Reinforcement of HLT with GPU

Ryosuke Itoh, KEK

Belle II Trigger/DAQ Workshop 2025

Osaka Metropolitan University
Oct.24, 2025

Consideration on further HLT reinforcement strategy

- HLT reinforcement should be considered from two separate aspects:

1) Mitigation for the higher trigger rate (by the luminosity/background increase) and the maintenance of existing HLT units **until LS2.**

2) Preparation for the new detector configuration **after LS2**.

- For 1), we have a baseline upgrade plan.
  = adding more PC servers to the existing 15 HLT units.
  -> Is this really enough?

- For 2), we need to consider the change in VTX readout
  * **We don't need to manage RoI feedback any more.**
  * Possible increase in the event size. (VTX data go through EB1).

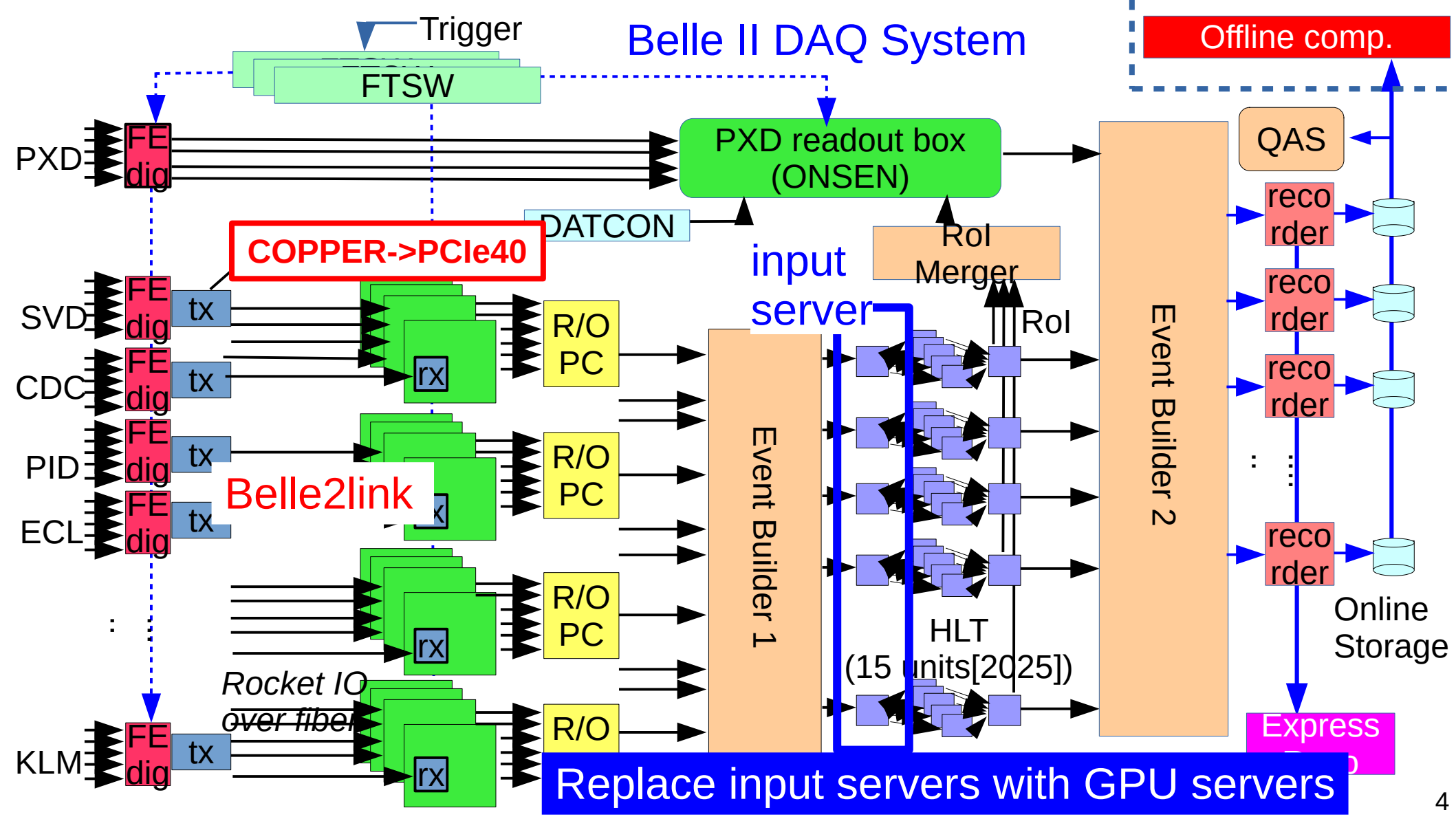
Proposal for 1) : Addition of GPU based Level 3 trigger

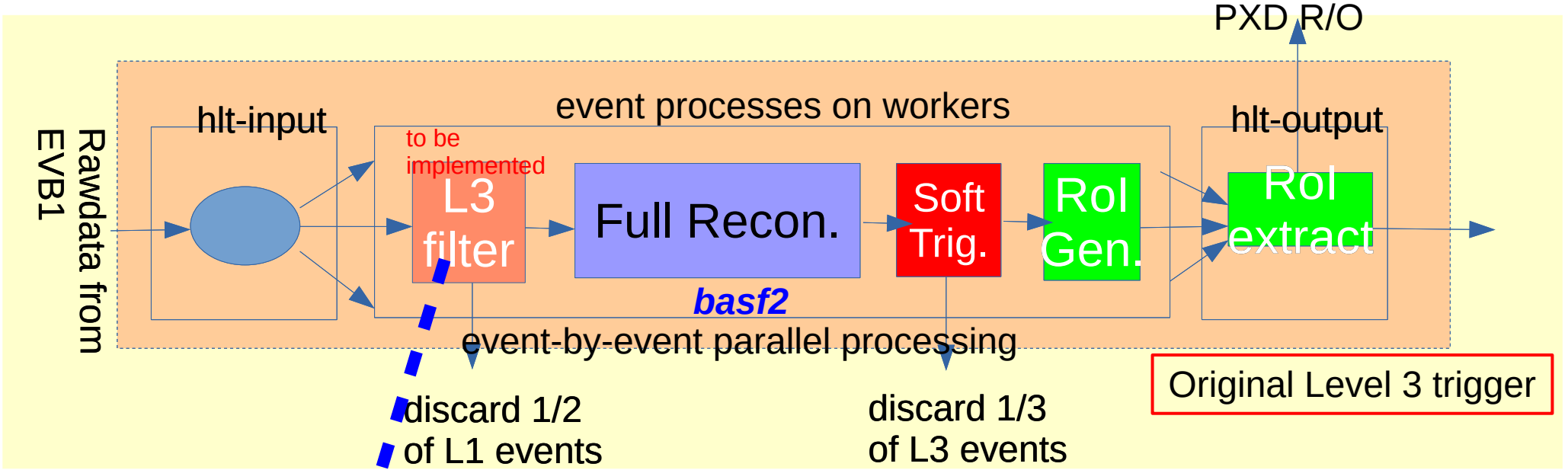
- Offload the Level3 trigger software (to be implemented in current HLT) to GPU.

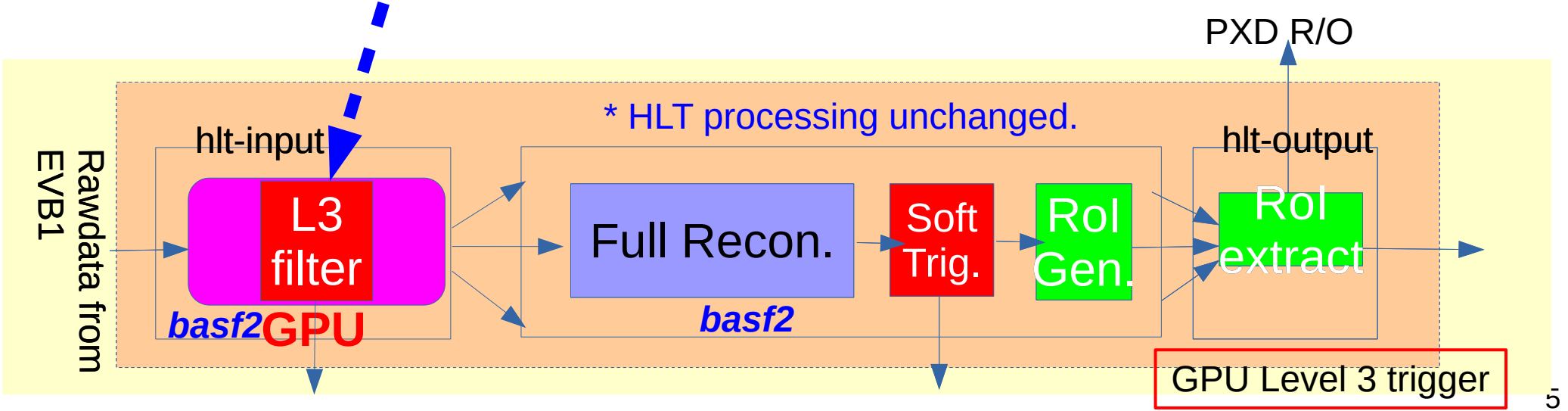
* GPU can provide tremendous number of cores at a much lower price.
* GPU can be programmed using a language CUDA which is mature and compatible with C++. <= *easy to keep compatibility with Belle2 soft.*
* GPU works as a co-processor for CPU
  -> mixed CPU+GPU processing is possible.

=> Integrate GPU processing in basf2 and build a seamless software development environment for Level3 software.

Idea: Implement GPU as soon as possible as an alternative to the baseline upgrade plan.


Belle II DAQ System
Trigger
FTSW
Offline comp.
PXD
FE dig
PXD readout box (ONSEN)
QAS
DATCON
COPPER->PCIe40
input server
RoI Merger
RoI
SVD
CDC
PID
ECL
tx
rx
R/O PC
Belle2link
Event Builder 1
Event Builder 2
reco rder
HLT (15 units[2025])
Online Storage
Rocket IO over fiber
KLM
R/O
Express
Replace input servers with GPU servers


PXD R/O
event processes on workers
hlt-input
to be implemented
L3 filter
Full Recon.
Soft Trig.
RoI Gen.
hlt-output
RoI extract
basf2
event-by-event parallel processing
Rawdata from EVB1
discard 1/2 of L1 events
discard 1/3 of L3 events
Original Level 3 trigger


PXD R/O
* HLT processing unchanged.
hlt-input
L3 filter
basf2GPU
Full Recon.
Soft Trig.
RoI Gen.
hlt-output
RoI extract
basf2
Rawdata from EVB1
GPU Level 3 trigger

Why GPU? : Comparison with FPGA-based HLT reinforcement

- Another candidate implementation for HLT reinforcement is the use of FPGA-based co-processor (just like Xilinx Versal).

- FPGA is useful when a very short latency is required (O(μsec)).
  <-> Required HLT latency is O(sec).....

- The number of cores in one FPGA (Versal : AI engines, which is essentially RISC processors) is much less than that of GPU.
  GPU (CUDA cores) : > 10,000, Versal (AI cores) : up to 400.

- Programming of FPGA with HLS is much difficult compared to that for GPU with CUDA. (CUDA is widely used and mature.)
  [Assuming the porting of existing C++ code on CPU.]
  ** Vector cores and DSPs on FPGA are not usable for normal C++ codes w/o special programming technique.*

🔧 GPU vs. Versal (FPGA) – Running Standard C++ Applications in Parallel

ChatGPT

	GPU (RTX 5000 Ada / 6000 Ada / 5090)	Versal AI Core / AI Edge
Number of Parallel Cores	10,000+ CUDA cores (12,800 on RTX 5000 Ada, 18,176 on RTX 6000 Ada)	Hundreds of scalar cores + vector units
Programming Model	CUDA C/C++ (relatively easy to port existing C++ code)	Vitis + HLS or RTL (requires rewriting C++ into HLS/RTL)
Throughput	Tens of TFLOPS in FP32, strong FP64 performance	Lower overall FP32/FP64 throughput (limited number of AI Engines/tiles)
Latency	ms–hundreds of ms (can be hidden with streams/pipelining)	µs–tens of µs (excellent for ultra-low latency)
Performance per Dollar	◎ (very high for non-ML workloads)	△ (flexible but more expensive per FLOP)
Development Cost	Low (CUDA toolchain is mature, lots of documentation)	High (RTL/HLS design and verification required)
Best Use Case	**Massively parallel SIMD, FP32/FP64 HPC, batch processing, DAQ event analysis**	Ultra-low latency, deterministic pipelines, custom logic

製品仕様

プロセッサ サブシステムの機能

	VC1502	VC1702	VC1802	VC1902	VC2602	VC2802
アプリケーション プロセッシング ユニット	デュアル コア Arm® Cortex®-A72、48 KB/32 KB L1 キャッシュ (パリティおよび ECC 付き)、1 MB L2 キャッシュ (ECC 付き)					
リアルタイム プロセッシング ユニット	デュアル コア Arm Cortex-R5F、32 KB/32 KB L1 キャッシュ、および 256 KB TCM (ECC 付き)					
メモリ	256 KB オンチップ メモリ (ECC 付き)					
接続性	イーサネット (x2)、USB 2.0 (x1)、UART (x2)、SPI (x2)、I2C (x2)、CAN-FD (x2)					

AI エンジンと DSP エンジンの機能

	VC1502	VC1702	VC1802	VC1902	VC2602	VC2802
AI エンジン	198	304	300	400	0	0
AI エンジン-ML	0	0	0	0	152	304
DSP エンジン	1,032	1,312	1,600	1,968	984	1,312

Rough Cost Comparison

- With the M&O budget per year allocated for HLT maintenance,

1) Baseline upgrade: we can buy up to 15 PC servers (Xeon 6548N).
   -> 64(cores/server) x 15 = 960 cores.
   => ~10% increase in processing power

2) GPU (RTX5000Ada) : we can buy up to 15 GPU units.
   -> 12,448 (cores/GPU) x 15 = 186,720 cores! => ~2,500%!!

3) FPGA (Versal VC1902) : we can buy up to 4 Versals(?)
   -> 400 (cores/AIengine) x 4 = 1,600 cores. => ~20%

Design Policy

- Keep existing HLT as is.
  Current HLT software processing is not touched at all.

- Add GPU processing before the existing HLT framework as a part of the HLT system.
  * **Level 3 trigger software incorporated from existing C++ codes is implemented in GPU.**
  * Filter out background events before the actual HLT processing.

- Build GPU software development environment fully compatible with the Belle II software library.
  “Seamless software environment between CPU and GPU”

GPU-L3 project

1. Seamless Software Development Environment

- The software on GPU is desired to be compatible with the existing Belle2 software coded mostly in C++.

- GPU has to be programmed using a different language, CUDA.

- The Belle2 software is developed using “scons” and runs on a framework called “basf2” as a set of dynamically-linked modules.

- CUDA compiler (nvcc) is added to scons.
- Development environment is built so that mixed C++ and CUDA codes can co-exist in Belle2 library and dynamically loaded in basf2.

=> Seamless execution of GPU-modules and existng C++ modules on basf2 becomes possible.

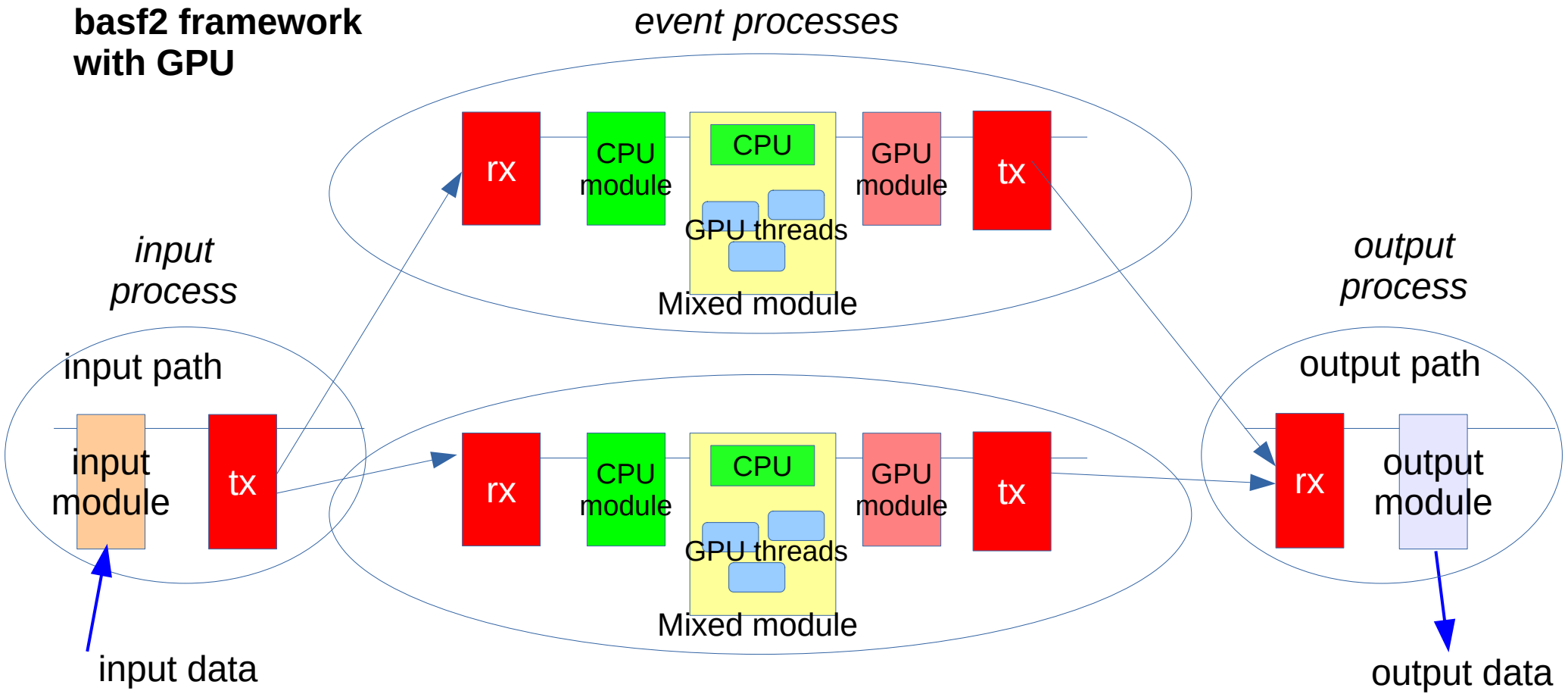
** Applicable in the offline use.*

GPU package in Belle2 library

```
itohepyc% tree -df gpu
gpu
|-- gpu/cdctracker
|   |-- gpu/cdctracker/include
|   `-- gpu/cdctracker/src
|-- gpu/cdcunpack
|   |-- gpu/cdcunpack/include
|   `-- gpu/cdcunpack/src
|-- gpu/eclunpack
|-- gpu/modules
|   |-- gpu/modules/cudatest
|   |   |-- gpu/modules/cudatest/include
|   |   `-- gpu/modules/cudatest/src
|   |-- gpu/modules/eventhandle
|   |   |-- gpu/modules/eventhandle/include
|   |   `-- gpu/modules/eventhandle/src
|   |-- gpu/modules/gpuunpacker
|   |   |-- gpu/modules/gpuunpacker/include
|   |   `-- gpu/modules/gpuunpacker/src
|   `-- gpu/modules/l3unpack
|       |-- gpu/modules/l3unpack/include
|       `-- gpu/modules/l3unpack/src
|-- gpu/rawdata
|   |-- gpu/rawdata/include
|   `-- gpu/rawdata/src
|-- gpu/tools
`-- gpu/utils
    `-- gpu/utils/include
```

```
itohepyc% scons -j 1 gpu
scons: Reading SConscript files ...
........
######> Targets (Detailed):
Target: gpu
scons: Building targets ...
nvcc -o build/Linux_x86_64/debug/gpu/modules/l3decision/src/GPUSRootInput.o -c \
--compiler-options -fPIC -arch sm_86 -rdc=true -Xcompiler -Wno-unused-parameter\
 -Xcompiler -std=c++17 -Xcompiler -Wall -Xcompiler -isystem/home/usr/itoh/belle\
2/externals/v02-00-02b/include -Xcompiler -Iinclude/ -Xcompiler -Wextra -Xcompi\
ler -Wshadow -Xcompiler -Wstack-usage=200000 -Xcompiler -g -Xcompiler -isystem/\
home/usr/itoh/belle2/externals/v02-00-02b/Linux_x86_64/common/include/python3.8\
 -Xcompiler -isystem/home/usr/itoh/belle2/externals/v02-00-02b/include/CLHEP -X\
compiler -isystem/home/usr/itoh/belle2/externals/v02-00-02b/Linux_x86_64/common\
/include/Geant4 -Xcompiler -isystem/home/usr/itoh/belle2/externals/v02-00-02b/L\
inux_x86_64/common/include -Xcompiler -isystem/home/usr/itoh/belle2/externals/v\
02-00-02b/include/root -Xcompiler -isystem/home/usr/itoh/belle2/externals/v02-0\
0-02b/include/belle_legacy -D_PACKAGE_='"gpu"' -DG4UI_USE_TCSH -DRaveDllExport=\
 -DHAS_SQLITE -DHAS_CALLGRIND -DHAS_OPENMP -I include -I /home/usr/itoh/belle2/\
externals/v02-00-02b/Linux_x86_64/common/include/libxml2 gpu/modules/l3decision\
/src/GPUSRootInput.cu
.....
nvcc -shared -arch sm_86 -o modules/Linux_x86_64/debug/libl3decision.so build/L\
inux_x86_64/debug/gpu/modules/l3decision/src/GPUSRootInput.o build/Linux_x86_64\
/debug/gpu/rawdata/src/GPURawCOPPER.o build/Linux_x86_64/debug/gpu/rawdata/src/\
GPURawHeader.o build/Linux_x86_64/debug/gpu/rawdata/src/GPURawTrailer.o build/L\
inux_x86_64/debug/gpu/cdcunpack/src/GPUCDCHit.o build/Linux_x86_64/debug/gpu/cd\
cunpack/src/GPUCDCUnpack.o build/Linux_x86_64/debug/gpu/cdcunpack/src/GPUWireID\
.o -Llib/Linux_x86_64/debug -L/home/usr/itoh/belle2/externals/v02-00-02b/Linux_\
x86_64/opt/lib -L/home/usr/itoh/belle2/externals/v02-00-02b/Linux_x86_64/common\
/root/lib -L/usr/local/cuda/usr/local/cuda/targets/x86_64-linux/lib/stubs -L/us\
r/local/cuda/targets/x86_64-linux/lib -L/home/usr/itoh/belle2/externals/v02-00-\
02b/Linux_x86_64/common/lib -lcdc -ldaq -lframework -lrawdata_dataobjects -lcud\
adevrt -lcudart_static
*** map        : modules/Linux_x86_64/debug/libl3decision.b2modmap
scons: done building targets.
```

basf2 framework with GPU

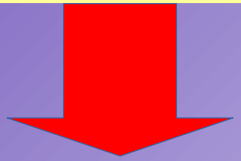


parallel processing utilizing multi-core CPU

2. GPU implementation in HLT framework

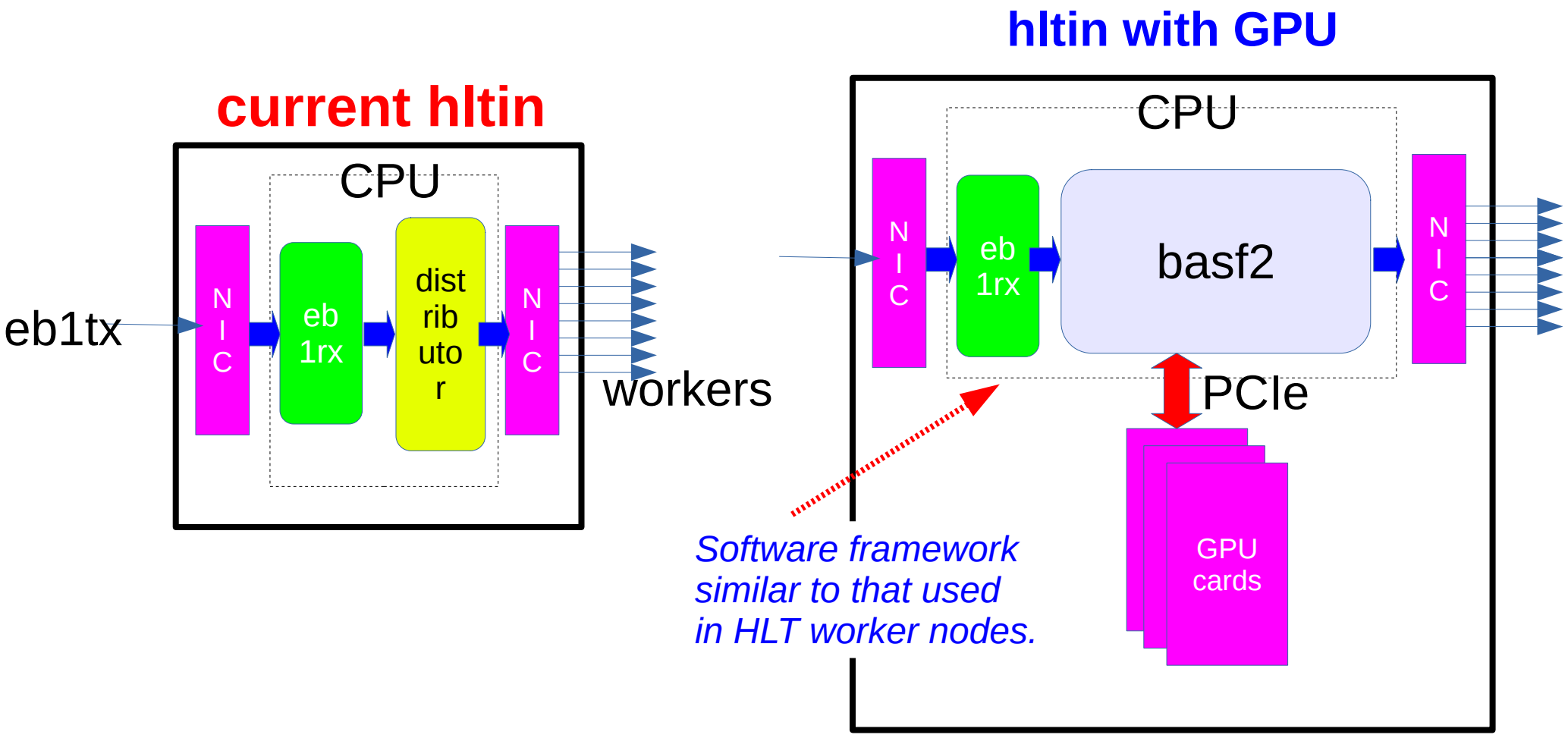
- Currently we have ~7200 Xeon cores in ~200 worker nodes in HLT.
  Baseline upgrade :
  addition of 1 worker(~70 cores) for 15 HLT units per year.

- One GPU has O(10,000) cores although the processing power per core is not so much compared to that of a multi-core CPU.



Place GPUs in the input servers of 15 HLT units and use them for Level 3 filtering.

* Having GPUs in all worker servers is too much.


current hltin
CPU
NIC
eb 1rx
distributor
NIC
eb1tx
workers
hltin with GPU
CPU
NIC
eb 1rx
basf2
NIC
PCIe
GPU cards
Software framework
similar to that used
in HLT worker nodes.

3. Test bench system

recycled from another purpose

```
itohepyc% lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          48 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   96
  On-line CPU(s) list:    0-95
Vendor ID:                AuthenticAMD
  Model name:             AMD EPYC 7643 48-Core Processor
    CPU family:           25
    Model:                1
    Thread(s) per core:   2
    Core(s) per socket:   48
    Socket(s):            1
    Stepping:             1
    Frequency boost:      enabled
    CPU(s) scaling MHz:   63%
    CPU max MHz:          3640.9170
    CPU min MHz:          1500.0000
    BogoMIPS:             4599.87
```

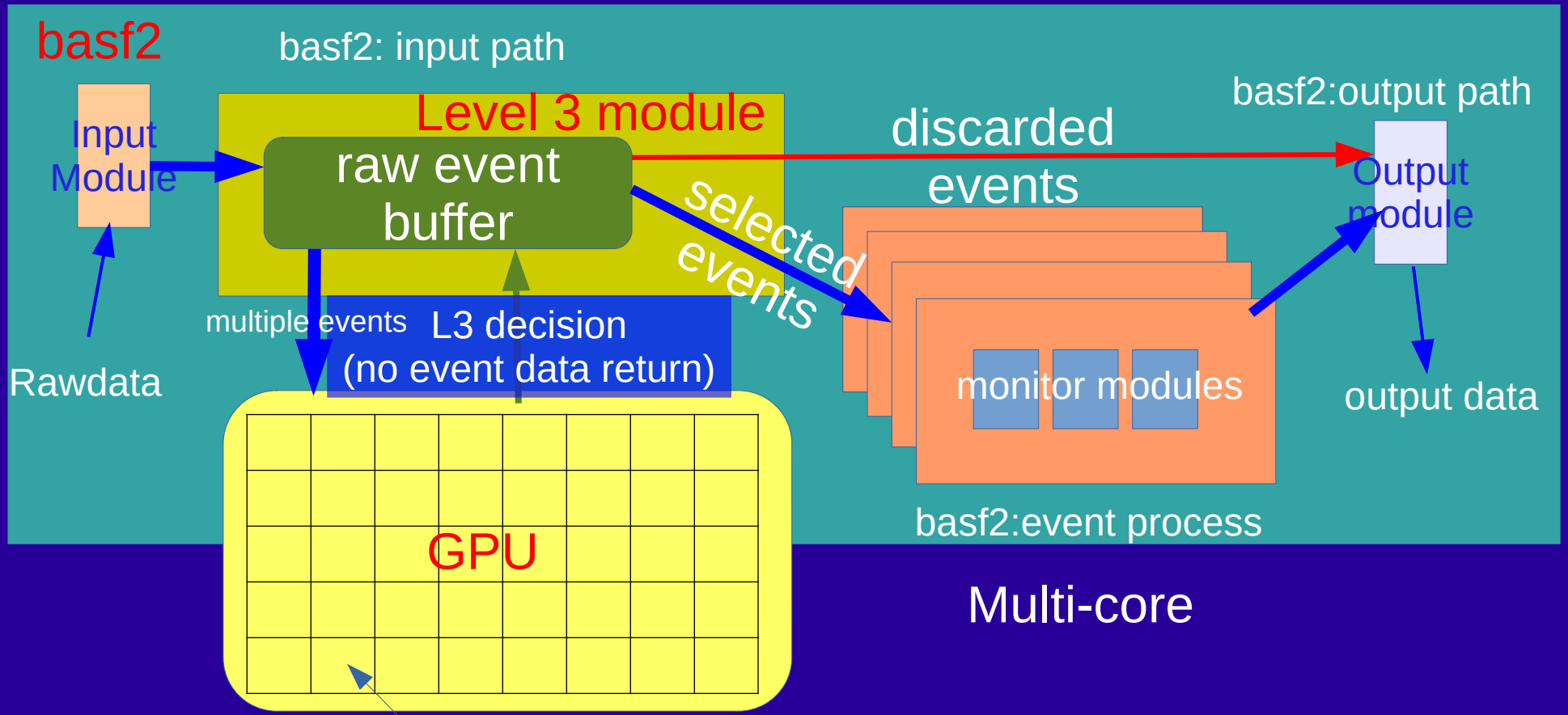
```
itohepyc% nvidia-smi
Thu Jan 30 13:59:22 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 565.57.01              Driver Version: 565.57.01      CUDA Version: 12.7     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A2                      Off |   00000000:81:00.0 Off |                    0 |
|  0%   33C    P8              5W /   60W |       1MiB /  15356MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

CPU : AMD EPYC 7643
(48 cores/96 threads)

```
itohepyc% cat /etc/redhat-release
Rocky Linux release 9.5 (Blue Onyx)
```

GPU : NVIDIA A2 Tensor
(1280 CUDA cores;
10 RT cores;
40 Tensor cores)

4. Data Flow for GPU and Level 3 selection



“thread” : SIMD parallel processing / event
L3 processing is performed on each thread

Parallelized data flow

- Batch processing nature of GPU <- SIMD processing
  * event-by-event processing is not efficient.
  * batch processing of a bunch of events is required.

- Processing of one “batch”
  1. buffer a bunch of events in the host memory.
  2. copy them to GPU memory
  3. start GPU parallel processing and wait for the completion
  4. transfer back results and compare with events in host memory
  5. send chosen events to next basf2 module

- Pipelining is required to ensure smooth, uninterrupted event data flow.

Actual Implementation

- Pipelining of the batch processing is implemented with "**CUDA Stream**".
  * CUDA has a function to run a set of GPU processing in parallel asynchronously by assigning the "batch" to different stream.

- Using the CUDA Stream, the multi-buffering is implemented in the input module of basf2.

- The Level 3 selection is implemented in the stream as a part of the "batch".

-> First test module utilizing SeqRootInput was just implemented and the debugging is started.

5. Implementation of Level 3 trigger code

- Level 3 trigger software is now being implemented in normal HLT processing by Seokhee Park(KEK). The code was originally developed by Kakuno-san (TMU) in C++ as a module of basf2.

- It consists of the fast CDC tracking, ECL clustering and TRG handling.

- The porting of the code to CUDA (nvcc) is being tried.

- Geometry/constants access from GPU to Database is one of the challenges in the porting.

  ** C++ object initialization in host and copy the initialized object to GPU over shared memory -> Seems working and being tested.*

- Conversion from “fp64” to “fp32” is also an issue to ensure the performance advantage of GPU.

Current status of GPU-L3 project

Item	Status	Remark
Seamless software environment	Done and working	Still some problem
Data transport from host to GPU (and reverse)	Done and working	Parallelized data flow is under development
Raw data unpacking (CDC/ECL/TRG)	CDC -> done ECL/TRG -> not yet	channel->wire conv. with DB access is working
Geometry database access	In progress	Generalized GPU->DB access method....
CDC Track Finding/Fitting	Next step	
ECL clustering	not yet	
TRG processing	not yet	
Level 3 decision	not yet	

Machine learning approach for Level 3 trigger

- Current design of Level 3 code is based on the conventional CDC tracking + ECL clustering approach (reference design).

- But new algorithm can be developed using the machine learning technique.

- The GPU-capable Belle II software environment can be utilized for the development.

- Possibility to utilize Tensor cores in GPU in this case.

However, the development and validation of new code will take significantly longer time and the deployment is not realistic before LS2.

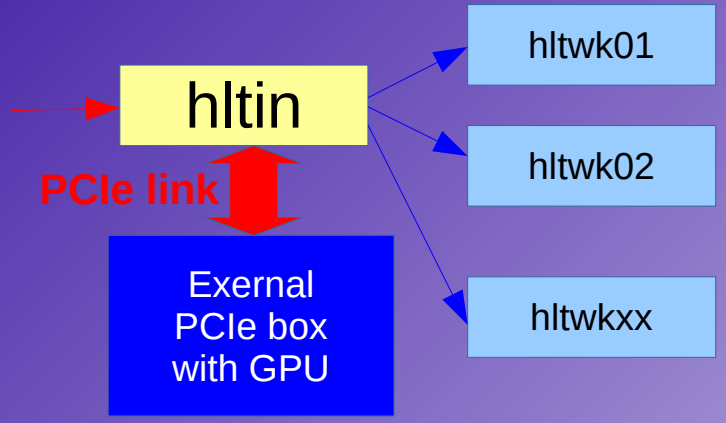
6. Test in Belle II DAQ

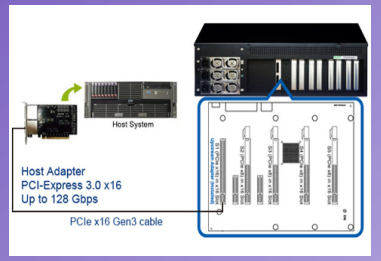
- Realistic test of GPU processing in Belle II DAQ system is being prepared.
  * Implement GPU card in “hltin” of one of HLT units.
  * Test the reference GPU-L3 processing in beam run
    (Preparing a set-up not to interrupt the actual data flow)

- Implementation is now in progress and the test will be done in coming beam run (2025c-2026ab runs).

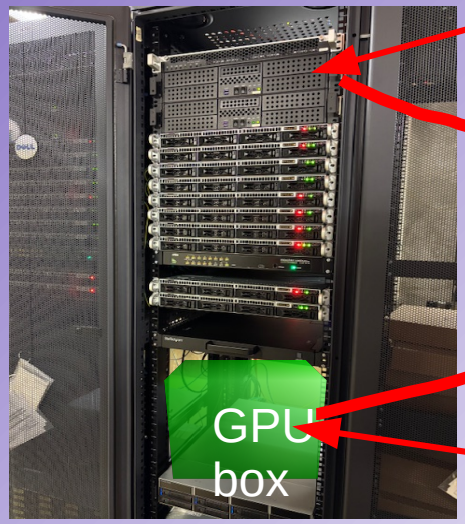
GPU Implementation in existing HLT unit (unit 15)

- Place GPU in an external rack-mount 4U unit (eGPU).
- Connect it to existing hltin via a special PCIe link.





netstor NA256-G4



hltin

PCIe-link

GPU box

**** GPU and GPU box have been delivered.***

Choice of GPU for Belle II

(for data flow up to 50kHz with 200kB/ev, before LS2)

* 1 GPU for each of 15 HLT units.
* Each GPU houses ~10,000 CUDA cores

-> in total of 150,000 cores

* GPU with better performance in fp64 but hopefully less dependent.
* Vector cores are supposed to be not used for the planned L3 processing.

- RTX5000Ada was chosen for the test bench

RTX 5090 : 21,760 cores, 32GB, ~100TFLOPS(fp32) : 50 万円

RTX 5000Ada : 12,448 cores, 32GB, 61TFLOPS(fp32) : 70 万円

RTX 6000Ada : 18,176 cores, 48GB, 91.1TFLOPS(fp32) : 150 万円

H100 : 14,592 cores, 80GB, 30TFLOPS(fp32) : 500 万円

A2 : 1,280 cores, 16GB, 4.5TFLOPS(fp32) : (used in test bench)

Versal VC1902 : 400 AI cores : 630 万円

7. Summary

- A significant shortage in the processing power of Belle II HLT is forseen because of the unexpected high background.

- The addtion of GPU in HLT framework is being considered for **the mitigation before LS2**.

- R&D on the GPU based Level 3 trigger is in progress.

- Planning to propose a new HLT upgrade strategy with GPU to replace the baseline upgrade plan after the test in beam run
  -> *Will be at the B2GM next summer*.

Acknowledge:
Development of GPU based basf2 framework is supported by Kakenhi grant (Kiban C).

Backup Slides

- Database access

- CDC [board,channel] mapping to wire number using DB.

* Currently implemented using simple memory copy of mapping table.

[host]

```
// Initialize wire map
//  GPUCDCUnpack cdcunpack;
unsigned short* local_wire_map = (unsigned short*) std::malloc ( 301*48*sizeof(unsigned short) );
GPUCDCUnpack::Instance().loadMap ( local_wire_map );
cudaMemcpy ( m_wire_map, local_wire_map, 301*48*sizeof(unsigned short), cudaMemcpyHostToDevice );
```

[host]

```
__host__
void GPUCDCUnpack::loadMap(unsigned short* wiremap)
{
  //  if ( m_map_initialized ) return;
  printf ( "[Host] loadMap called\n" );
  int minib = 1000;
  int maxib = -1;
  //  for (const auto& cm : (*m_channelMapFromDB)) {
  for (const auto& cm : (m_channelMapFromDB)) {
    const int isl = cm.getISuperLayer();
    const int il = cm.getILayer();
    const int iw = cm.getIWire();
    const int iBoard = cm.getBoardID();
    const int iCh = cm.getBoardChannel();
    const WireID  wireId(isl, il, iw);
    //    wiremap[iBoard][iCh] = wireId;
    //    wiremap[iBoard*48+iCh] = wireId;
    int ib = iBoard;
    int ic = iCh;
    if ( ib < minib ) minib = ib;
    if ( ib > maxib ) maxib = ib;
    wiremap[ib*48+ic] = wireId;
    //    if ( ib == 7 )
    //    printf ( "Loading map : board=%d, channel=%d : wireid = %d\n", ib, ic, wiremap[ib*48+ic] );
  }
  printf ( "LoadMap : minimum boardId = %d, maximum boardId = %d\n", minib, maxib );
}
```

[GPU]

```
__host__ __device__
void GPUCDCUnpack::setupMap ( unsigned short* wiremap )
{
  printf ( "[Kernel] setupMap called\n" );

  for (int ib=0; ib<301; ib++ ) {
    for ( int ic=0; ic<48; ic++ ) {
      //      m_map[ib][ic] = wiremap[ib][ic];
      m_map[ib][ic] = (unsigned short)wiremap[ib*48+ic];
      //      if ( ib==7 )
      //      printf ( "setupMap : board=%d, channel=%d : wireid = %d\n", ib, ic, m_map[ib][ic] );
    }
  }
  m_map_initialized = true;
}
```

Choice of GPU : from LHCb experience

- In the current test bench, NVIDIA A2 is being used. No. of CUDA cores is only 1024 and not enough for Level 3.

- In the 1st level HLT in LHCb, they use ~500 GPUs of NVIDIA Tesla V100, Quadro RTX 6000, and GeForce RTX 2080 Ti. Each GPU houses ~5000 CUDA cores.

- LHCb GPU-HLT handles the input data flow of 5TBytes/sec and reduces the flow down to 120GB/sec (1/60 reduction).

- Belle II case : input data flow = 200kB/ev*50kHz=10GB/sec
  scaling factor to LHCb = 10GB/5TB =0.002
  -> only 1-2 GPUs are enough for Belle II ????

LS2 is a good chance to implement new HLT design.

- Current CPU based HLT design is already old-fashioned.

- The maintenance cost of the current HLT is very high.
  * We need to replace out-of-support servers year by year.
  * CPU based servers are expensive.

- **Machine-learning based HLT software is recently spot-lighted.**
  * Background suppression by “pattern recognition” without event reconstruction.
  * Event classification by DNN.

- **For such new HLT algorithm, the use of GPU/FPGA is much suited.**

daq/gpu/modules/cudatest/src/
CudaTestModule.**cu**

CUDA kernel code

```
/**
 * CUDA Kernel Device code
 *
 * Computes the vector addition of A and B into C. The 3 vectors have the same
 * number of elements numElements.
 **/
__global__ void vectorAdd(const float *A, const float *B, float *C,
                          int numElements) {
  int i = blockDim.x * blockIdx.x + threadIdx.x;

  if (i < numElements) {
      C[i] = A[i] + B[i] + 0.0f;
  }
}
```

A basf module for GPU test written in **CUDA**.

* CUDA coding is mostly compatible with standard C++ and easily adopted in existing C++ code.
  <- "**nvcc**" works just as a frontend to g++.

In CudaTestModule::event()

```
// Copy the host input vectors A and B in host memory to the device input
// vectors in
// device memory
printf("Copy input data from the host memory to the CUDA device\n");
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);

if (err != cudaSuccess) {
  fprintf(stderr,
          "Failed to copy vector A from host to device (error code %s)!\n",
          cudaGetErrorString(err));
  exit(EXIT_FAILURE);
}

err = cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

if (err != cudaSuccess) {
  fprintf(stderr,
          "Failed to copy vector B from host to device (error code %s)!\n",
          cudaGetErrorString(err));
  exit(EXIT_FAILURE);
}

// Launch the Vector Add CUDA Kernel
int threadsPerBlock = 256;
int blocksPerGrid = (numElements + threadsPerBlock - 1) / threadsPerBlock;
printf("CUDA kernel launch with %d blocks of %d threads\n", blocksPerGrid,
       threadsPerBlock);
vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, numElements);
err = cudaGetLastError();

if (err != cudaSuccess) {
  fprintf(stderr, "Failed to launch vectorAdd kernel (error code %s)!\n",
          cudaGetErrorString(err));
  exit(EXIT_FAILURE);
}

// Copy the device result vector in device memory to the host result vector
// in host memory.
printf("Copy output data from the CUDA device to the host memory\n");
err = cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

if (err != cudaSuccess) {
  fprintf(stderr,
          "Failed to copy vector C from device to host (error code %s)!\n",
          cudaGetErrorString(err));
  exit(EXIT_FAILURE);
}
```

basf2 script

```
# Create Path
main = basf2.create_path()

# Dummy event generator
main.add_module("EventInfoSetter", evtNumList=[1], runList=[1], expList=[0])

# CudaTestModule
main.add_module ('CudaTest' )

basf2.process(main)
```

Execution result

```
itohepyc% basf2 cudatest.py
[INFO] Steering file: cudatest.py
[INFO] CudaTest: Constructor done.
[INFO] Starting event processing, random seed is set to '5dfc8bd7a911daae8f5cc6f2c79f1a73f0a0ac3c4296bc7badf200e523030ea5'
[INFO] CudaTest: started to measure elapsed time.
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
[INFO] CudaTest: terminate called
[INFO] CudaTest: Destructor.
```