiple instances of an algorithm are created for multiple concurrent pipelines, each processing its own piece of data. In the offload scenario, only one instance of the algorithm per hardware device is created, and fed aggregated data from multiple pipelines.
4. VELO Pixel GPU algorithm
The Pixel VELO is an upgrade for the VELO subdetector, which will replace the current VELO during Long Shutdown 2 [2]. The particle reconstruction in the Pixel VELO consists in finding all good tracks given a set of reconstructed clusters from the subdetector. Currently, a sequential implementation within the Gaudi framework exists [5], which performs a Track Forwarding local method [6] to select prospective tracks.

The Pixel VELO sequential algorithm can be divided in three sections: prepare, searchByPair and storeTracks. Amdahl's Law [7] predicts a theoretical maximum limit to the speedup obtainable by an algorithm running on N cores compared to running on a single core, bound to the percentage of code which is parallelizable P in the studied algorithm. The maximum speedup obtainable is bound to the parallelizable section of the algorithm. In our case, searchByPair takes 78% of the current execution time, and therefore the maximum speedup obtainable in the algorithm is 4.2.

\[
S(N) = \frac{1}{(1 - P) + P/N}
\]

Amdahl's Law.

We have created an algorithm from scratch following a local search strategy, with parallelism in mind. Track seeding is done by searching clusters in triplets as shown in figure 2, spreading the processing of triplets of sensors across blocks for an NVIDIA accelerator. The best seeds are forwarded using a similar local method to the one employed by David Rohr [4], fitting additional clusters based on a least squares fit, until all sensors are exhausted in both directions. An optional final Track Selection is applied, following a criterion based on fit minimization and length maximization [8].
In order to accustom the offloaded algorithm to our Gaudi Offloading engine, a Gaudi Algorithm is required, divided in three logical parts. A prepare stage takes the cluster data from the TES, which is in an Array-Of-Structures format (AoS). A conversion into a many-core friendly Structure-Of-Arrays (SoA) is performed. Later, in a kernel_offload stage, the data is forwarded to the offload service, which passes the data to the GPU server. At this stage, an active wait is required on the client side, until the server performs the computation. A final store stage converts the data back to a Gaudi-friendly data format, and reinserted in the Gaudi chain.

Our algorithm has a Reconstruction Efficiency of 75% compared to the sequential version, as tested with a Monte Carlo dataset from the April 2013 upgrade. This indicates that more tuning is required. Even processing one event at a time, we observe an 11-fold speedup, which shows the potential behind GPU processing for LHCb subdetectors.

5. Conclusions and future work
The Gaudi framework was created at the turn of the century, and its design did not foresee the current situation of widespread use of massively parallel hardware. We have developed an offloading mechanism that remedies this oversight and ported the Pixel VELO event detection algorithm to use it.

In order to hide the transmission overhead of the engine and prevent the client side from stalling upon completion of the execution by the server side, several strategies are being considered. GaudiMT, a multi-threaded version of Gaudi, is under development, scheduled to release by the end of 2013. The Offload engine could be merged into the GaudiMT environment, prospectively hiding the communication with the server.

A Server Scheduler is under development in order to account for a multi-event multi-algorithm execution on the server side. Other transmission mechanisms, like memory sharing for local or GPUDirect for remote are in consideration for future releases of the engine.

We intend to use the parallel Pixel VELO implementation as a benchmark with GaudiMT, to compare the GPU performance compared to a multithreaded performance on the CPU and measure the offloading overhead introduced in a parallel environment of Gaudi.

References
[1] Navarro A P and Markus F 2010 Event reconstruction in the LHCb online cluster Journal of Physics: Conference Series 219
[2] Collaboration L 2011 Letter of intent for the LHCb upgrade CERN
[3] Barrand G, Belyaev I, Binko P, Cattaneo M, Chytracek R, Corti G, Frank M, Gracia G, Harvey J and Herwijnen E 2001 GAUDI - a software architecture and framework for building HEP data processing applications 140 45-55
[4] Rohr D 2012 ALICE TPC Online tracking on GPU based on Kalman filter Diploma Thesis
[5] Callot O Rec / Tf / PatPixel [Online] http://svn.cern.ch/guest/lhcb/Rec/trunk/Tf/PatPixel/
[6] Callot O 2012 Fast pixel pattern recognition
[7] Amdahl G 1967 Validity of the single processor approach to achieving large-scale computing capabilities 30 483–485
[8] Cámpora Pérez D, Sancho Caparrini F, Martinez del Amor M and Neufeld N 2013 A study of a parallel implementation for the pixel VELO subdetector