Hit Finding Algorithms for the DUNE Experiment Using Single Instructions Multiple Data Parallel Processing

Adam Abed Abud (CERN)

On behalf of the DUNE collaboration

September 6, 2023
TIPP 2023 - Cape Town

Outline

Introduction

- The DUNE experiment
- Data Selection in DUNE

Trigger Primitive Generation

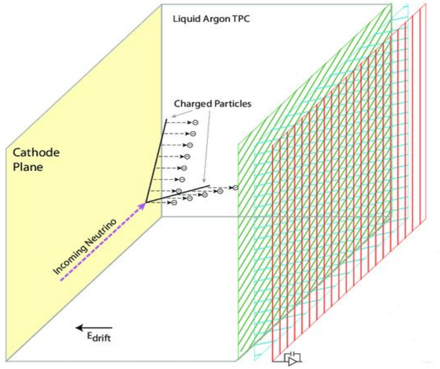
- Trigger Primitive Generation (TPG)

Performance results

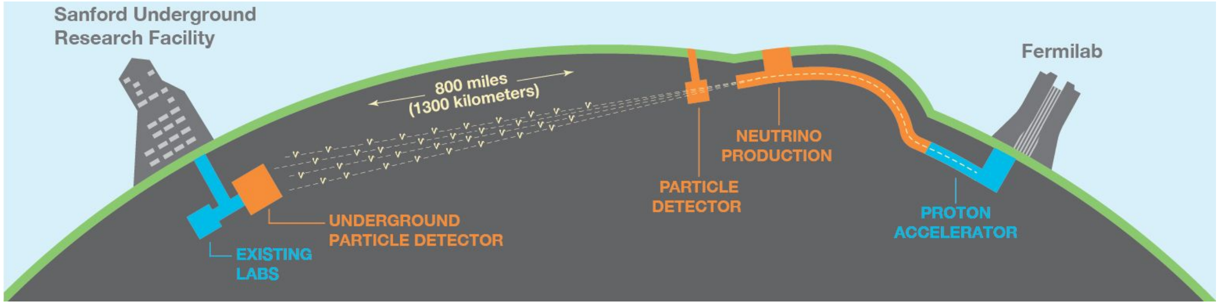
- Comparison between TPG algorithms
- Results from a prototype setup

Conclusion & Outlook

The DUNE Experiment



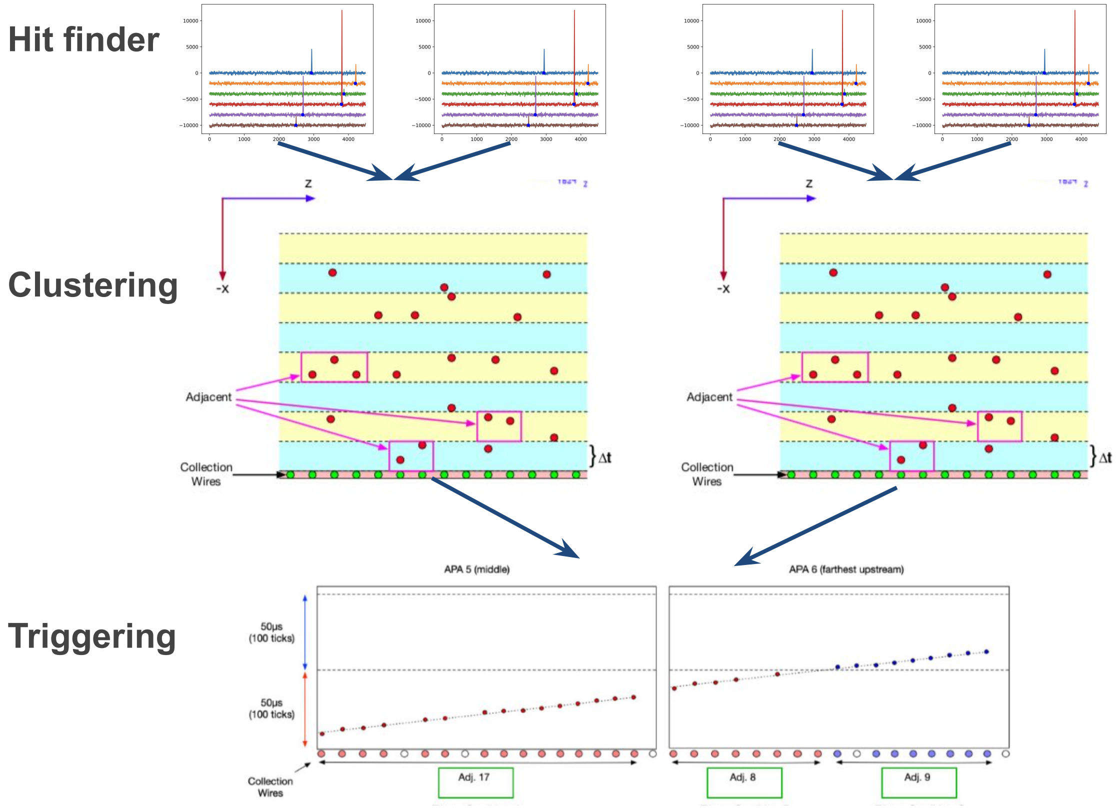
- DUNE is a next-generation long baseline neutrino experiment
- DUNE “Far Detectors” are Liquid Argon Time Projection Chamber (TPC) :
  - Digitalize signals at ~2 MHz and send ADC data (from ~0.5M ch) to the DAQ without any zero suppression
  - Modular apparatus: 150 detector segments generating ~1.5 TB/s in total
  - Use of **3 wire/strip planes** (2 for induction signal and 1 for collection signal)
- ProtoDUNE is a prototype built at CERN to test detector technologies for the DUNE Far Detectors
  - Fiducial LAr volume is 5% of a single DUNE module



Source: [DUNE Technical Design Report]

Data Selection in the DUNE Data Acquisition

General overview



- The Data Selection responsible for selecting interesting data for long term storage

Problem:

- Full complement of TPC data cannot be filtered in dedicated nodes (high data rates and volume)

Solution

- Use **self-triggering** mechanism at the detector segment level (on readout nodes)
- Hit finding algorithm localize interesting signals (Trigger Primitives, TP) on each channel

Trigger Primitive Generation (TPG)

Trigger Primitive Generation (hit finder) is the process of identifying signal on each wire above electronic noise

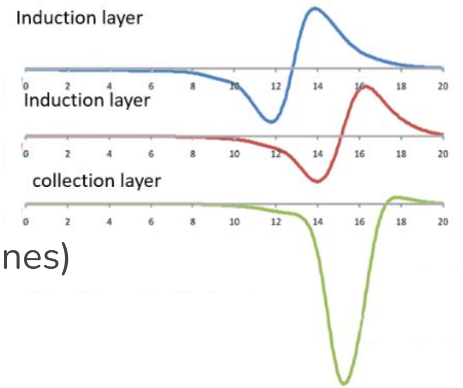
- A stream of TPs is a minimal, ready-to-use dataset that can be used for physics analysis
  - Contains: Timestamp, channel, time-over-threshold, charge collected

DUNE requirements for the TPG and scale of the problem:

- Process the total stream of continuous, fixed-rate TPC data from two detector segments in a single readout host
  - Single detector segment generates ~**10 GB/s**

Challenges for the TPG:

- Process in quasi real-time the data generated by the detector
- Difficult identifying a signal for induction channel planes
  - Signal is bipolar and S/N ratio is lower for two wire planes (induction planes)



Trigger Primitive Generation Chain

- Incoming data from the Readout System are kept in memory
- Hit finding algorithm is executed on every channel of the incoming ADC data

TPG Chain:

1. **Data expansion**: expand ADC data from 14 bit to 16 bit to ease interface with rest of computing infrastructure
2. Initialize pedestal values
3. Execute the hit finding chain (**CPU / memory intensive task)**
    - Use **SW parallelization techniques** to **speed-up** execution (e.g. **Single Instruction Multiple Data or SiMD**)
        i. Execute the TPG algorithm on multiple channels
        ii. Scalar implementation is not sufficient to sustain data rates from the detector (HW resources limited)
    - FPGAs can be used but not ideal due to increased implementation complexity, development time
4. **TP extraction:** parse output from hit finding and forward it to the next subcomponent of the Readout System

Acceleration using parallel processing techniques
Short digression on Single Instructions Multiple Data (SiMD)

- Advanced Vector Extensions (AVX) are extensions to x86 instruction set architecture
  - Available for Intel and AMD processors
- **SiMD** is used to **parallelize** and increase the throughput when executing **intensive algorithm calculations** by performing the same operation on multiple data (**vectorized code**)
- **Change** in **programming paradigm** when developing vectorized code compared to scalar code
  - Execute instructions on multiple data points simultaneously
  - Focus on optimizing memory access and overall data manipulation (bit level operations)
- Different architectures provide different register sizes and instruction sets
  - **AVX2 registers: 256 bit**

Current software implementation for TPG:

- TPG threads each use **8 AVX2 registers**
  - Processing more AVX registers (i.e. more channels) in a single thread does not fit in host available resources

Hit finding: Simple Threshold algorithm

A simple algorithm targeting unipolar planes

Decode format from Front-End → **Pedestal subtraction** → **Evaluate signal above threshold**

- Decode ADC data using specific detector format
- Expand 14-bit ADCs to 16-bits

- Pedestal subtraction is used to bring the waveform baseline to zero
- Calculate the median using of a frugal streaming method**:** use one entry at at time and update the median if N entries are higher than a fixed threshold

- A TP is identified if the incoming signal is greater than a configurable fixed threshold

Scalar vs Vectorized code

```cpp
void
frugal_accum_update(int16_t& m, const int16_t s,
                    int16_t& acc, const int16_t acclimit)
{
  if (s > m)
    ++acc;
  if (s < m)
    --acc;

  if (acc > acclimit) {
    ++m;
    acc = 0;
  }

  if (acc < -1 * acclimit) {
    --m;
    acc = 0;
  }
}

```

```cpp
inline void
frugal_accum_update_avx2(__m256i& __restrict__ median,
                         const __m256i s,
                         __m256i& __restrict__ accum,
                         const int16_t acclimit,
                         const __m256i mask)
{
  // if the sample is greater than the median, add one to the accumulator
  // if the sample is less than the median, subtract one from the accumulator.

  __m256i is_gt = _mm256_cmpgt_epi16(s, median);
  __m256i is_eq = _mm256_cmpeq_epi16(s, median);

  __m256i to_add = _mm256_set1_epi16(-1);

  to_add = _mm256_blendv_epi8(to_add, _mm256_set1_epi16(1), is_gt);
  to_add = _mm256_blendv_epi8(to_add, _mm256_set1_epi16(0), is_eq);

  // Don't add anything to the channels which are masked out
  to_add = _mm256_and_si256(to_add, mask);

  accum = _mm256_add_epi16(accum, to_add);

  is_gt = _mm256_cmpgt_epi16(accum, _mm256_set1_epi16(acclimit));
  __m256i is_lt =
    _mm256_cmpgt_epi16(_mm256_sign_epi16(accum, _mm256_set1_epi16(-1 * acclimit)),
                       _mm256_set1_epi16(acclimit));

  to_add = _mm256_setzero_si256();
  to_add = _mm256_blendv_epi8(to_add, _mm256_set1_epi16(1), is_gt);
  to_add = _mm256_blendv_epi8(to_add, _mm256_set1_epi16(-1), is_lt);

  // Don't add anything to the channels which are masked out
  to_add = _mm256_and_si256(to_add, mask);

  median = _mm256_adds_epi16(median, to_add);

  // Reset the unmasked channels that were >10 or <-10 to zero
  __m256i need_reset = _mm256_or_si256(is_lt, is_gt);
  need_reset = _mm256_and_si256(need_reset, mask);
  accum = _mm256_blendv_epi8(accum, _mm256_setzero_si256(), need_reset);
}
```

Investigation of other TPG algorithms

Absolute Running Sum algorithm

- **Motivation for investigating TPG algorithms:**
  - Identify an algorithm that runs efficiently on all three wire planes
    - Additional topological information may improve trigger at lower energy thresholds
    - Fiducialize away low-energy background events
  - Provide good signal to noise ratio

- **Problems** of Simple Threshold algorithm:
  - Works well only on collection planes
  - Not resilient against coherent noise

- **Solution:**
  - Development of the **Absolute Running Sum (AbsRS) TPG algorithm:** $y_n = R \cdot y_{n-1} + \frac{|x_n|}{s}$
    - Use **absolute value** to compensate for negative pulses in induction channels
  - Simulations show that AbsRS works on all planes of the LAr TPC detector with satisfactory S/N ratio
  - AbsRS also acts as low pass filter [1] [2] [3]

Hit finding chain: AbsRS algorithm

Decode format from Front-End	Pedestal subtraction	Apply AbsRS	RS pedestal subtraction	Evaluate TPs
• Decode ADC data using specific detector format • Expand 14-bit ADCs to 16-bits	• Pedestal subtraction is used to bring the waveform baseline to zero • Calculate the median using of a frugal streaming method	• Apply the AbSRS on the incoming ADC entry	• Apply a pedestal subtraction on the RS value	• A TP is identified if incoming signal is greater than threshold

Setup of the system for performance testing

- **Setup of the system for testing:**
  - Data Producers: emulate data produced from a single detector unit
  - Consumers: parse data from the Data Producers and insert into memory
  - TPG threads: threads executing the hit finding algorithm
- **Testing objectives:**
  - **Characterization of the TPG algorithm**: synthetic benchmark to emulate TPs
    - Produce TPs at the average expected DUNE rate of 100 Hz per channel
- The **TPG** is a **CPU**, **cache** and **memory** bandwidth **intensive algorithm**
- **Tuning is needed** to better use all of the host machine's resources
  - Use of dedicated cores for TPG threads
    - HW and data locality to the processing threads are fully controlled by SW configuration
    - Reduces the need for context switching
  - Select hosts with higher clock rate:
    - Reduce number of instructions per cycle and overall CPU utilization

Comparison between TPG algorithms
Simple Threshold vs AbsRS

- **Single CPU core utilization** between Simple Threshold and AbsRS threads: **40% vs 65%**
  - Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz
  - CPU breakdown for each component of the TPG algorithm

Simple Threshold

Single core utilization:
~40% CPU Utilization

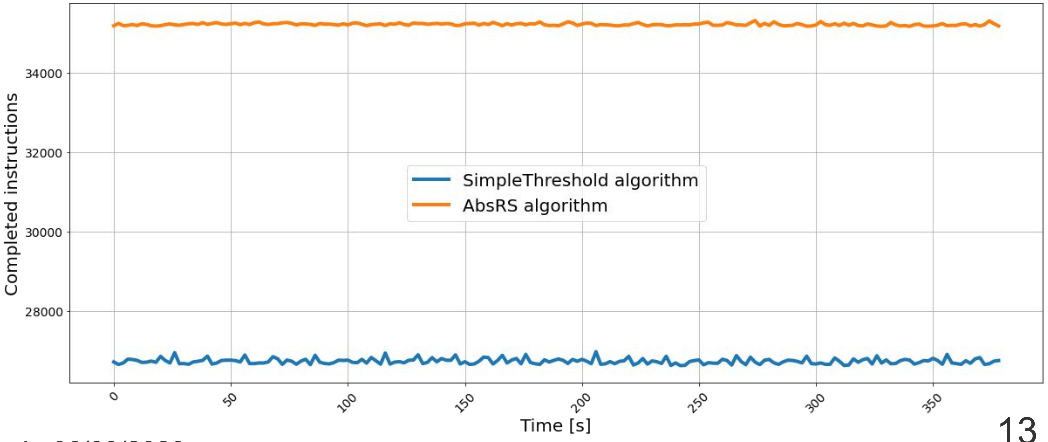
Frame Expansion: 13 %
TPG Processing: 23 %
Other: 4 %

AbsRS

Single core utilization:
~65% CPU Utilization

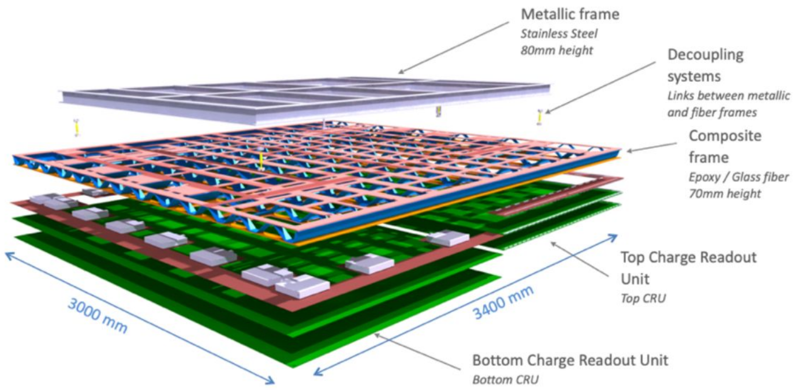
Frame Expansion: 13 %
TPG Processing: 47 %
Other: 5 %

- Increased algorithmic complexity for AbsRS
  - Number of instructions increase by 25%
- In general, average number of instructions is not impacted by presence of signals above threshold
  - TPG is always executed on all the channels
  - Allows transitioning from a synthetic benchmarking to testing on real physics signals



Results from a prototype setup (1/2)

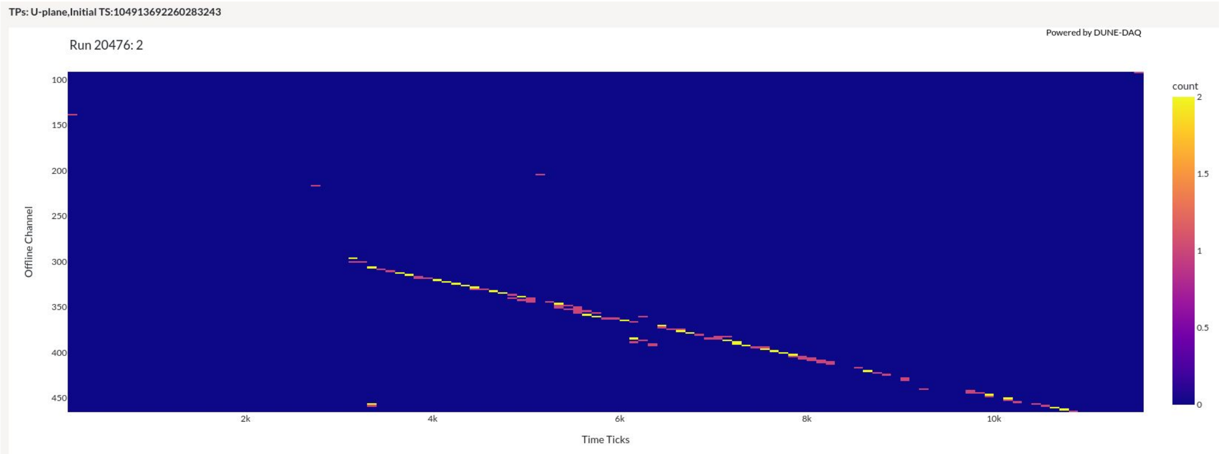
- Prototype setup (Coldbox) used to test detector components available at CERN
- TPC with a drift volume of ~30 cm:
  - Based on a vertical drift TPC detector
  - Hit finder can be exercised on real signals (e.g. cosmic muons)
- **Testing objectives**
  - Performance evaluation of TPG algorithms (Simple Threshold, AbsRS)
  - Use TPs to exercise different trigger algorithms (e.g. long cosmic tracks)



Sources: [Prototyping the Dune Vertical Drift TPC](), Oliver Lantwin

Results from a prototype setup (2/2)

- View of a single induction plane for TPs obtained in a run in March 2023
  - Offline channel vs time expressed in 62.5 MHz ticks
    - AbsRS algorithm & track trigger: long cosmic tracks are identified in the detector
  - **Managed to fully sustain the detector data rates within the resources of a single host**
- Testing using a prototype setup allows to **move from a synthetic benchmark to testing on a real system**
  - Fluctuations in TP rate can be investigated, controlled and monitored when using the Coldbox
  - Learn how to operate and configure TPG algorithms (e.g. setting up the thresholds, etc.)
  - Understand the performance difference between TPG algorithms and bottlenecks of host machine using real signals



Conclusion & Outlook

- **Achieved Trigger Primitive Generation within the resources of single readout host**
- **Performance optimization** and benchmarking of **Trigger Primitive Generation algorithms**
- Important milestones achieved in the last 6 months using a prototype setup at CERN:
  - Produced Trigger Primitives on induction planes
  - Tested a new TPG algorithm: Absolute Running Sum
  - Used induction planes for trigger algorithms

Outlook

- Evaluate the impact on the TPG algorithm with the new detector data format that has been recently introduced (WIB-Ethernet)
- Optimize the TPG algorithms with the new detector format on different server topologies
- Testing TPG algorithms on ProtoDUNE detector expected for 2024
- Further TPG algorithm studies:
  - Study further TPG algorithms
  - Optimize the execution of the TPG implementation
  - Add more relevant quantities to the Trigger Primitive objects

Thank you

adam.abed.abud@cern.ch

September 6, 2023
TIPP 2023 - Cape Town

Pedestal Finding

Algorithm design

1. Set a value for the maximum number of samples (X) to process before updating the median
2. Start with an accumulator = 0 and the estimate of the median (= ADC value of the first WIB)
3. If ADC value > median
   - Accumulator += 1
4. If ADC value < median
   - Accumulator -= 1
5. If Accumulator = X
   - Median += 1 ; reset accumulator to 0
6. If Accumulator = -X
   - Median -= 1 ; reset accumulator to 0