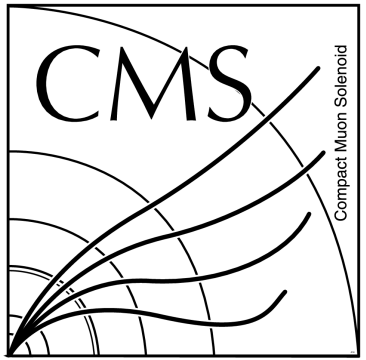
Kalman Filter track reconstruction on FPGAs for acceleration of the High Level Trigger of the CMS experiment at the HL-LHC

CHEP 2018

Sioni Summers



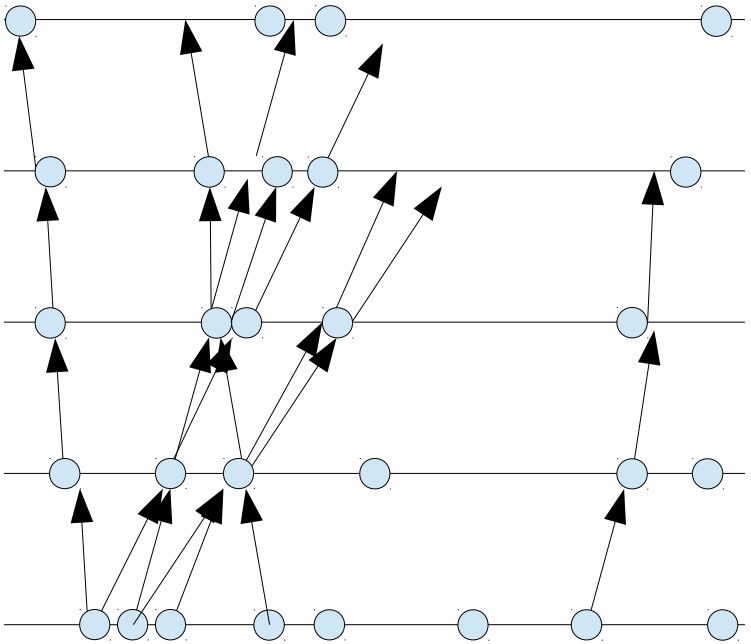
Imperial College London

Contents

- FPGAs and Dataflow Engines
- Kalman Filter
- Kalman Filter on a Dataflow Engine

Introduction

- FPGA accelerators increasingly popular for data centres
  - Reconfigurable hardware accelerators
  - Fixed low latency processing
  - Power efficient: Flops / Watt
- CMS tracking is a compute intensive part of reconstruction
  - Processing time scales steeply with pileup
- With High Luminosity LHC conditions, can FPGAs accelerate the algorithm, potentially saving time and resources at the HLT?



FPGAs / Dataflow Engines

FPGAS

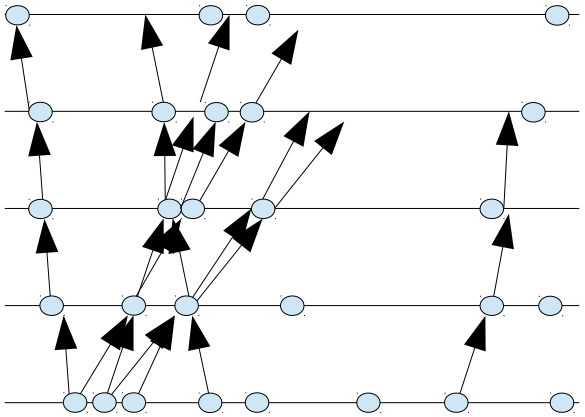
- Configurable digital circuit
  - $O(10^4)$ multipliers
  - O(10) MB internal memory
  - Millions of general purpose logic cells
- Fine grained parallelism of operations
  - e.g. all element-wise multiplications of a matrix product simultaneously
  - Low latency
- Pipeline parallelism
  - Fully pipelined block accepts new data every clock cycle
  - High throughput

Dataflow Engines (DFEs)

- FPGA on a PCIe card
  - O(10) GB DDR RAM on card
  - DFE-DFE optical links
- MaxJ / MaxCompiler / MaxIDE
  - High level language & compiler for programming DFEs
  - Design scheduling, custom numerics
- MaxelerOS
  - Software for easier integration with host applications

Combinatorial Kalman Filter

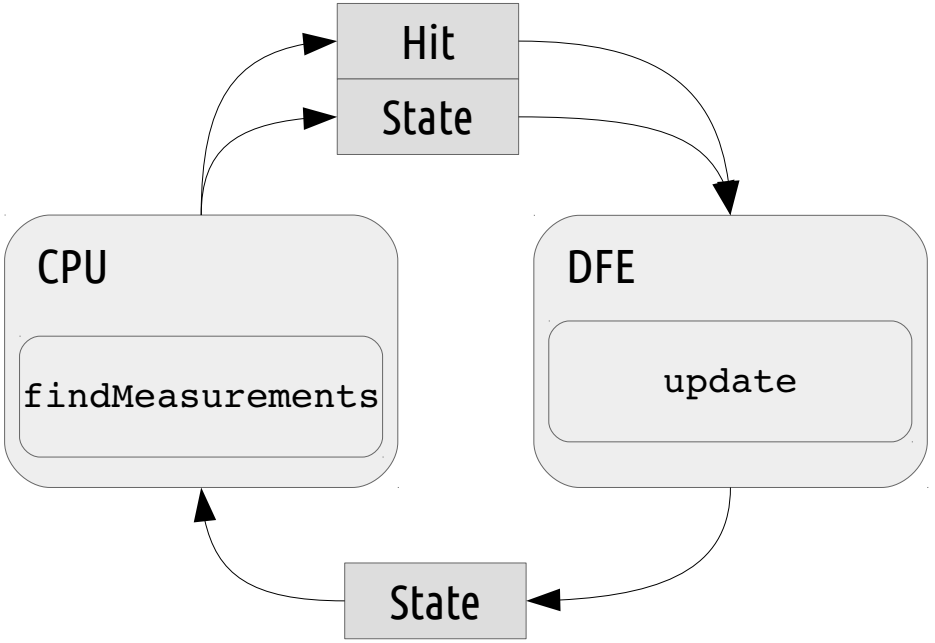
- Iteratively propagate & update track parameter estimate
- For all track candidate states:
  - Propagate state to next detector layer
  - Find all compatible measurements
  - Update states according to formalism
  - Keep best N (limits combinatorial explosion)
- Update requires sequence of products and sums of small matrices



$$
\begin{aligned}
x_k^{k-1} &= \mathbf{F}_{k-1} x_{k-1} \\
\mathbf{C}_k^{k-1} &= \mathbf{F}_{k-1} \mathbf{C}_{k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \\
r_k^{k-1} &= m_k - \mathbf{H}_k x_k^{k-1} \\
\mathbf{R}_k^{k-1} &= \mathbf{V}_k + \mathbf{H}_k \mathbf{C}_k^{k-1} \mathbf{H}_k^T \\
\mathbf{K}_k &= \mathbf{C}_k^{k-1} \mathbf{H}_k^T \left(\mathbf{R}_k^{k-1}\right)^{-1} \\
x_k &= x_k^{k-1} + \mathbf{K}_k r_k^{k-1} \\
\mathbf{C}_k &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{C}_k^{k-1} \\
\chi_+^2 &= r_k^{k-1T} \left(\mathbf{R}_k^{k-1}\right)^{-1} r_k^{k-1} \\
\chi_k^2 &= \chi_{k-1}^2 + \chi_+^2
\end{aligned}
$$

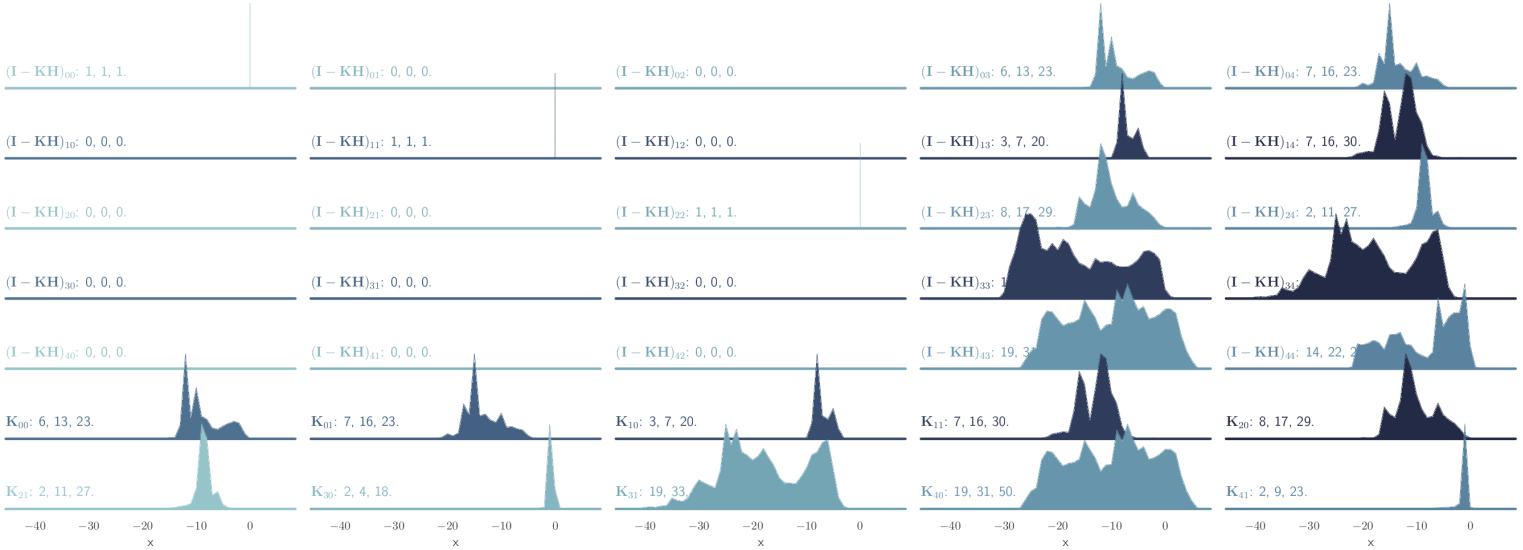
Porting to DFE

- Find compatible measurements on CPU
  - Measurement finding is highly data dependent processing
  - Not suited to effective implementation in FPGA (no std::find for FPGAs!)
- Update state on DFE
  - Parallelisable matrix operations
  - Same operations on every data



Kalman Filter Kernel Numerics

- Floating point on FPGAs is expensive: use fixed-point where possible
- Numerical profiling shows dynamic range required: too wide to use fixed point with sensible resource usage → use floating point

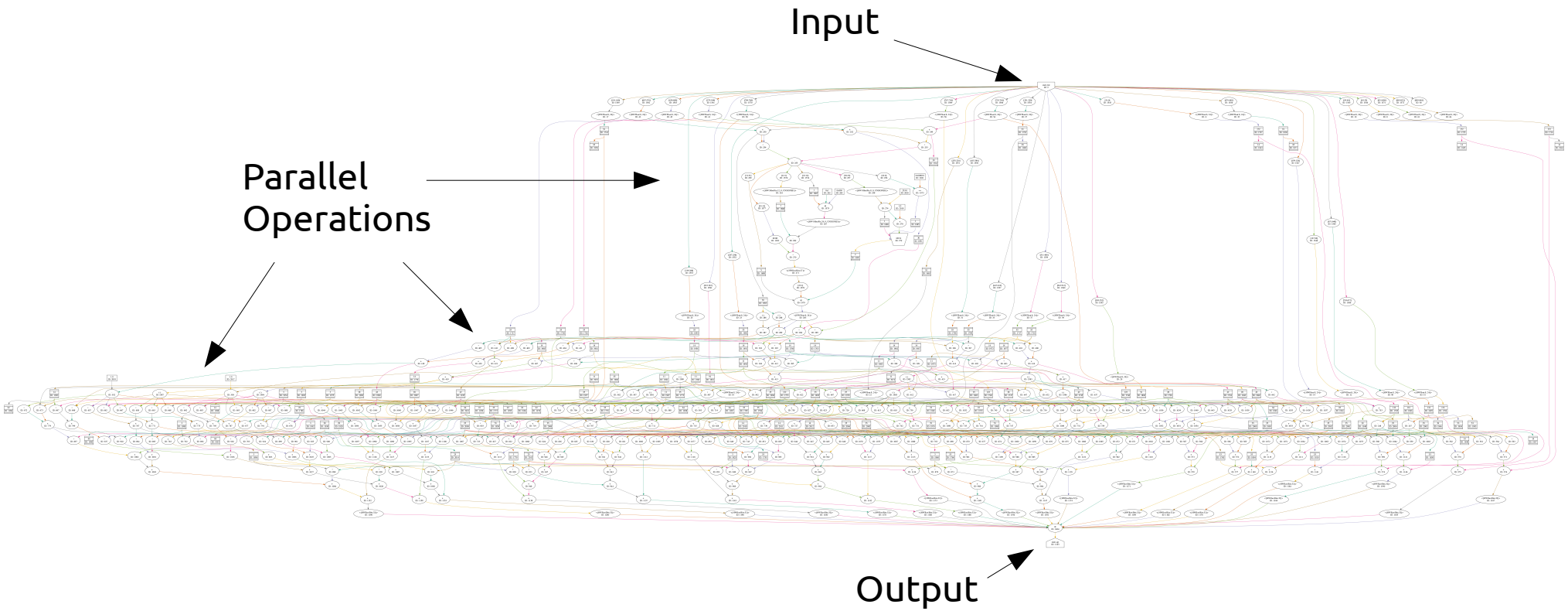


Kernel Metrics

- Latency is 181 clock cycles at 250MHz: ~725ns
  - ~250ns measured on CPU
  - But *fully pipelined*: new state & hit enters on each clock cycle
- Modest multiplier usage
- Logic utilised for add/subtract and (de)normalisation
- Memories used for synchronisation

```
PRELIMINARY RESOURCE USAGE
Logic utilization:      86453 / 262400   (32.95%)
Multipliers (18x18):      298 / 3926     (7.59%)
Block memory (bits):  1929838 / 52572160 (3.67%)
```

Dataflow Graph



Host-side interface

- DFE design interfaced with CMS experiment software tracking
- MaxelerOS handles the low level communication
- Reorder some loops to send more data to DFE in one transaction
  - As much as possible before updated states are needed
- Send states 'run length encoded'
  - Multiple hits are filtered with the same state → use less link bandwidth

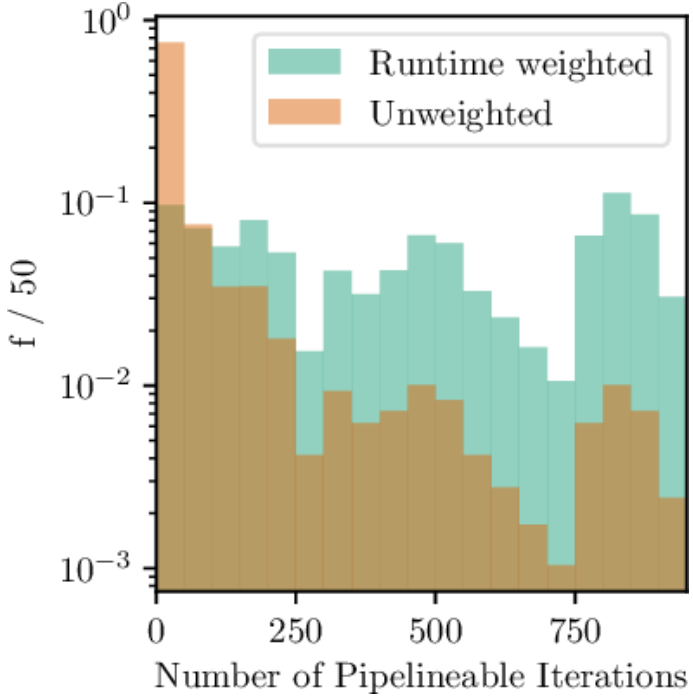
Dataflow Engine setup

- Maxeler 'MPC-X' node
  - 'DFEs as a shared resource'
  - Altera Stratix V FPGAs
  - 8 DFEs per 1U box
  - Infiniband network & PCIe switch
- Intel Xeon X5650 2.67GHz CPUs

FPGA vs. CPU timing

- Testing with $t\bar{t}$ + 200 pileup simulated events
- So far, no speedup!
- FPGA achieves ~4.5x rate increase
  - Limited at 3 GB/s PCIe bandwidth
    - Cannot use more KFs in parallel, either
- Long initial latency costly
  - 0.5 ms
  - Algorithm latency not enough to hide it
- FPGA would become faster for n > 2500 iterations

	Throughput (MHz)	FIFO Latency (ms)
CPU	3.9	-
FPGA	17.4	0.5



Future Work

- Overcome per-transaction latency cost
  - Move measurement finding to DFE
  - Even if task is slower, may enable total speedup by utilising the fast memory bandwidth compared to PCIe
- Investigate fixed-point arithmetic
  - Wide types or dynamic fixed point scaling (once per update rather than once per operation)
  - Must take care with numerical stability
- Use on-card networking for tighter integration

Conclusion

- FPGAs offer predictable low latency, and best Flops / Watt
- Dataflow Engines provide easy development and use of FPGA processing
- Part of track reconstruction Kalman Filter ported to DFE
- Promising performance shown, but further development needed to achieve speedup