FPGA Tracking with oneAPI

Karol Hennessy, Kurt Rinnert

24th April 2024

UNIVERSITY OF LIVERPOOL

Overview

- The plan
- The algorithm
- The oneAPI framework
- Our experience
- Current implementation and performance
- Future work and final thoughts

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

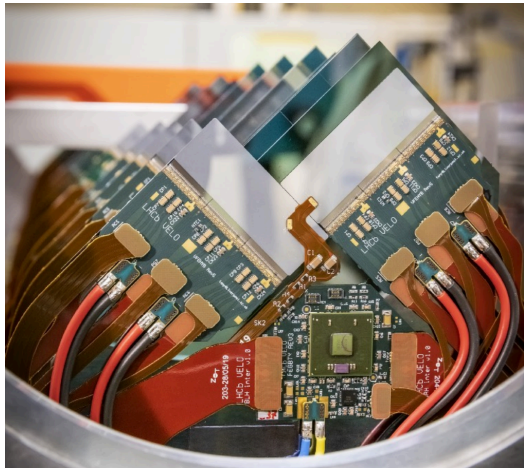
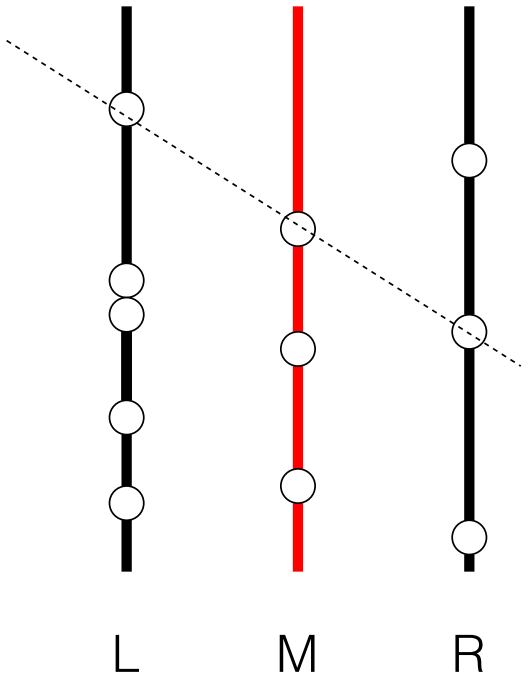


Figure 1: LHCb VELO

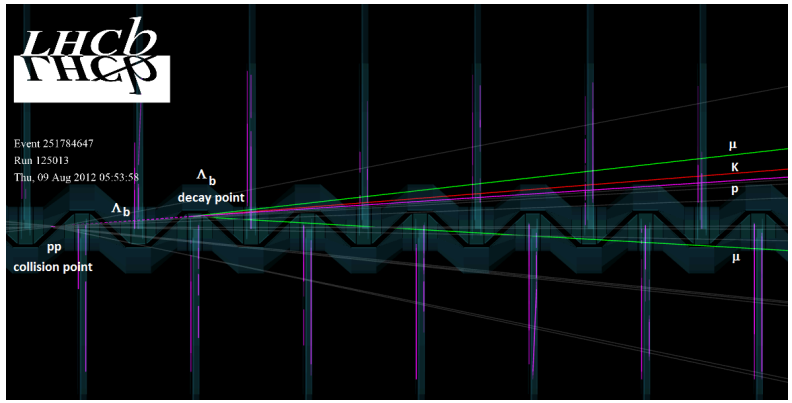
The Plan

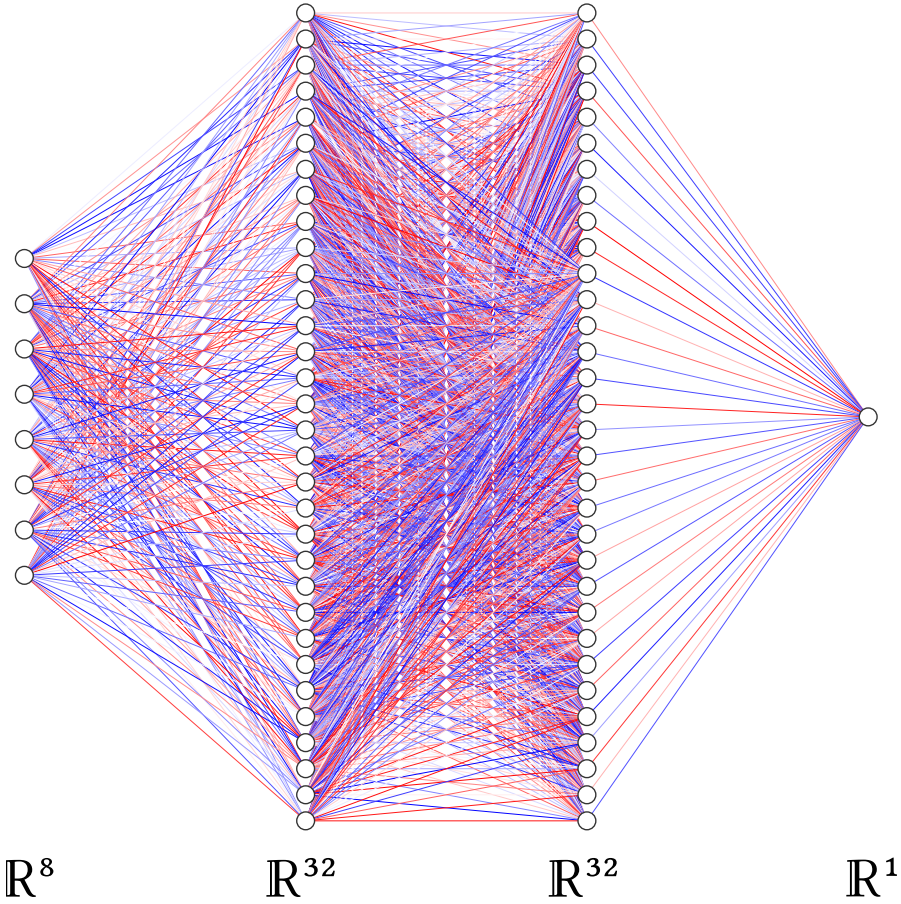
- Use LHCb VELO as a test-bed.
- Connect the dots
  - Take 3 detector planes (**L**eft, **M**iddle, **R**ight)
  - **make all combinations of triplets from hit clusters**
  - **choose the “best” triplet**
  - connect triplets and make track fit
- Replace the steps in conventional algorithms with learned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



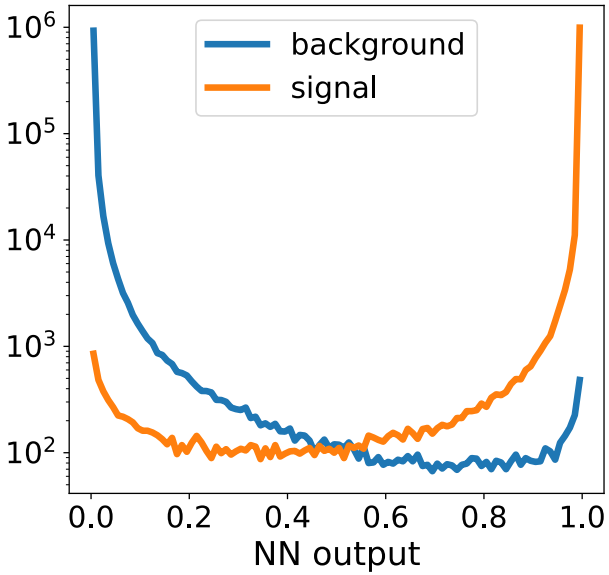
The algorithm

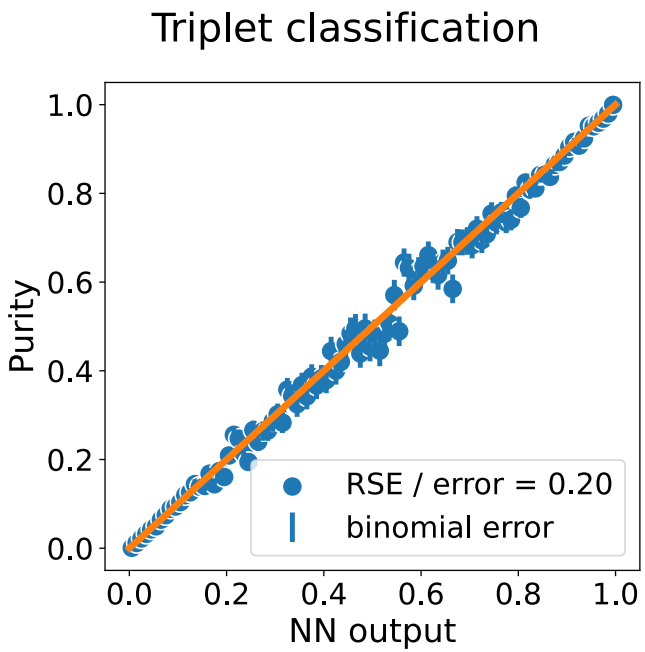
- NN developed using PyTorch
- Homebrew - developed for HEP
- Using Monte-Carlo data from LHCb
  - Inclusive-b & Minimum Bias
- Check the resultant tracklets vs. MC-truth

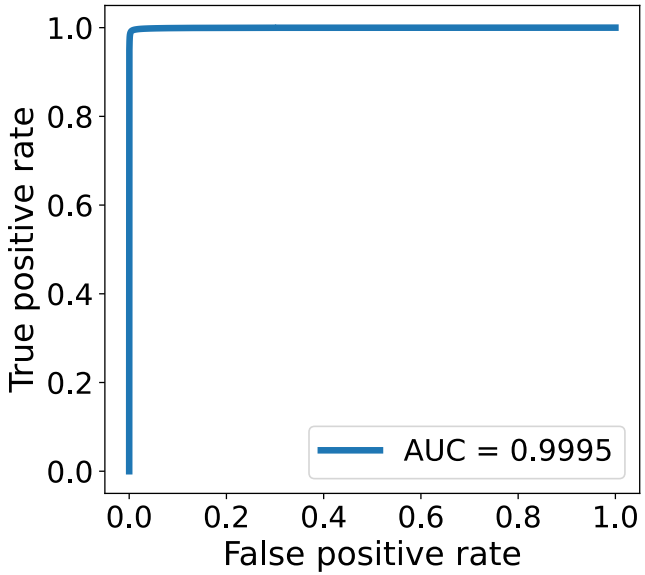




Network Performance

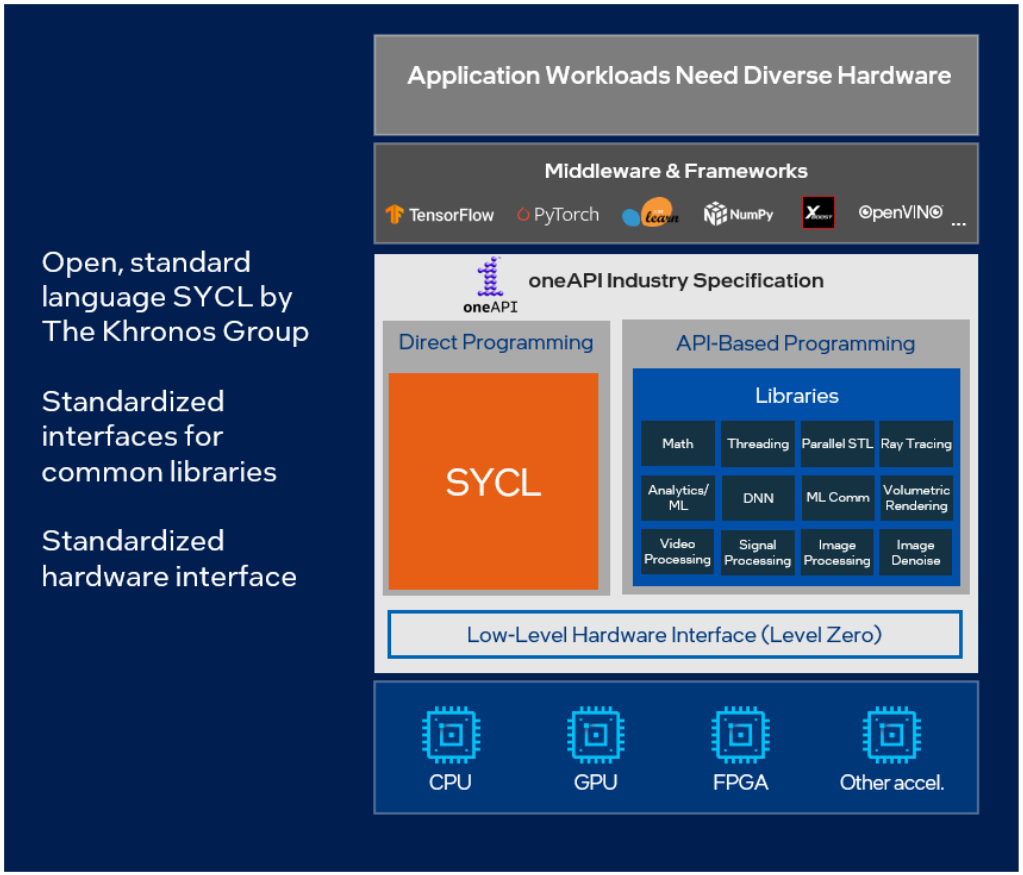






- non-linear activation between input and hidden layers is ReLU
- Performance: high efficiency ~92%, purity ~97%, low ghost rates

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

- **Currently deployed in LHCb testbed**
- 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++)
  - *Expect to add student in October*
- **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

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		510662 (28%)	981925 (27%)	2483.6 (19%)	6476 (7%)	1472 (17%)	
Board interface		268666	537332	1536	0	778	Platform interface logic.
Pipe resources		1122 (<1%)	82548 (2%)	696 (5%)	52 (<1%)	0 (0%)	
EOECounter<0ul>	Kernels.hpp:663	1816 (<1%)	2336 (<1%)	0 (0%)	15 (<1%)	0 (0%)	1 compute unit.
LeftManager<0ul>	Kernels.hpp:174	12849 (<1%)	30308 (<1%)	16 (<1%)	476 (<1%)	0 (0%)	1 compute unit.
RightManager<0ul>	Kernels.hpp:281	7325 (<1%)	13767 (<1%)	16 (<1%)	245 (<1%)	0 (0%)	1 compute unit.
RightManager<1ul>	Kernels.hpp:281	7325 (<1%)	13767 (<1%)	16 (<1%)	245 (<1%)	0 (0%)	1 compute unit.
RightManager<2ul>	Kernels.hpp:281	7325 (<1%)	13767 (<1%)	16 (<1%)	245 (<1%)	0 (0%)	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		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(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);
                            A = TripletPipeOut<PIPEID_A>::read(succ
```

Source of the slowdown

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:433	Yes	1	480.00	19	1	1
Kernel: LeftManager0	Kernels.hpp:135						
LeftManager0.B1	Kernels.hpp:137	Yes	1		15	4	1
LeftManager0....	Kernels.hpp:149	Yes	1		4		1
LeftManage...	Kernels.hpp:149	Yes	1	480.00	14	0	1
LeftManage...	Kernels.hpp:161	Yes	1	480.00	8	0	1

II Initiation Interval

1

BETTER !

Kernels.hpp

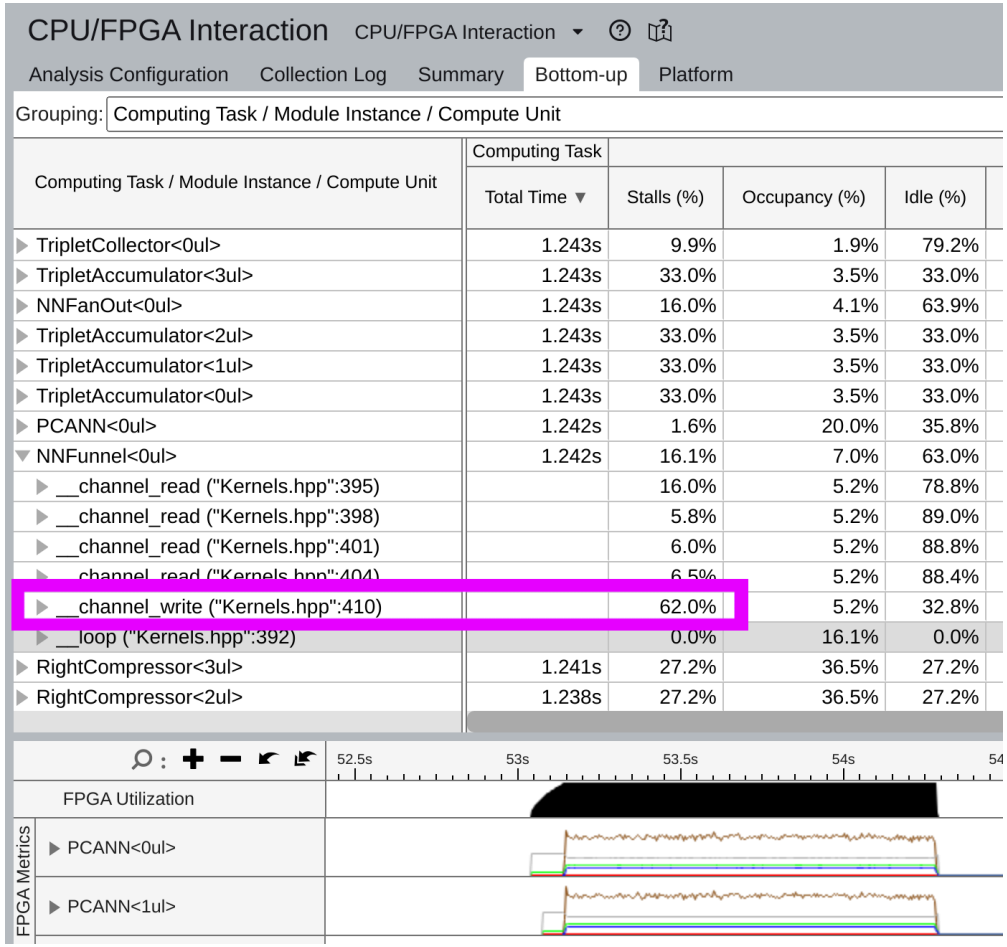
```
413 {
414     sycl::event e = q.submit([&](sycl::handler &h)
415                              { h.single_task<TASK>([=]() [[intel
                                    ::kernel_args_restrict]] {
416
417 // NEED TO DEAL WITH EOE HERE
418             [[intel::fpga_register]] bool successA = 0;
419             [[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 C;
426             [[intel::fpga_register]] uint8_t sel = 0;
427             [[intel::fpga_register]] bool readA = 0;
428             [[intel::fpga_register]] bool readB = 0;
429             [[intel::fpga_register]] bool dowrite = 0;
430             [[intel::fpga_register]] bool doRead = 1;
431
432              int countA =0 , countB =0 , countCa = 0, countCb =0;
433             while(true){
434
435                 readA = (successA & successB) ? 0 : ((eoeA & !eoe
436                 readB = (successA & successB) ? 0 : ((eoeB & !eoe
437                 //sycl::ext::oneapi::experimental::printf("reads
                      -%u pipeC-%u %u %u %u %u %u \n", PIPEID_A, P
                      PIPEID_C, successA, successB, eoeA, eoeB, readA
438
439                 // read
440                 if (readA)
441                     A = TripletPipeOut<PIPEID_A>::read(successA);
```

Details

Coalesce01.B1:

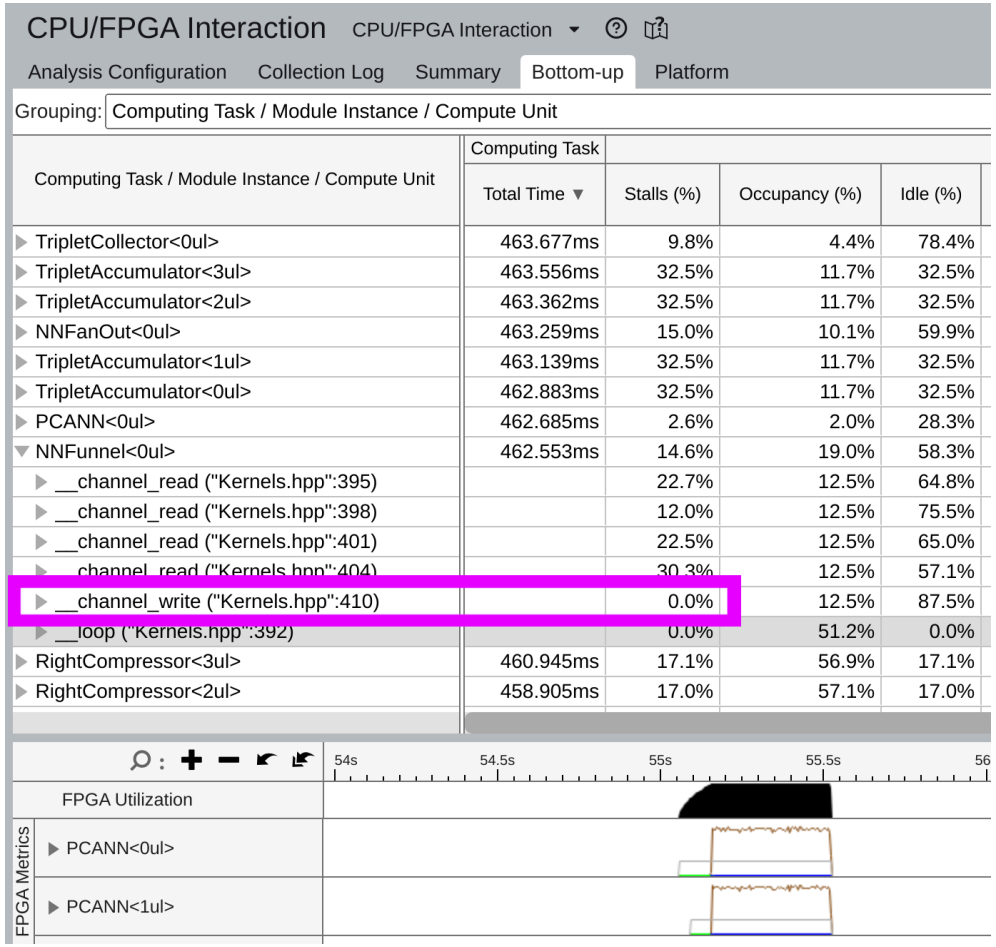
- Hyper-Optimized loop structure: enabled.
- II is an approximation due to the following stallable instructions:

The tools: Intel® VTune™ Profiler



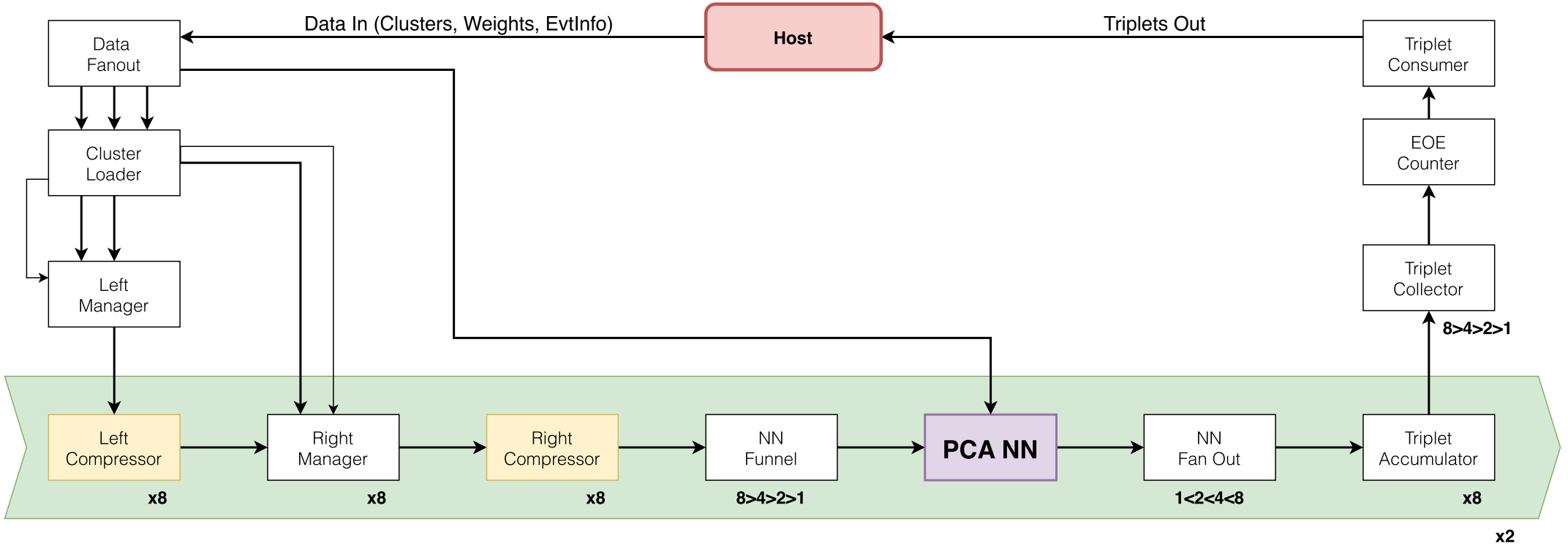
- Allows us to profile our kernels
- Simple compiler switch to activate
- Pinpoint what is slowing down the design
- Adds resource usage (scales with design size)
- Optimally a kernel has
  - 100% occupancy
  - 0% stalls

The tools: Intel® VTune™ Profiler



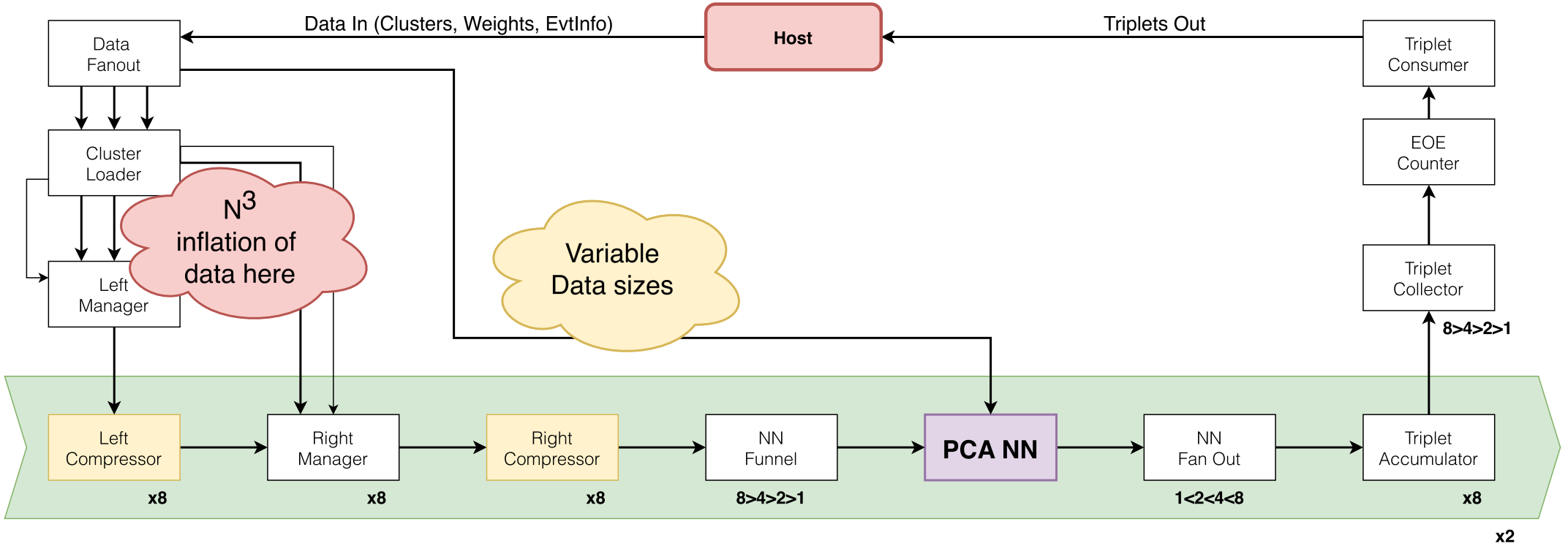
- Allows us to profile our kernels
- Simple compiler switch to activate
- Pinpoint what is slowing down the design
- Adds resource usage (scales with design size)
- Optimally a kernel has
  - 100% occupancy
  - 0% stalls

Current Design



- Many of the workers are to optimise dataflow

Current Design



- Many of the workers are to optimise dataflow

Current Performance

- Bandwidth utilisation ~1.1 GB/s (limit ~20 GB/s by PCIe)
- Clock frequency (Fmax) 358 MHz (target 480 MHz)
  - Scope for improvement
- Resource Usage

	User	Base System	Total (estimated)	Compiled
Logic (ALMs)	13%	15%	28%	52%**
RAMs	8%	11%	19%	11%
DSPs	8%	9%	17%	8%

** Unclear why such disagreement between compiled and estimated resource usage (some extra included for profiling)

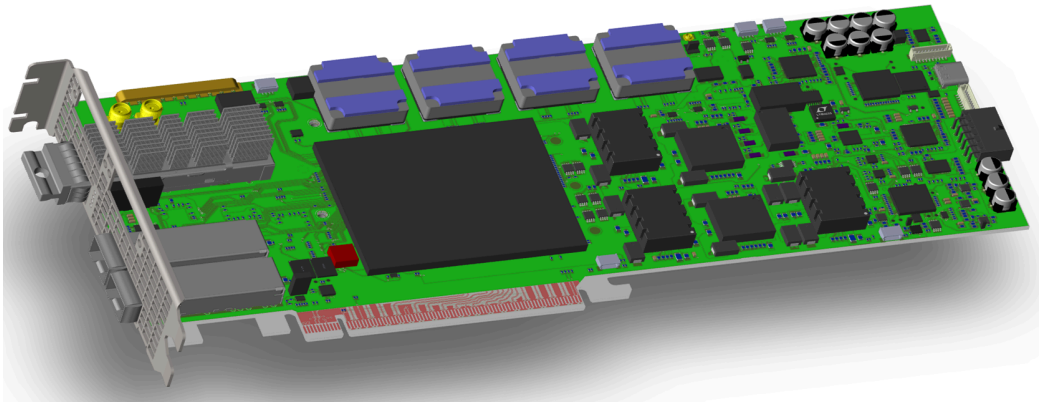
- Removing “Base System” should allow more processing units

The experience

- The good
  - The **tools are very powerful**
  - Documentation is good - correct approaches, pitfalls, training videos
  - **Direct support from Intel®/Altera** (weekly meetings) and **BittWare**
  - **turn around time is fast** - ~1 month to first NN inference on hardware
- 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 straightforward
    - Switch statement -> multiplexor
- The realistic
  - **you can't blindly code without knowing how FPGAs work**

Next steps

- IP Flow - **Export the oneAPI algorithm 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
- **PCIe400**
  - UII readout card
  - ~2× performance of the AGF-027
  - Requires development of a Board Support Package



- **Scale** to future luminosities; other detectors

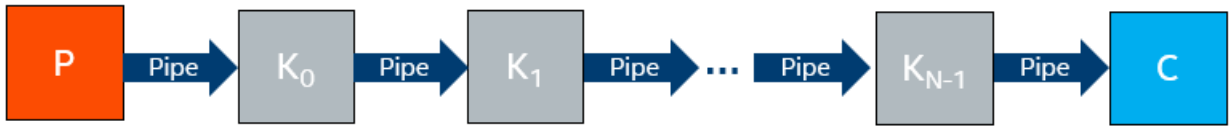
Final Thoughts

- 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
- In short time, **we've managed to put an ML Tracking algorithm on an FPGA**
- Still many avenues to explore before declaring a true performance number
  - reduced precision, IP Flow, increased Fmax etc.
- 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 !

The experience - the realistic

- 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

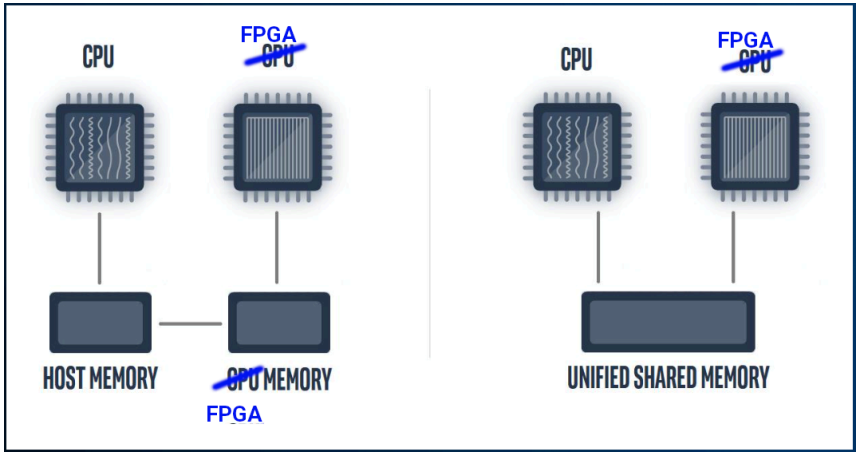


- The product is fairly young, but **it does work** once you get past some of the pimples

Hardware specificities

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

Unified Shared Memory

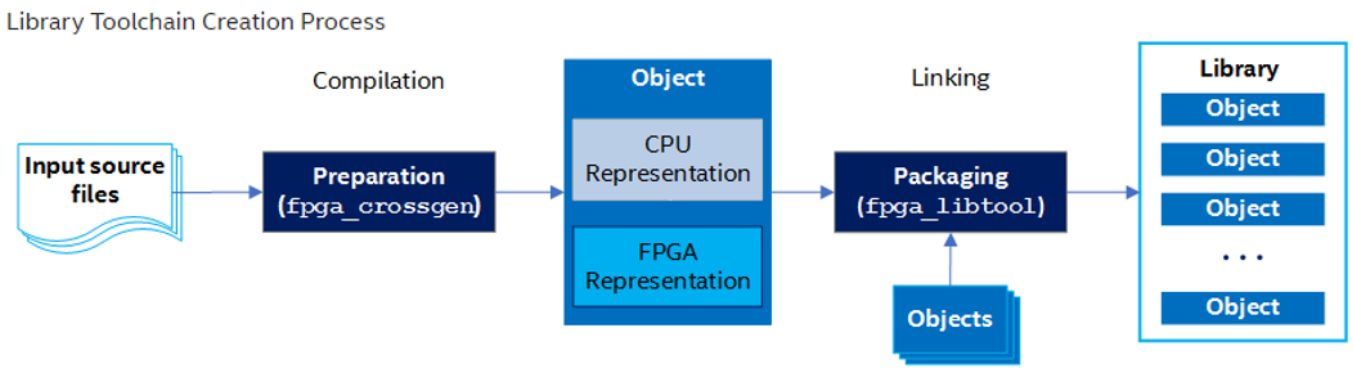


- USM is a language feature of SYCL
- Can define memory:
  - on host and access on FPGA/GPU
  - on FPGA/GPU and access on host
  - shared - accessible on both

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
```