Particle Tracking on FPGAs using oneAPI

Karol Hennessy, Kurt Rinnert

***Abstract*—The next era of LHC experiments will provide an unprecedented volume of data, aiming to achieve a tenfold increase in integrated luminosity. Processing these data presents formidable computing challenges. In the case of the LHCb detector, a fully software based trigger has been employed in its current design, which processes events at ∼30 MHz. Currently, GPU-based compute acceleration is harnessed to manage the high track densities within the detector. The computing challenge intensifies with HL-LHC hit multiplicities, leading to extremely large combinatorics in forming particle tracks from hits. To tackle this challenge, a novel tracking algorithm, *TrackNN*, employing machine learning, has been devised to generate track stubs for early particle track reconstruction. This algorithm is tailored for deployment on an FPGA, leveraging parallel streams of pipelined neural nets to optimize bandwidth and resource utilization. Central to this approach is the integration of Intel's oneAPI framework. OneAPI enables algorithm development through high-level C++ coding whilst allowing integration with conventional RTL designs. Early profiling tools identify resource estimates and bottlenecks in minutes rather than the hours typical of traditional FPGA development cycles. The machine learning algorithm and oneAPI development cycle will be presented, sharing some early performance measurements using LHCb Vertex Locator data.**

***Index Terms*—Firmware, DAQ, Readout, HLS, oneAPI, FPGA, HL-LHC**

I. Introduction

FIELD Programmable Gate Arrays (FPGA) are used extensively in High-Energy Particle Physics experiments. They are particularly suited to data acquisition (DAQ) systems where they can process vast amounts of serialised detector data, and make real-time decisions on based on the data content. However, to date, developing algorithms for these devices has been time consuming and is typically the domain of electronic engineers rather than software developers. Conventional FPGA development flow requires the modelling a synchronous digital circuit (known as a register-transfer level model or RTL), using a hardware description language (HDL), such as VHDL[1] or Verilog. The process of converting FPGA logic design into gates is known as synthesis. Several attempts have been made to speed up FPGA development by using higher level languages to accelerate the process by creating a level of abstraction. This is commonly known as High Level Synthesis (HLS).

This work was supported by the Science and Technology Facilities Council of the UK. Karol Hennessy and Kurt Rinnert are with the Department of Physics, University of Liverpool, L69 7ZE, UK.

Special thanks to FAE Christian Faerber from Intel®/Altera®for regular expert assistance with the oneAPI framework

Corresponding author, Karol Hennessy (e-mail: karol.hennessy@cern.ch)
Corresponding author, Kurt Rinnert (e-mail: kurt.rinnert@cern.ch)

[1]VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

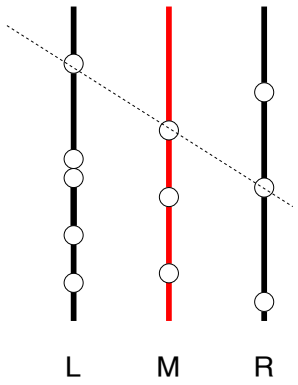

Fig. 1. Three detector planes: **L**eft **M**iddle and **R**ight are shown. One "triplet" combination is exactly one hit from each plane - an example is given by the dashed line. All such combinations are made as input to the *TrackNN* algorithm, which determines a probability for each.

A significant disadvantage of HLS was that most implementations were tied to a single hardware vendor. By contrast, conventional hardware description languages are open specifications and, by now, standardised and transferable. A new standard, SYCL[2] [1], has been developed to address this shortcoming. It has been designed for hardware accelerators (currently GPUs and FPGAs), and is based on C++. We have chosen the Intel® oneAPI implementation of SYCL. At the time of writing, this was determined to the most mature implementation for FPGAs.

The goal of the study described in the rest of this paper is two-fold: (1) to deploy a particle detector tracking algorithm to FPGA and (2) to evaluate the oneAPI in terms of productivity and accessibility for developers.

II. The TrackNN algorithm

The LHCb [2] Vertex Locator [3] was chosen as an test-case for the development of a general particle tracking algorithm. The algorithm assumes a series of detector planes with hits in x, y, z coordinates. Three adjacent detector planes are chosen and hit "triplets" are generated from all combinations of a hit from each detector plane (see Fig. 1). A neural-network algorithm, *TrackNN*, trained on LHCb VELO Monte-Carlo data samples, is used to determine the best "triplet" from the combinations. An earlier version of the algorithm is described in more detail in [4]. Full tracks are made by stitching triplets together to make full tracks. Tracking in LHC detectors is particularly challenging on CPU architectures due to the high number of particle hits generating large combinatorics. As such, highly parallel processing on GPUs and FPGAs is commonly employed to tackle this task. The final processing

[2]SYCL not an abbreviation

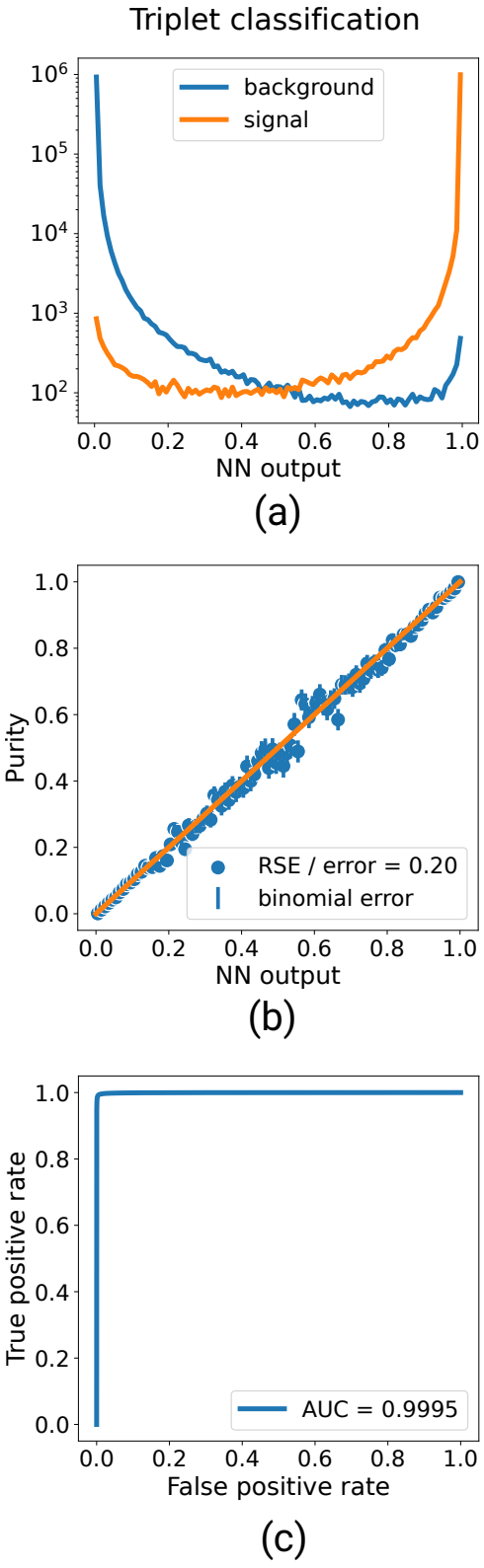

Fig. 2. (a) Network shows clear separation between signal and background samples (note the log scale). (b) Purity as a function of NN output. The NN output can be treated as a probability between 0 and 1. (c) Network shows very low false positive rate.

involving full tracking is considered more suitable for a CPU implementation where highly optimised fitting routines have been developed, and combinatorics have already been significantly reduced. The current iteration of the algorithm used in this study looks only at triplet generation.

TrackNN was developed using PyTorch [5]. It was trained using LHCb Monte-Carlo Inclusive-b and Minimum Bias data samples. The neural network consists of four fully connected layers, with 8 inputs, 32 nodes in each hidden layer, and one output. Fig. 2 shows some performance metrics for the network. In terms of overall performance for the whole LHCb VELO detector, an efficiency of 92%, and a purity of 97% were obtained.

III. FPGA Implementation

The BittWare IA-840f PCIe FPGA card [6] was chosen as the implementation target for the *TrackNN* algorithm. This card has an Altera® Agilex-7 F-series AGF-027 FPGA [7] . This card comes with a Board Support Package from BittWare which is a necessary component for oneAPI functionality. A recent generation processor is also required for oneAPI - Intel® 3rd Gen. Xeon Scalable or above.

Fig. 3 shows a example of a basic SYCL code snippet. A queue is used to define all of the actions to be submitted to a device. In this case an FPGA is chosen (*fpga_selector*). Several memory blocks, in the form of float arrays are allocated on the host and associated with the queue. A kernel named *VectorAdd* is defined, and submitted to the queue. The code in yellow in the figure is what runs on the device; all other code is running on the host CPU. The same memory that was allocated on the host can be accessed on the device via "host pointers". A small work function is defined to add two vectors of numbers. This loop can be unrolled to speed up execution. Internally this is done by exploiting the parallel architecture of the device. In the case of the FPGA, more resources will be used, but the latency of the loop can be reduced significantly (down to one clock cycle in the ideal case). Finally, a "wait" call is used to instruct the host to wait until the kernel has completed before executing further code. In this way, we ensure the array *sum* is not read prematurely, and the contents of the array will be valid and stable.

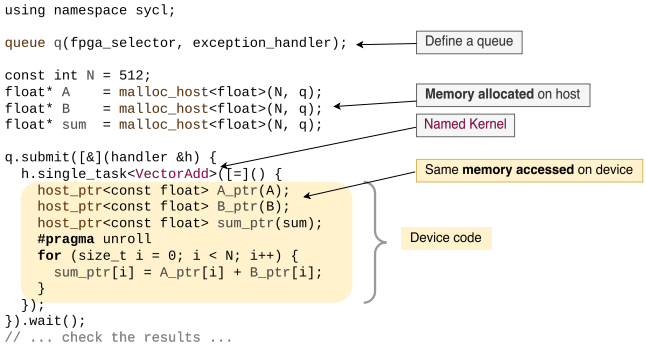

Fig. 3. A SYCL code snippet. Using Unified Shared Memory (USM) declared on the host can be accessed on the device.

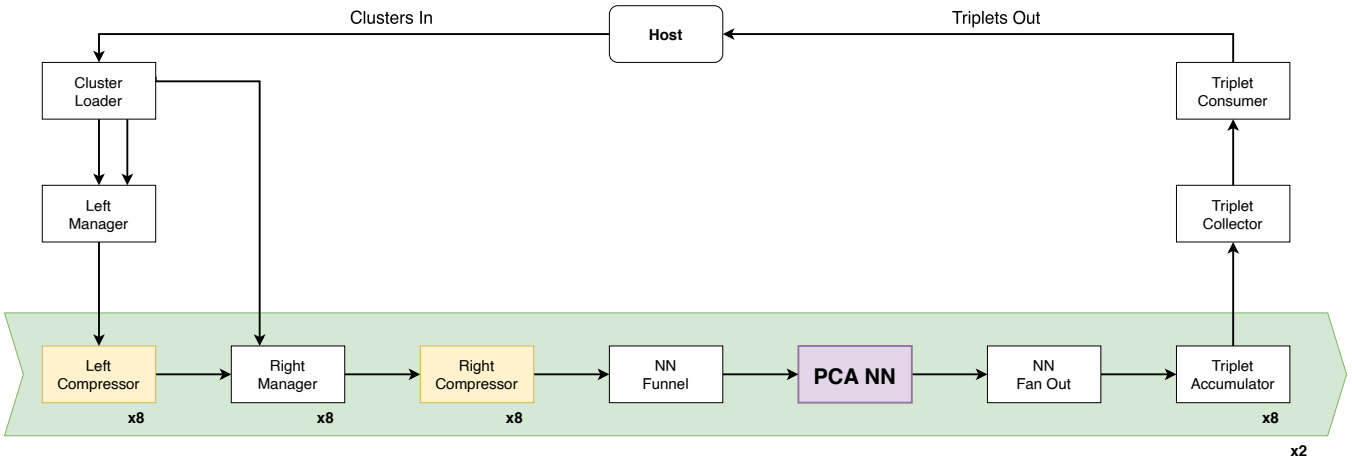

Fig. 4. Data flow from the *Host* (centre top) through the kernel pipelines to the neural net kernels, *PCA NN* (purple), and back to the host. Several kernels have 8 instances, and there are two parallel copies of the pipeline given in green.

The kernels developed for the *TrackNN* algorithm follow an extended version of the formula given in Fig. 3. Communication with the host is achieved using Unified Shared Memory (USM) which allows host memory to be easily accessed from the FPGA[3].

Fig. 4 shows the current design of the *TrackNN* software. The host sends clusters into the device and receives triplets back from the device (both via USM). Communication between kernels is done via SYCL "pipes". In FPGA terms, pipes are essentially FIFOs. Pipes can be blocking or non-blocking depending on the user's preference for flow control. The principal "work" kernel with the neural network is shown in purple marked *PCA NN*. Most of other the kernels shown are organising the data into and out of this kernel. The Cluster Loader takes the input cluster stream and divides it into the left, middle and right detector planes, and send the data onward. The Left and Right Managers are responsible for creating the triplet combinations. It is important to note here that $O(N^3)$ inflation happens to the data by the time they reach the Right Compressor. This dominates the overall throughput of the system.

They Left and Right kernels are split in order to improve throughput. The Compressors assist with variable data sizes. Note that the pipeline defined by the green section has two instances, and several kernels have eight instances within that. In each green pipeline, the *NN Funnel* combines eight data streams into one feeding the *PCA NN* kernel, and conversely the *NN Fan Out* does the opposite. The *Triplet Collector* takes all the data streams from both pipelines and sends it to the *Triplet Consumer* which finally streams the data back to the host via USM.

In summary, most of this complex networking is to achieve higher throughput. The choices driving this design were made using feedback and profiling using oneAPI tools which will be described in the next section.

[3]Alternatively, device memory can also be accessed on the host.

IV. oneAPI Tools

Three tools have been used extensively in the development of the *TrackNN* software with oneAPI. The are:

- The oneAPI FPGA Emulator
- The oneAPI Report Tool
- The Intel® VTune™ Profiler

The FPGA Emulator runs on the CPU host and generates a threads for the "on-device" kernels. The emulator is not a perfect copy of the FPGA, and runs much slower. The emulator allows us to check the correctness of our code without launching a full compilation for the device. Full compilation takes hours to complete even on a modern CPU, whereas the compiling for the emulator takes mere seconds. Consequently, development time spent checking for correctness is significantly reduced.

The oneAPI Report Tool also executes quite quickly (about 30 s for simple designs, a few minutes for designs with many kernels). The Report Tool provides a lot of useful information. An early resource estimate provides detail on the estimated number of logic and memory elements required for each kernel. This feedback allows to focus effort on which kernels to optimise for resource usage. Our experience is that the estimates given very closely match those produced by the Altera® Quartus compiler after synthesis.

The Report Tool also gives feedback on which kernels have been well pipelined. A loop analysis report indicates the Initiation Interval (II) for each loop used in the design. Essentially, the II tells us how many clock cycles are required for each iteration of the loop. Ideally this will have a value of 1. If the value is higher than 1 the analysis report gives the origin of the slowdown (typically a memory dependency). One can try to improve that part of the code in order to improve the throughput.

The VTune™ Profiler is used after a full compilation has been performed. A compiler flag is used to include profiling counters in the build of the FPGA firmware. This adds some extra resource usage to the design (which scales with the design size). Using the profiler, it is possible to see the

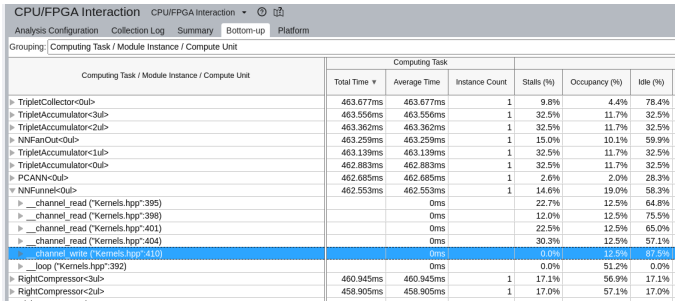

Fig. 5. Screenshot of Intel® VTune™ Profiler. A channel_write call is stalling 62% of the time. Fixing this one stall improved throughput by about a factor of three. Note the unexpanded rows (e.g. NN Funnel) show average percentages of several internal operations, and are not expected to sum to 100%.

input and output bandwidth of the running firmware. Profiling metrics in the form of Stall, Occupancy and Idle are listed for each loop and pipe operation. Stalls indicate a process is waiting to execute due to some bottleneck (e.g., it's not possible to write to a pipe because it is full). Occupancy is the fraction of time spent executing out of the total time. Idle is the time spent neither stalled nor occupied. An example is given in Fig. 5.

V. Experience using oneAPI for FPGA development

The authors had the following observations developing with oneAPI for FPGAs: The tools mentioned in the previous section were found to very powerful. One of the authors had no experience with conventional RTL programming with VHDL or Verilog. Yet, he managed to produce a first working algorithm on the hardware, in a short period of time (less than one month). The FPGA Emulator was found to be a significant improvement over the conventional FPGA programming tools which require a lot of effort devoted to test-benches and simulation in order to prove correctness. The fast compilation time makes this part of the development flow very similar to CPU programming.

The Report Tool is also very useful. In particular, the Loop Analysis report quickly points out inefficiencies in the code, and points to the the part that requires improvement. That said, exactly what causes the compiler to generate a large II, is somewhat opaque. For example, replacing a switch block with an if..else block can in some cases significantly reduce the II. This should not be the case, as most of the time such blocks are functionally equivalent. Some trial and error was required to find the optimal solution for the compiler[4]

The VTune™ Profiler is a multi-language and multi-device capable profiling tool. As such, it has many features and takes some time to become familiar with it, and many of the results can seem confusing at first. However, as with all of the tools mentioned in this paper, there is plentiful documentation with manuals, tutorials and training videos available online. VTune™ provides essential feedback when running a program on the FPGA. It is key to getting performance out of the design.

Lastly, it is worth mentioning that in order to efficiently program an FPGA, it is necessary to know about the architecture itself. *Pipelining* is an essential concept that must be understood to achieve reasonable results. This is a different paradigm to GPU programming which relies on massive parallelism. Pipelining allows many different tasks to be connected together in a chain, much like an assembly line. Optimisation is achieved by keeping all of the workers busy for as much time as possible. Parallelism can be added by having multiple pipelines. This is exemplified in the *TrackNN* architecture given earlier in Fig. 4.

[4]It should be noted that observations are for oneAPI version 2023.1. Improvements to the compiler are expected with newer versions.

VI. Future Work

The next upgrade of the LHCb experiment plans to use the PCIe400 DAQ readout board [8]. This will use an Agilex M-series FPGA with approximately twice the performance of the AGF-027 on the BittWare IA-840f card. Since LHCb will purchase many of these cards, we wish to explore the feasibility of deploying oneAPI algorithms to this card.

A board support package (BSP) would need to be developed for the PCIe400 to enable it to be used with oneAPI. Alternatively, there exists a feature of oneAPI that does not require a BSP, known as is "IP Flow". This allows an algorithm developed with oneAPI to be exported as an IP block, and integrated into an existing RTL design. This offers the potential to supercharge DAQ processing flows with high level algorithms.

The next generation of particle detectors will have significantly higher luminosities (7× higher in the case of LHCb) so it will be necessary to prove the *TrackNN* algorithm at those scales. Comparison with GPU is also essential in order to evaluate the FPGA as the right platform in terms of performance and cost. This is another reason the oneAPI framework has been chosen - it is possible to target GPUs rather than FPGA using the same framework. The code will require non-trivial changes in order to fairly compare the two device types, but learning a completely new tool-set would not be necessary.

VII. Concluding remarks

The oneAPI framework has been evaluated for FPGA development for High Energy Physics experiments. The *TrackNN* neural network tracking algorithm was ported to oneAPI in approximately four months by two developers. A fully optimised design has yet to be achieved, but the current implementation produces consistently verifiable results from the Altera® AGF-027 FPGA that was chosen. In what has traditionally been a domain predominantly restricted to electronic engineers, the authors see significant potential in this technology to make FPGA hardware accessible to software developers, and thereby further exploit them as compute accelerators.

References

[1] The Khronos Group Inc. *Khronos SYCL Registry* [Online]. Available: https://registry.khronos.org/SYCL/

[2] The LHCb Collaboration, "The LHCb Detector at the LHC", *JINST*, vol. 3, Aug. 2008, S08005.
[3] The LHCb Collaboration, "LHCb VELO Upgrade Technical Design Report", CERN, Switzerland, 2013. [Online] Available: https://cds.cern.ch/record/1624070.
[4] P. Marshall, "Developing a Hybrid Machine Learning Model for VELO Upgrade Track Reconstruction", Ph.D. thesis, Dept. Physics, Univ. Liverpool, Liverpool, UK, 2022. p. 69. [Online] https://cds.cern.ch/record/2838196
[5] The Linux Foundation *PyTorch* [Online]. Available: https://pytorch.org
[6] BittWare *IA-840f Data Sheet* 2024 [Online]. Available: https://www.bittware.com/files/IA-840F_datasheet_r3v1.pdf
[7] Altera® *Agilex 7 Product Brief* 2024 [Online]. Available: https://www.intel.com/content/www/us/en/content-details/762901/agilex-7-fpgas-and-socs-product-brief.html
[8] J. Langouet *et al.* *Future DAQ Boards: PCIe400* 2024 [Online]. Available: https://indico.icc.ub.edu/event/163/contributions/1416/attachments/683/1354/230330_future_DAQ_boards.pdf