Kalman Filter track reconstruction on FPGAs for acceleration of the High Level Trigger of the CMS experiment at the HL-LHC

Sioni Summers[1,*] and *Andrew* Rose[1]on behalf of the CMS Collaboration

[1]Imperial College London

Abstract. Track reconstruction at the CMS experiment uses the Combinatorial Kalman Filter. The algorithm computation time scales exponentially with pileup, which will pose a problem for the High Level Trigger at the High Luminosity LHC. FPGAs, which are already used extensively in hardware triggers, are becoming more widely used for compute acceleration. With a combination of high performance, energy efficiency, and predictable and low latency, FPGA accelerators are an interesting technology for high energy physics. Here, progress towards porting of the CMS track reconstruction to Maxeler Technologies' Dataflow Engines is shown, programmed with their high level language MaxJ. The performance is compared to CPUs, and further steps to optimise for the architecture are presented.

1 Introduction

At the CMS experiment [1] at the LHC, events are filtered by a trigger system, the final stage of which is the High Level Trigger (HLT) processor farm. Reconstruction of charged particle tracks is an important, and compute intensive, part of the event processing in the HLT. Tracking is used only sparingly, for events which have not been rejected using information from the calorimeter or muon subdetectors. At the High Luminosity Large Hadron Collider (HL-LHC), where the CMS experiment will be delivered up to 200 simultaneous proton-proton collisions per bunch crossing (pileup), the track reconstruction task will be made even more difficult. The large number of charged particles will take longer to reconstruct compared to LHC conditions, outpacing the anticipated improvement in computing power before the startup of the HL-LHC. For the HLT this means that more processors will be needed to keep up with the incoming rate of events. The use of alternative processing architectures is being explored by CMS [2], aiming to accelerate the reconstruction, using fewer processors and lowering power consumption.

Dataflow Engines (DFEs) are a compute acceleration platform comprising an FPGA for processing, onboard memory, and a connection to a host CPU. The large number of processing resources available in FPGAs makes enables a high level of parallelism. Many operations can execute simultaneously, reducing the latency, while a processing pipeline, created by the placement of registers, provides high throughput. The language MaxJ and compiler MaxCompiler, provided by Maxeler Technologies, are used to develop applications for DFEs, simplifying the design process.

[*]e-mail: sioni.summers10@imperial.ac.uk

Tracking at CMS is carried out with the Combinatorial Track Finder [3]. The procedure finds charged particles by iterating the four step procedure: seeding, building, fitting, and selection. In the seeding step, hits in the inner pixel detector are used to make estimates of track parameters. During track building these initial estimates are followed, searching for hits in the other tracker layers, with the track parameters updated using a Kalman Filter [4, 5]. Found tracks are fit again using a Kalman Filter and smoothing step. Finally, cuts are performed on the resulting tracks, improving the quality of selection. The steps are executed iteratively, each time targeting different classes of tracks, with the hits formed into tracks on preceding iterations removed. Track reconstruction accounts for over half of the total event reconstruction time, and the time spent in tracking increases rapidly with pileup, requiring a factor 10 more time with 140 pileup than with 70 pileup [6]. The majority of the track reconstruction time is spent on track building [3].

The Kalman Filter is ubiquitous in embedded applications, which frequently use FPGAs for their capabilities for tight integration with sensors. Examples include guidance in aviation, and control and sensor-fusion applications such as in vehicles and robotics. Within these domains the Kalman Filter has been implemented on FPGAs with high throughput and low latency in applications such as an automotive antilock braking system (ABS) [7], and robotic self-localization [8]. The HEP tracking problem differs from these applications in the combinatorial aspect. The general problem of state estimation from sensors for guidance does not have the same ambiguity when selecting the next measurement to filter, requiring the exploration of multiple possible combinations. A low latency Kalman Filter running on an FPGA was used for a track fit in a demonstration of track finding for the CMS Level-1 Trigger [9]. Track reconstruction has also been ported to other parallel architectures, such as GPUs and Intel Xeon Phi [10].

2 Kalman Filter on a Dataflow Engine

The CMS track building primarily consists of three procedures: propagation of track parameters to a detector surface, searching for hits in proximity to the projection, and updating the track parameters according to the Kalman Filter. The propagation and parameter update are the most suited to be executed on the DFE, which can perform the same computation on different data in a pipelined manner. For each track trajectory propagated to a detector layer, multiple compatible hits may be found. All of these are used for a Kalman Filter update with the trajectory, after which only the few with the lowest χ^2 are kept. It is these kept, updated, tracks which are propagated again, so there are always fewer propagations than trajectory updates. For these reasons, the Kalman Filter update was chosen to be implemented on the DFE.

The Kalman Filter state update is a series of products and additions of matrices of sizes 5×5, 5×2, and 2×2. In the FPGA, all these operations are carried out on different circuitry. All the products of pairs of elements for a matrix multiplication are executed simultaneously, with the sums carried out using a tree of adders. The MaxCompiler scheduler automatically places the sequence of operations in series where they cannot be performed in parallel.

2.1 Number representation

FPGAs allow completely customised data representations. In terms of resource usage of the FPGA, the fixed-point representation is the most efficient. With fixed-point data, integer operations are used, which avoids the normalisation-denormalisation steps that are required when working with floating point data. However, this number representation has a limited dynamic range compared to floating point, so can be prone to under- or over-flow.

Table 1. The range of the exponent of floating point variables in the Kalman Filter state update.

Matrix	**row** \ **column**	1	2	3	4	5
C	1	18	27	34	31	31
	2	-	37	45	34	40
	3	-	-	33	44	29
	4	-	-	-	34	37
	5	-	-	-	-	26
$\mathbf{x}^T$	-	19	25	23	21	20
$\mathbf{K}^T$	1	23	20	29	18	50
	2	23	30	27	41	23

The tracking software was numerically profiled in order to determine whether a fixed-point representation is suitable, using simulated events of $t\bar{t}$ with 200 pileup. Table 1 shows the range of the exponent of the floating point variables in some of the key matrices of the Kalman Filter. A fixed point representation would require at least as many bits as shown in the Table to represent the value without under- or over-flow across its full range, albeit with 1 bit of precision at the smallest value. To maintain as much precision as a single precision floating point representation of the same value, a further 24 bits would be needed in addition to the presented value.

The Altera Stratix V DSP blocks (which perform the multiplications) have two inputs of 27 bits each. Multiple blocks can be combined to perform multiplications on wider inputs, where up to 9 blocks would be required for some of the products in the Kalman Filter. Compared to this, a floating point product can be executed in one DSP block, with additional logic outside the DSP carrying out the (de)normalisation of the exponents. The design presented here was implemented using single precision floating point in order to guarantee precision across the full range of a variable, without using very wide multipliers.

2.2 Dataflow

The CMS experiment software (CMSSW [11]) tracking, running on a host CPU, was modified to use the DFE to perform state updates rather than executing them on the CPU. The flow of data between host and DFE is shown in Figure 1. Each state update requires a Kalman Filter state and a hit, each represented by a vector of measurements and their covariance matrix. The DFE passes these through the update algorithm, and returns the updated state to the host. The host then performs the projection to the next detector layer and the search for compatible hits, before sending these to the DFE.

The host application was reconfigured to buffer a number of hits and states before running the DFE update, to minimise the number of transactions between the devices, and to fully utilise the pipeline on the DFE. As mentioned, each state may require updating with multiple hits, so all of these are buffered together. In addition, multiple states are undergoing processing simultaneously, starting from the seeds produced, and these are all buffered. Ultimately, all current combinations of states and hits are sent together, before the updated states are required to find the next hits.

Figure 2 shows the frequency of the number of state update inputs that can be buffered and sent to the DFE for processing in one transaction, using simulated $t\bar{t}$ events with 200 pileup. While transactions with small numbers of updates are the most frequent, transactions with large numbers of updates take correspondingly longer to process on the CPU, hence the

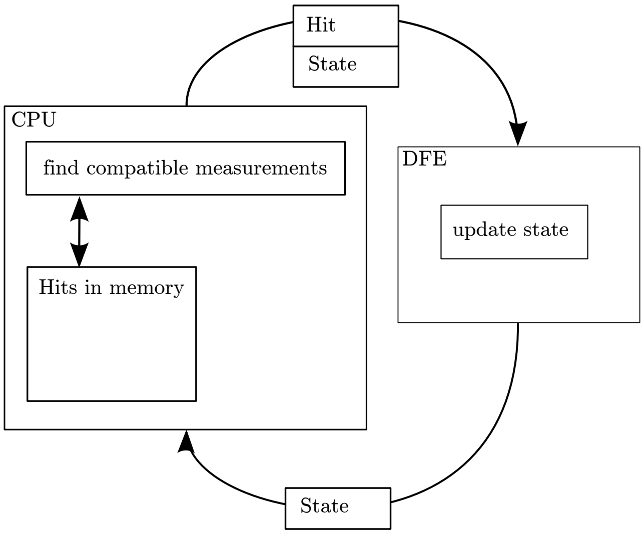

Figure 1. Illustration of the processing carried out by the DFE and host CPU and the movement of data between them.

Table 2. Resource usage and timing metrics of the Kalman Filter state updater, and peripheral communications, on a Maia DFE. Resources are also expressed as a percentage of the Altera Stratix V 5SGSD8 device used.

Resource	**Number Used**	**Percentage**
ALMs	76054	28.98 %
DSP Blocks	149	7.59%
Block Memory	1027	40.01%
Latency (cycles)	181	
Clock Frequency	250 MHz	

algorithm pipelining on the DFE is expected to provide better performance as the amount of processing per transaction increases.

3 Performance

The Kalman Filter state update design was compiled for a Stratix V 5SGSD8 FPGA on a Maxeler Maia DFE, hosted in an MPC-X machine. Table 2 shows the FPGA resource utilisation and timing performance of the algorithm. At the achieved clock frequency of 250 MHz the design takes approximately 750 ns to perform the update, while the full pipelining means that a new state-hit combination can be input every 4 ns. The most used resource is the block memory, at 40% of the total available, and is mostly used to hold intermediate calculations.

The design was tested in a system with the MPC-X DFE system and a host with an Intel Xeon Sandy Bridge E5-2650v2 processor, running the CMSSW application. Communication between the host and a PCIe switch in the MPC-X takes place over an Infiniband network. Each transaction to the DFE was found to incur a large 500 μs overhead, where the time per iteration on the CPU is (255 ± 34) ns. After the initial overhead, the DFE iterates at a rate of 17.4 MHz compared to 3.92 MHz on the CPU. The DFE would eventually become faster when performing around 2500 updates per transaction, but, as can be seen from Figure 2,

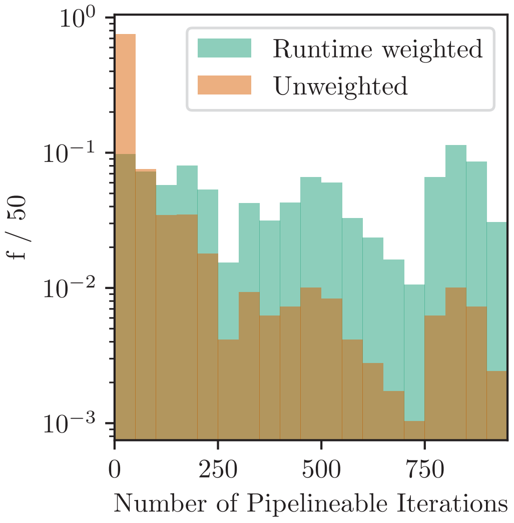

Figure 2. Histogram of the number of Kalman Filter state and hit combinations that can be buffered and sent to the DFE in one iteration. As well as the count, the fraction of time spent processing that number of updates on the CPU is shown.

this is insufficient to accelerate the tracking given realistic numbers of updates in high pileup simulations. The rate of iteration on the DFE is lower than the 250 MHz upper limit of the algorithm clock frequency, and is limited by the 2 GBs^{-1} bandwidth of the PCIe connection.

4 Future Work

Further work is needed to obtain an overall speedup of the tracking with a DFE. A major gain will be made by reducing the communication between the host and DFE, which may be achieved by performing more of the computation on the DFE. The most significant improvement would be to remove the loop between the devices, which avoids repeatedly spending the initial overhead. This would require the entire tracking algorithm to be ported to the DFE, with an efficient implementation of the searching for hits.

Following this, further gains might be made by further numerical optimisation, using fixed-point representation if possible. A fixed-point implementation would reduce the logic usage around multiplications, and simplify additions, also reducing logic. The (de)normalisation required for floating point introduces some strain on the routing of operations in the FPGA, so removing this would potentially enable a faster clock frequency. If the tracking performance is not harmed by using reduced precision, then the DSP usage could be reduced by using narrower bit widths. Reducing the resource usage through the methods outlined would allow multiple copies of the algorithm to run in parallel in a single device, increasing the achievable throughput.

Additional gains will be made from the continued improvement in FPGA acceleration technology. Both Altera (now Intel) and Xilinx continue to produce devices with greater amounts of programmable resources. Their smaller process sizes will enable lower power consumption and faster clock frequencies. Next generation devices from Intel have more tailored support for floating point operations, with the entire operation taking place inside the DSP core, which ought to improve the achievable clock frequency of the design. Future PCIe and networking specifications will enable higher bandwidth communication between host and coprocessor, increasing the processing throughput of the design.

5 Conclusion

The Kalman Filter state update of the CMS experiment track reconstruction has been ported to a Dataflow Engine, and interfaced with the tracking software. A faster rate of processing was achieved on the DFE, but with a long initial overhead preventing an overall speedup. Steps to achieve a speedup were outlined, including moving the remaining computation on the CPU over to the DFE, and changing to a fixed-point numerical representation.

6 Acknowledgements

The authors would like to acknowledge the support of Maxeler Technologies for their contribution to this work.

References

[1] The CMS Collaboration, J. Instrum. **3**, S08004 (2008)
[2] J.M. Andre, U. Behrens, J. Branson, P. Brummer, O. Chaze, S. Cittolin, C. Contescu, B.G. Craigs, G.L. Darlea, C. Deldicque et al., J. Phys.: Conf. Ser. **898**, 032019 (2017)
[3] The CMS Collaboration, J. Instrum. **9**, P10009 (2014)
[4] R.E. Kálmán, Trans. ASME J. Basic Eng. **82**, 35 (1960)
[5] R. Fruhwirth, Nucl. Instrum. Meth. **A262**, 444 (1987)
[6] M. Rovere, J. Phys.: Conf. Ser. **664**, 072040. 8 p (2015)
[7] F. Sandhu, H. Selamat, S.E. Alavi, V. Behtaji Siahkal Mahalleh, IEEE Sens. J. **17**, 5749 (2017)
[8] S. Cruz, D.M. Muñoz, M. Conde, C.H. Llanos, G.A. Borges, in *Proc. IEEE 4th Latin Amer. Symp. Circuits Syst. (LASCAS)* (IEEE, 2013), pp. 1–4
[9] R. Aggleton, L. Ardila-Perez, F. Ball, M. Balzer, G. Boudoul, J. Brooke, M. Caselle, L. Calligaris, D. Cieri, E. Clement et al., J. Instrum. **12**, P12019 (2017)
[10] G. Cerati, P. Elmer, S. Krutelyov, S. Lantz, M. Lefebvre, M. Masciovecchio, K. McDermott, D. Riley, M. Tadel, P. Wittich et al., J. Phys.: Conf. Ser. **1085**, 042016 (2018)
[11] D.J. Lange, J. Phys.: Conf. Ser. **331**, 032020 (2011)