FPGA Tracking with oneAPI

Karol Hennessy, Kurt Rinnert

24th April 2024

UNIVERSITY OF LIVERPOOL

Overview

Same beach, 10 years ago

Thanks to the organisers. Nice to be back to beautiful Quy Nhon!

Overview

- The plan
- The algorithm
- The framework
- The experience
- The current implementation
- The future and final thoughts

Same beach, 10 years ago

Thanks to the organisers. Nice to be back to beautiful Quy Nhon!

The Plan

- Use LHCb VELO as a test-bed.
- Connect the dots
  - Take 3 detector planes (**L**eft, **M**iddle, **R**ight)
  - make all combinations of triplets from hit clusters
  - choose the “best” triplet
  - connect triplets and make track fit
- Replace the steps in conventional algorithms with learned functions.
- Port the resulting models to FPGA

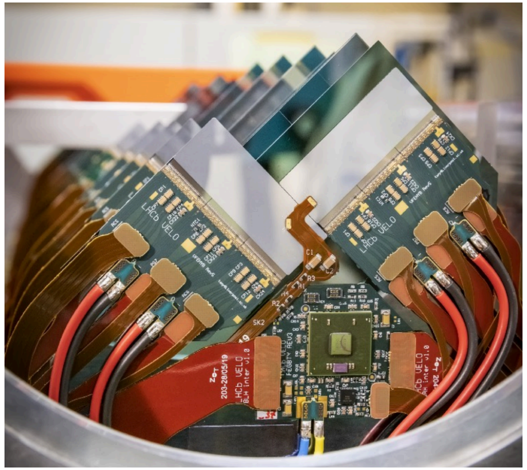
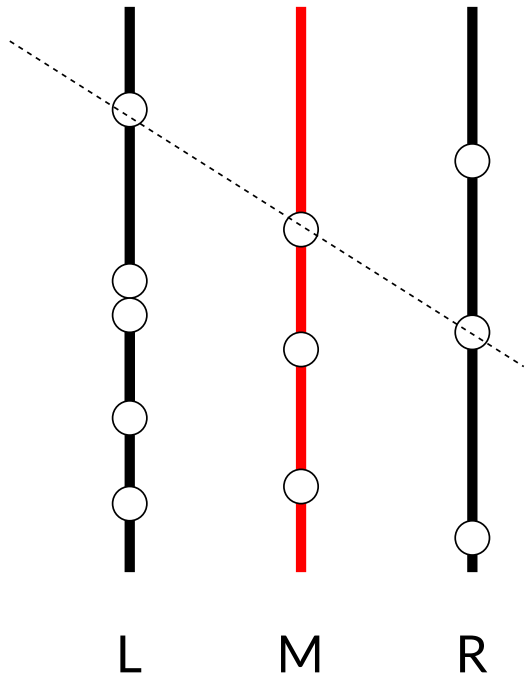


Figure 3: LHCb VELO

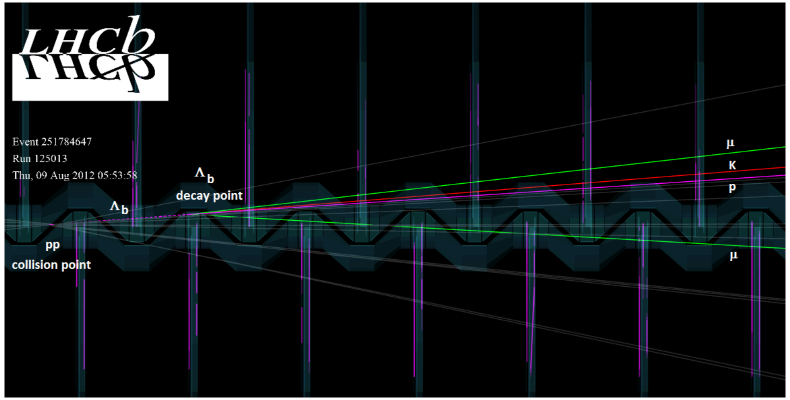
The Plan

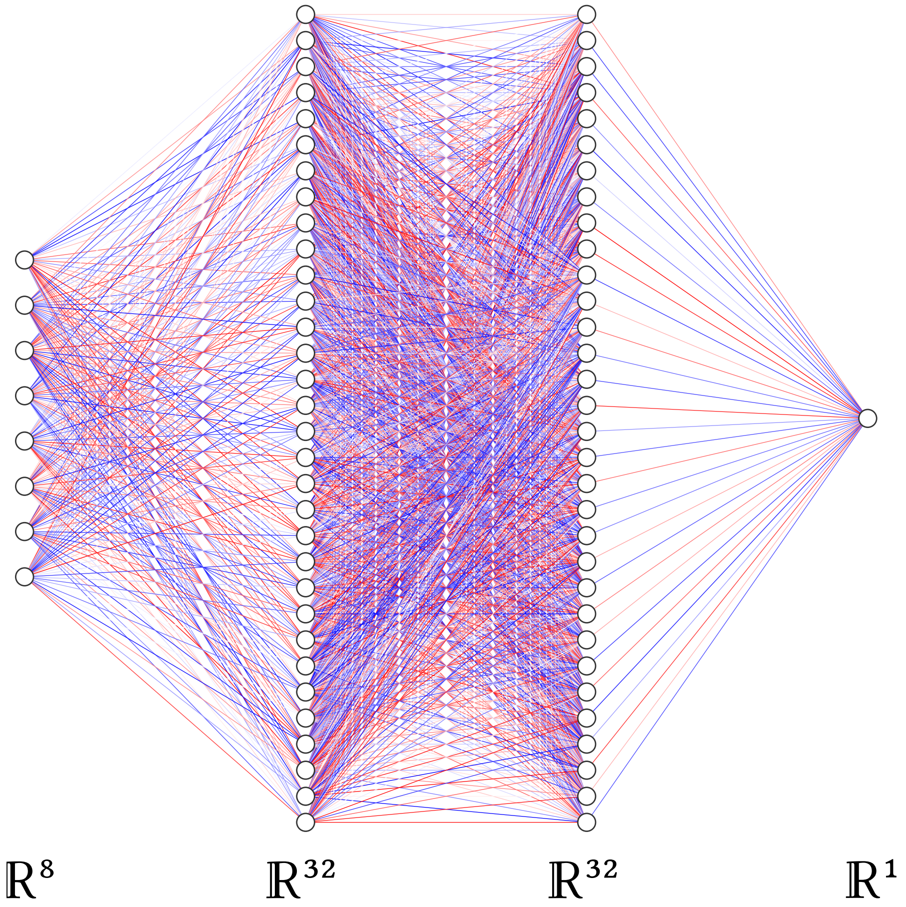
- Use LHCb VELO as a test-bed.
- Connect the dots
  - Take 3 detector planes (**L**eft, **M**iddle, **R**ight)
  - **make all combinations of triplets from hit clusters**
  - **choose the "best" triplet**
  - connect triplets and make track fit
- Replace the steps in conventional algorithms with learned functions.
- **Port the resulting models to FPGA**



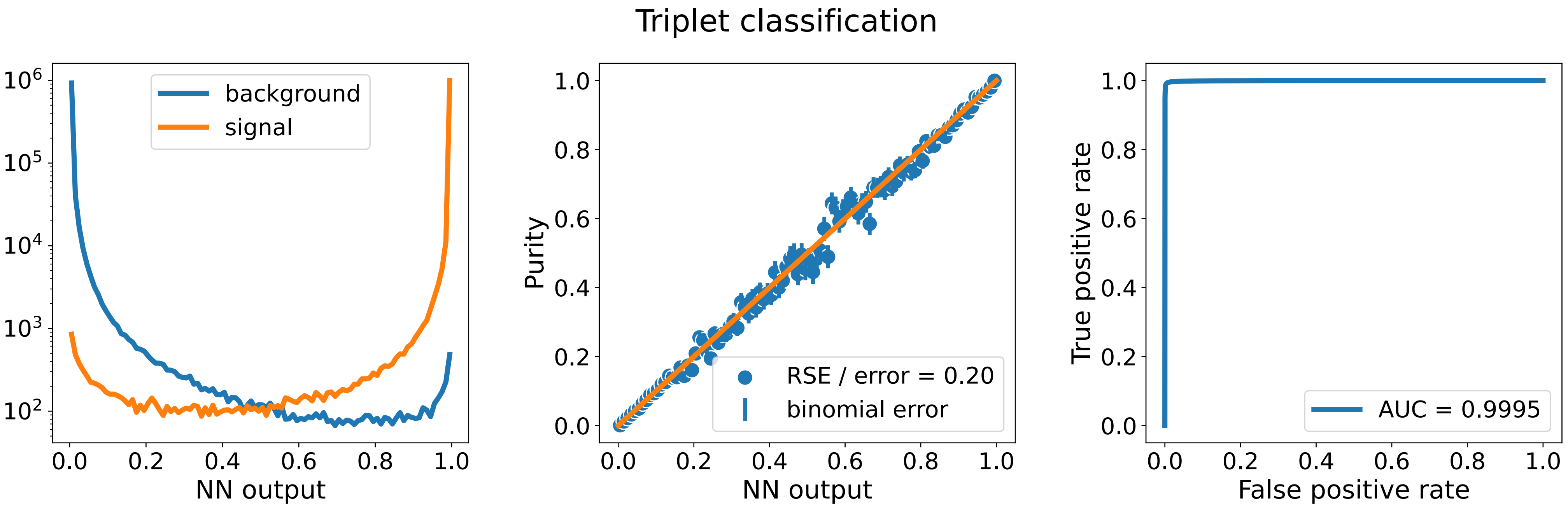
The algorithm

- NN developed using PyTorch
- Homebrew - developed for HEP
- Using Monte-Carlo data from LHCb
  - Inclusive-b & Minimum Bias
- Check the resultant tracklets vs. MC-truth





Network Performance

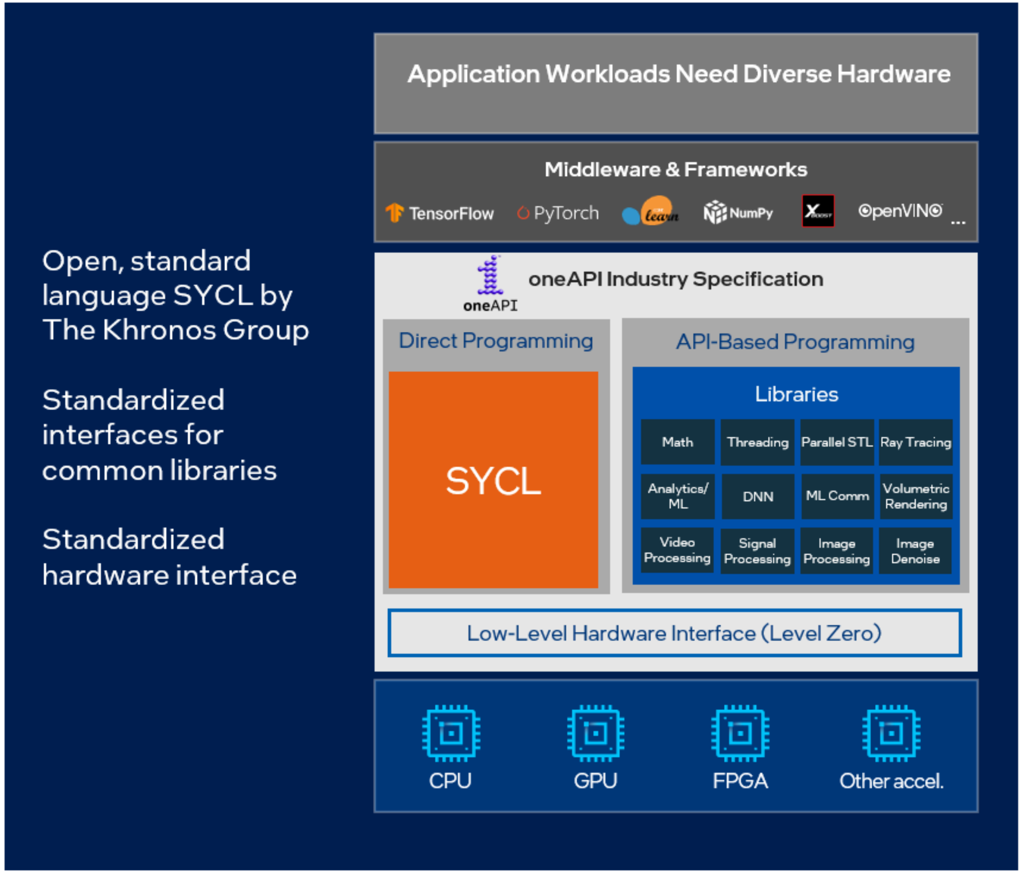


- non-linear activation between input and hidden layers is ReLU
- PCA rotation from features [12] to input layer [8]

Network Performance

- Performance: high efficiency 92%, purity 97%, low ghost rates

The framework



- oneAPI: the pitch
  - Write code in **C++** and **SYCL** (Data Parallel C++)
  - Verify and Deploy to accelerator (GPU, FPGA)
  - Integrate with existing workflows
    - e.g. add IP to an existing RTL design
  - Not restricted to ML/AI
  - Unified Shared Memory
  - *Open Standard*

The Hardware

- BittWare IA-840f
  - Altera Agilex-7 F-series AGF027 FPGA
  - 2.7M LEs

- Supermicro server
  - Intel® 3rd Gen Xeon Scalable (Ice Lake) or above

- Both FPGA and CPU requirements are non-negligeable in terms of cost

The implementation plan

- **Two person team**
  - one **C++** developer (but no FPGA development experience)
  - one **FPGA** developer (not up to date with 21st century C++)
- **Start simple**
  - Try to convert the PyTorch algorithm into C++/SYCL
  - Deploy to FPGA
  - Check results match CPU version
- **Then start scaling up**
  - How much can we parallelise/fit onto the device?
  - How much throughput can we get?

SYCL code - simple example

```
using namespace sycl;

queue q(fpga_selector, exception_handler);

const int N = 512;
float* A    = malloc_host<float>(N, q);
float* B    = malloc_host<float>(N, q);
float* sum  = malloc_host<float>(N, q);

q.submit([&](handler &h) {
  h.single_task<VectorAdd>([=]() {
    host_ptr<const float> A_ptr(A);
    host_ptr<const float> B_ptr(B);
    host_ptr<const float> sum_ptr(sum);
    #pragma unroll
    for (size_t i = 0; i < N; i++) {
      sum_ptr[i] = A_ptr[i] + B_ptr[i];
    }
  });
}).wait();
// ... check the results ...
```

Define a queue

Memory allocated on host

Named Kernel

Same **memory accessed** on device

Device code

oneAPI Tools

The oneAPI tools: FPGA Emulator

- **How do we know if our code works?**
  - Essentially:
    - `make fpga_emu`
  - Creates a binary that runs on the CPU with threads for each “on-device” kernel
  - **Compiles in seconds!**
    - Runs slower than FPGA (naturally)
  - We can **check the correctness of our code** before doing a real build for hardware
    - Full compilation for hardware still takes hours

oneAPI Report Tool: Resource Usage

Summary Views Throughput Analysis Area Estimates

Area Estimates

Notation *file:X* > *file:Y* indicates a function call on line X was inlined using code on line Y.

	Source Location	ALUTs	FFs	RAMs	MLABs	DSPs	Brief Details
Kernel System		563305 (31%)	1610314 (44%)	4627.8 (35%)	7711 (8%)	1160 (14%)	
Board interface		268666	537332	1536	0	778	Platform interface logic.
Pipe resources		1008 (<1%)	76184 (2%)	348 (3%)	0 (0%)	0 (0%)	
Coalesce01	Kernels.hpp:415	4501 (<1%)	5845 (<1%)	0 (0%)	33 (<1%)	0 (0%)	1 compute unit.
Coalesce23	Kernels.hpp:415	4501 (<1%)	5845 (<1%)	0 (0%)	33 (<1%)	0 (0%)	1 compute unit.
Coalesce57	Kernels.hpp:415	4501 (<1%)	5845 (<1%)	0 (0%)	33 (<1%)	0 (0%)	1 compute unit.
LeftManager0	Kernels.hpp:135	11891 (<1%)	25182 (<1%)	64 (<1%)	451 (<1%)	2 (<1%)	1 compute unit.
LeftManager1	Kernels.hpp:135	11891 (<1%)	25182 (<1%)	64 (<1%)	451 (<1%)	2 (<1%)	1 compute unit.
LeftManager2	Kernels.hpp:135	11891 (<1%)	25182 (<1%)	64 (<1%)	451 (<1%)	2 (<1%)	1 compute unit.
LeftManager3	Kernels.hpp:135	11891 (<1%)	25182 (<1%)	64 (<1%)	451 (<1%)	2 (<1%)	1 compute unit.
PCATripletNN0	Kernels.hpp:251	32720 (2%)	81691 (2%)	447 (3%)	912 (<1%)	91.5 (1%)	1 compute unit.
PCATripletNN1	Kernels.hpp:251	32720 (2%)	81691 (2%)	447 (3%)	912 (<1%)	91.5 (1%)	1 compute unit.
PCATripletNN2	Kernels.hpp:251	32720 (2%)	81691 (2%)	447 (3%)	912 (<1%)	91.5 (1%)	1 compute unit.

Report: Loop Analysis

Loop Analysis ☐ Show blocks

Name	Source Location	Pipelined	Block Scheduled II	Block Estimated fMAX	Latency	Speculated Iterations	Max Itera
Kernel: Coalesce01	Kernels.hpp:415						
Coalesce01.B1	Kernels.hpp:428	Yes	7	480.00	31	0	1
Kernel: Coalesce23	Kernels.hpp:415						
Coalesce23.B1	Kernels.hpp:428	Yes	7	480.00	31	0	1
Kernel: Coalesce57	Kernels.hpp:415						
Coalesce57.B1	Kernels.hpp:428	Yes	7	480.00	31	0	1
Kernel: LeftManager0	Kernels.hpp:135						

Kernels.hpp

```
413 {
414     sycl::event e = q.submit([&](sycl::handler &h)
415                         { h.single_task<TASK>([=]() [[intel
                                ::kernel_args_restrict]] {
416
417 // NEED TO DEAL WITH EOE HERE
418             [[intel::fpga_register]] bool successA = 0;
419             [[intel::fpga_register]] bool successB = 0;
420             [[intel::fpga_register]] bool eoeA = 0;
421             [[intel::fpga_register]] bool eoeB = 0;
422             // [[intel::fpga_register]] bool wait = 0;
423             [[intel::fpga_register]] ResultTriplet A;
424             [[intel::fpga_register]] ResultTriplet B;
425             [[intel::fpga_register]] ResultTriplet tmp;
426             [[intel::fpga_register]] uint8_t sel = 0;
427             int countA =0 , countB =0 , countCa = 0, countCb =0;
428             while(true){
429
430
431                 switch (sel){
432                     case 0 :
433                             A = TripletPipeOut<PIPEID_A>::read(succ
434                             B = TripletPipeOut<PIPEID_B>::read(succ
435                             break;
436                     case 1 :
437                             TripletPipeOut<PIPEID_C>::write(A);
438                             A = TripletPipeOut<PIPEID_A>::read(succ
439                             B = TripletPipeOut<PIPEID_B>::read(succ
440                             break;
441                     case 2 :
442                             TripletPipeOut<PIPEID_C>::write(B);
443                             A = TripletPipeOut<PIPEID_A>::read(succ
```

Details

- Compiler failed to schedule this loop with smaller II due to memory dependency:
  - From: Non-Blocking Pipe Read Operation (handler.hpp: 1166 > Kernels.hpp: 438 > pipes.hpp: 32)
  - To: Non-Blocking Pipe Read Operation (handler.hpp: 1166 > Kernels.hpp: 433 > pipes.hpp: 32, Kernels.hpp: 497)

Report: Loop Analysis

Loop Analysis ☐ Show blocks

Name	Source Location	Pipelined	Block Scheduled II	Block Estimated fMAX			Max Itera
Kernel: Coalesce01	Kernels.hpp:415						
Coalesce01.B1	Kernels.hpp:428	Yes	7	480.00	31	0	1
Kernel: Coalesce23	Kernels.hpp:415						
Coalesce23.B1	Kernels.hpp:428	Yes	7	[illegible]	31	0	1
Kernel: Coalesce57	Kernels.hpp:415						
Coalesce57.B1	Kernels.hpp:428	Yes	7	480.00	31	0	1
Kernel: LeftManager0	Kernels.hpp:135						

II Initiation Interval

7

Kernels.hpp

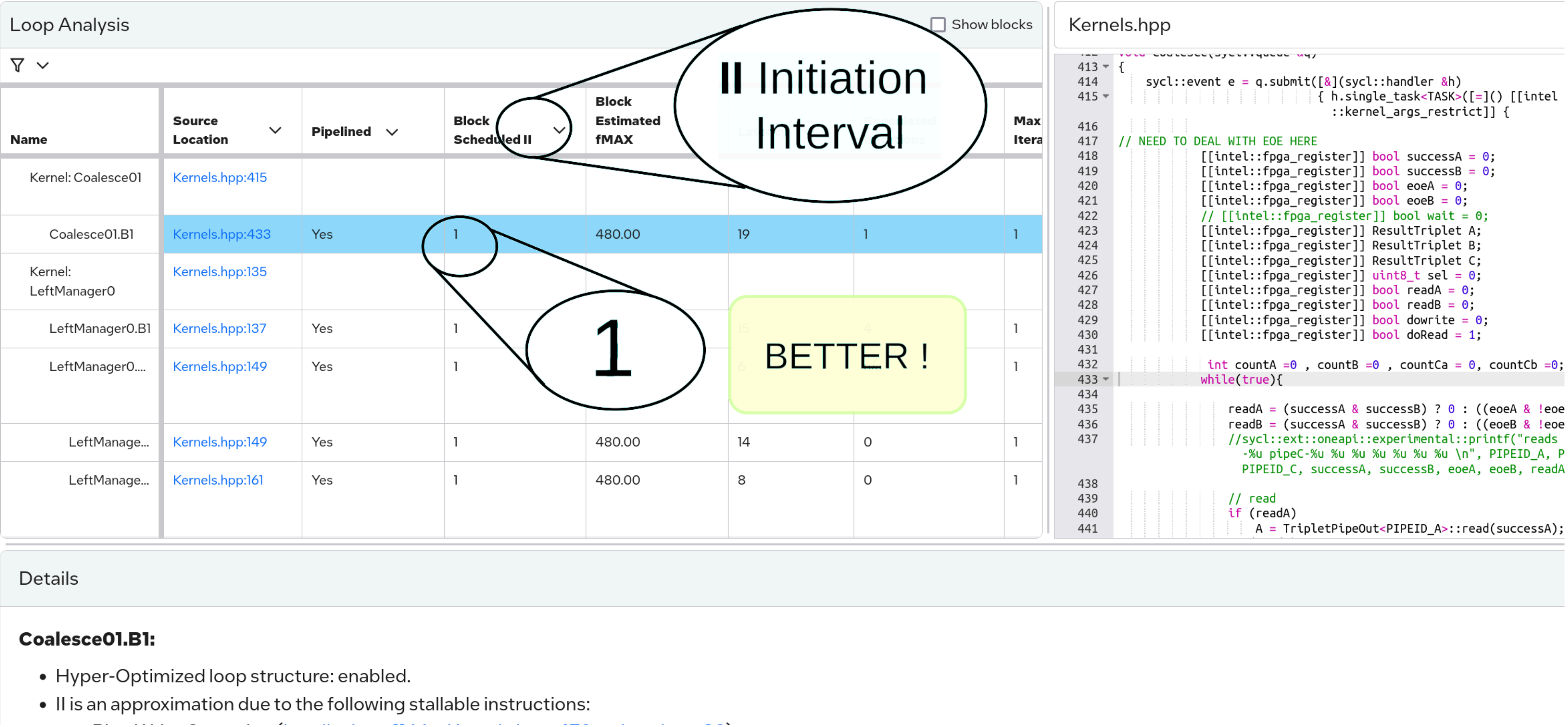
```
413 {
414     sycl::event e = q.submit([&](sycl::handler &h)
415                                 { h.single_task<TASK>([=]() [[intel
                                      ::kernel_args_restrict]] {
416
417 // NEED TO DEAL WITH EOE HERE
418             [[intel::fpga_register]] bool successA = 0;
419             [[intel::fpga_register]] bool successB = 0;
420             [[intel::fpga_register]] bool eoeA = 0;
421             [[intel::fpga_register]] bool eoeB = 0;
422             // [[intel::fpga_register]] bool wait = 0;
423             [[intel::fpga_register]] ResultTriplet A;
424             [[intel::fpga_register]] ResultTriplet B;
425             [[intel::fpga_register]] ResultTriplet tmp;
426             [[intel::fpga_register]] uint8_t sel = 0;
427             int countA =0 , countB =0 , countCa = 0, countCb =0;
428             while(true){
429
430
431                 switch (sel){
432                     case 0 :
433                             A = TripletPipeOut<PIPEID_A>::read(suc
434                             B = TripletPipeOut<PIPEID_B>::read(suc
435                             break;
436                     case 1 :
437                             TripletPipeOut<PIPEID_C>::write(A);
438                             A = TripletPipeOut<PIPEID_A>::read(suc
439                             B = TripletPipeOut<PIPEID_B>::read(suc
440                             break;
441                     case 2 :
442                             TripletPipeOut<PIPEID_C>::write(B);
                                A = TripletPipeOut<PIPEID_A>::read(suc
```

Details

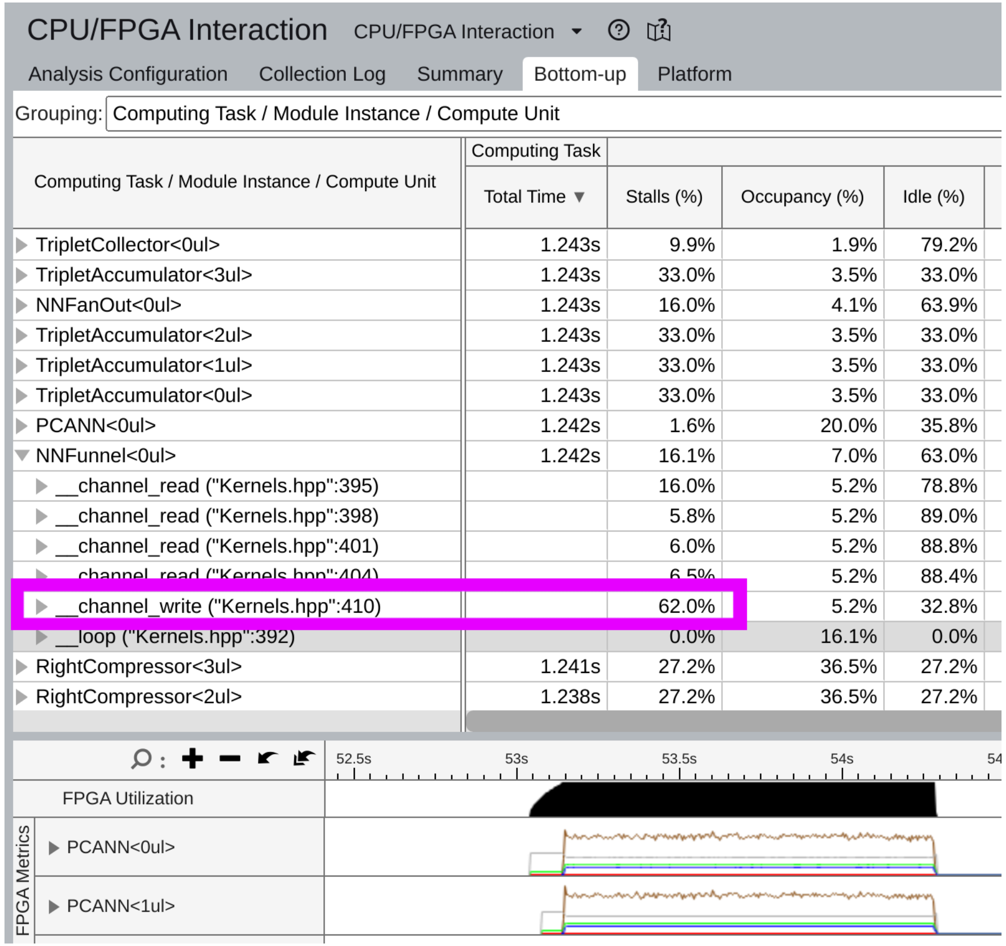
- Compiler failed to schedule this loop with smaller II due to memory dependency:
  - From: Non-Blocking Pipe Read Operation (handler.hpp: 1166 > Kernels.hpp: 438 > pipes.hpp: 32)
  - To: Non-Blocking Pipe Read Operation (handler.hpp: 1166 > Kernels.hpp: 433 > pipes.hpp: 32, Kernels.hpp: 497)

Source of the slowdown

Report: Loop Analysis



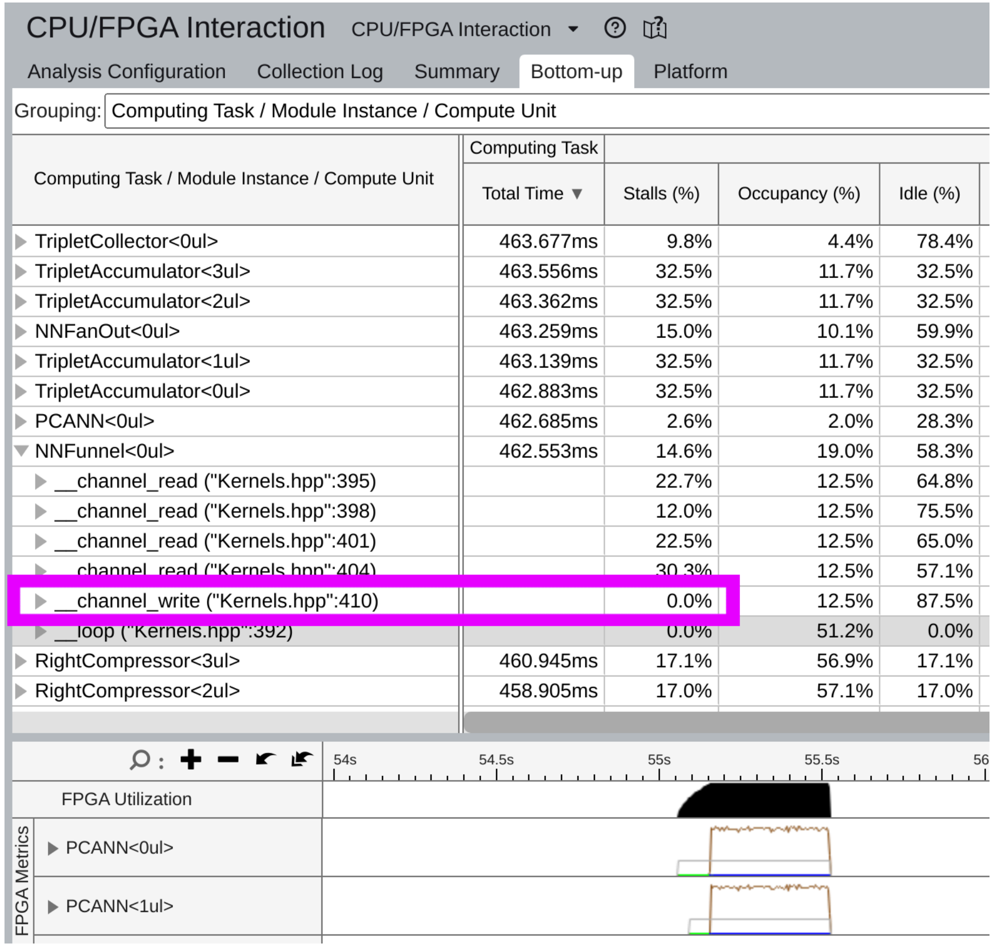
The tools: Intel® VTune™ Profiler



Computing Task / Module Instance / Compute Unit	Computing Task			
	Total Time ▼	Stalls (%)	Occupancy (%)	Idle (%)
TripletCollector<0ul>	1.243s	9.9%	1.9%	79.2%
TripletAccumulator<3ul>	1.243s	33.0%	3.5%	33.0%
NNFanOut<0ul>	1.243s	16.0%	4.1%	63.9%
TripletAccumulator<2ul>	1.243s	33.0%	3.5%	33.0%
TripletAccumulator<1ul>	1.243s	33.0%	3.5%	33.0%
TripletAccumulator<0ul>	1.243s	33.0%	3.5%	33.0%
PCANN<0ul>	1.242s	1.6%	20.0%	35.8%
NNFunnel<0ul>	1.242s	16.1%	7.0%	63.0%
__channel_read ("Kernels.hpp":395)		16.0%	5.2%	78.8%
__channel_read ("Kernels.hpp":398)		5.8%	5.2%	89.0%
__channel_read ("Kernels.hpp":401)		6.0%	5.2%	88.8%
__channel_read ("Kernels.hpp":404)		6.5%	5.2%	88.4%
__channel_write ("Kernels.hpp":410)		62.0%	5.2%	32.8%
__loop ("Kernels.hpp":392)		0.0%	16.1%	0.0%
RightCompressor<3ul>	1.241s	27.2%	36.5%	27.2%
RightCompressor<2ul>	1.238s	27.2%	36.5%	27.2%

- Allows us to profile our kernels
- Simple compiler switch to activate
- Pinpoint what is slowing down the design
- Adds resource usage (scales with design size)
- Optimally a kernel has
  - 100% occupancy
  - 0% stalls

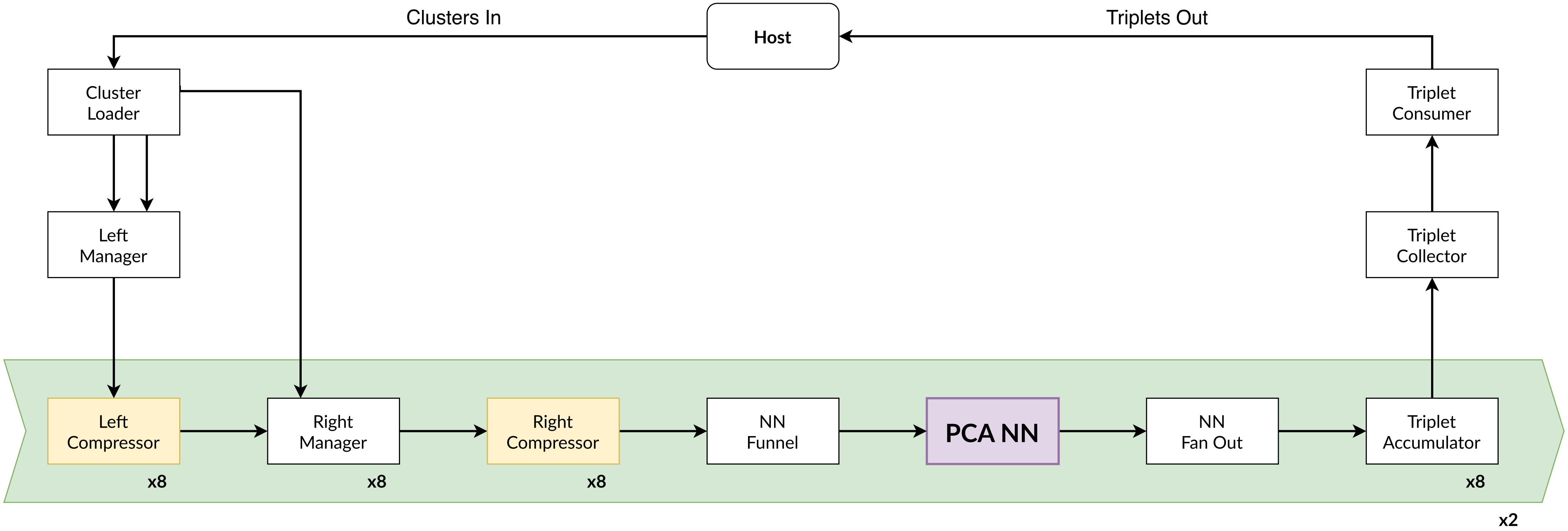
The tools: Intel® VTune™ Profiler



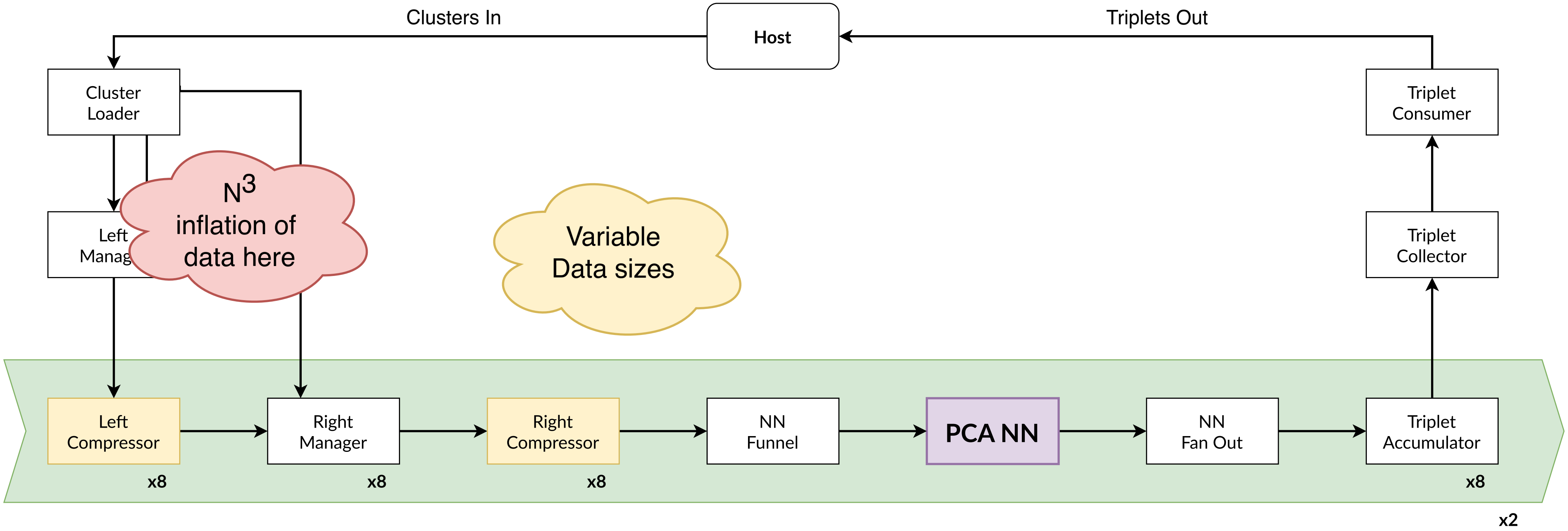
Computing Task / Module Instance / Compute Unit	Computing Task: Total Time ▼	Stalls (%)	Occupancy (%)	Idle (%)
TripletCollector<0ul>	463.677ms	9.8%	4.4%	78.4%
TripletAccumulator<3ul>	463.556ms	32.5%	11.7%	32.5%
TripletAccumulator<2ul>	463.362ms	32.5%	11.7%	32.5%
NNFanOut<0ul>	463.259ms	15.0%	10.1%	59.9%
TripletAccumulator<1ul>	463.139ms	32.5%	11.7%	32.5%
TripletAccumulator<0ul>	462.883ms	32.5%	11.7%	32.5%
PCANN<0ul>	462.685ms	2.6%	2.0%	28.3%
NNFunnel<0ul>	462.553ms	14.6%	19.0%	58.3%
__channel_read ("Kernels.hpp":395)		22.7%	12.5%	64.8%
__channel_read ("Kernels.hpp":398)		12.0%	12.5%	75.5%
__channel_read ("Kernels.hpp":401)		22.5%	12.5%	65.0%
__channel_read ("Kernels.hpp":404)		30.3%	12.5%	57.1%
__channel_write ("Kernels.hpp":410)		0.0%	12.5%	87.5%
__loop ("Kernels.hpp":392)		0.0%	51.2%	0.0%
RightCompressor<3ul>	460.945ms	17.1%	56.9%	17.1%
RightCompressor<2ul>	458.905ms	17.0%	57.1%	17.0%

- Allows us to profile our kernels
- Simple compiler switch to activate
- Pinpoint what is slowing down the design
- Adds resource usage (scales with design size)
- Optimally a kernel has
  - 100% occupancy
  - 0% stalls

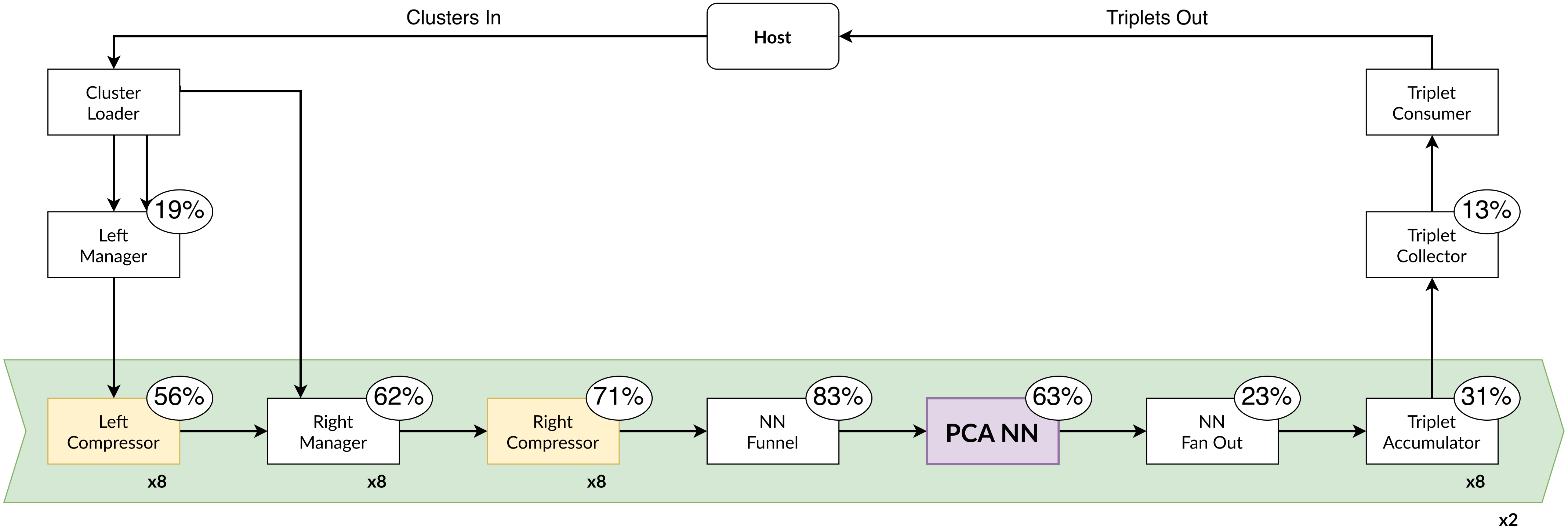
Current Design



Current Design



Current Design



- Occupancy per kernel; trying to keep the workers busy

The experience - the good

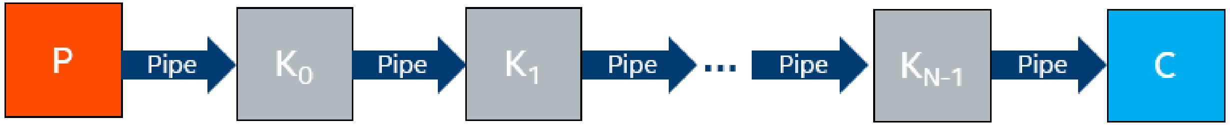
- The **tools are very powerful**
- Documentation is good - a lot of good information on correct approaches to coding, and pitfalls to avoid.
    - Several training videos too
- We have **direct support from Intel®/Altera** and weekly meetings with FAE Christian Faerber to help guide us and give feedback on the tools
    - Collaborating with **LHCb Online** who are also evaluating oneAPI
- Similarly, **BittWare** have been responsive on improving the BSP and install issues
- Getting to grips with the framework and **turn around time is fast** - about a month to get the first NN working on the hardware
- Can separate host code into a library, and change it without an FPGA recompile

The experience - the not so good

- *Some things work and some don't* for non-obvious reasons
  - Code works fine on the emulator but then stalls on the HW
- The compiler sometimes *fails to optimise code* that should be fairly straightforward to convert to RTL
  - Switch statement -> multiplexor
- We're are using oneAPI 2023.1 for compatibility with the BittWare BSP - some of these issues may already be resolved

The experience - the realistic

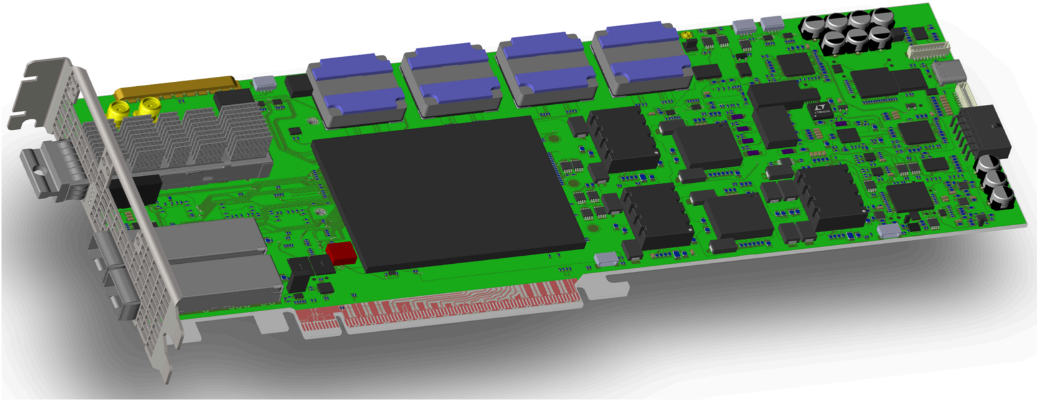
- You need to understand the hardware - **you can't blindly code without knowing how FPGAs work**
- Example: Pipelining
  - **run lots different tasks in a chain**, like an assembly line
  - optimise by keeping all the workers busy
  - Can **parallelise** by having multiple independent pipelines



- The product is fairly young, but **it does work** once you get past some of the pimples

The future

- **PCIe400**
  - **Next Gen readout card** for LHCb
  - **Approx. 2**$\times$ performance of the AGF-027 (10nm)
  - Requires development of a Board Support Package



- IP Flow - **Develop an algorithm with oneAPI and export as an IP**
  - Integrate with an existing RTL design
    - Potentially best of both worlds
    - Feed DAQ output to Tracking algorithm
- **Comparison with GPU** (also in oneAPI) - performance and cost
- **Scale** to future luminosities; other detectors

Final Thoughts

- We're looking at this technology as a means to leverage compute acceleration hardware from industry
    - The **algorithm is designed for HEP tracking detectors** - nothing is specific to LHCb
- This is about four months work by two people
    - We still have many avenues to explore before declaring a true performance number
        - reduced precision, IP Flow, increased Fmax etc.
    - But in that time, **we've managed to put an ML Tracking algorithm on an FPGA**
- There is **potential to make this hardware accessible to a new set of developers**
    - Already need compute acceleration for today's experiments
    - Need to evaluate where to put our money and our training

Thanks for listening !

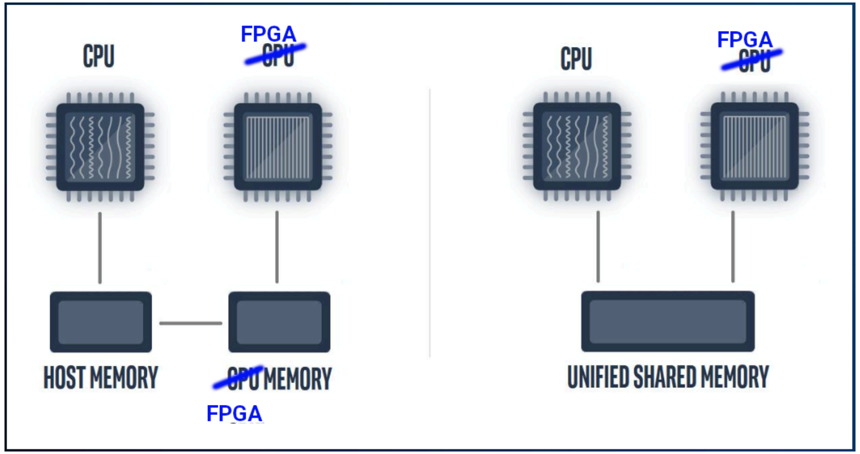
Current Resource Usage

- Logic (ALMs) : 75%
- RAMs (M20k) : 25%
- DSPs : 14%
- Clock rate : 310 MHz
- 2 NN inferences
- all data and calculation is currently float32

Hardware specificities

- versions
  - Intel® oneAPI 2023.1
  - Intel® Quartus® 2023.1
  - BittWare BSP 2023.1

Unified Shared Memory



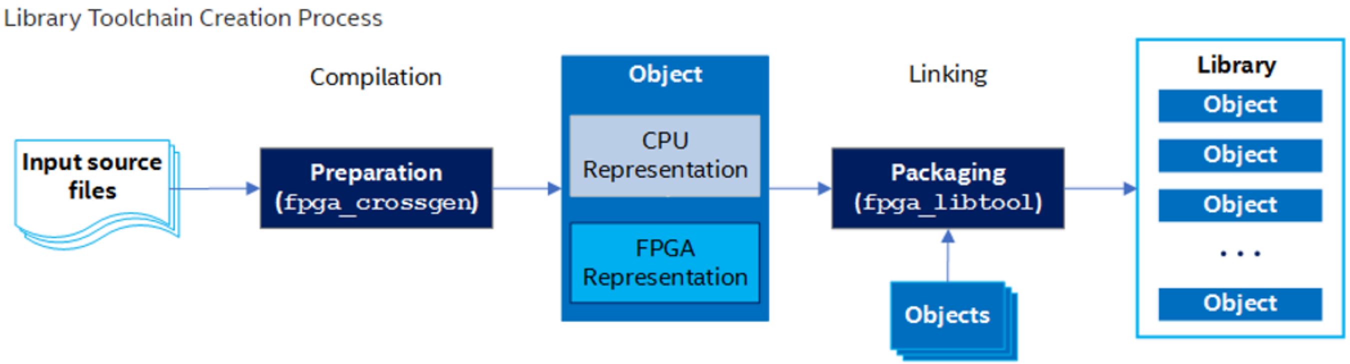
- USM is a language feature of SYCL
- Can define memory:
  - on host and access on FPGA/GPU
  - on FPGA/GPU and access on host
  - shared - accessible on both

Use of RTL Libraries for FPGA in oneAPI

- Create a static library file using RTL
  - fpga_crossgen: RTL -> object
  - fpga_libtool: objects -> library

  Files needed:
    - RTL wrapper
    - XML description
    - Emulation model file (SYCL-based)

  RTL design constraints:
    - RTL module must have a clock port, a resetn port, and Avalon® streaming interface input and output ports
    - A single pair of ready and valid logic must control all the inputs
    - Declare the RTL module as stall-free possible



- Include library file to use the functions inside your SYCL* kernels.

```
dpcpp -fintelfpga main.cpp lib.a
```