PARALLELIZING THE UNPACKING AND CLUSTERING OF DETECTOR DATA FOR RECONSTRUCTION OF CHARGED PARTICLE TRACKS ON MULTI-CORE CPUs AND MANY-CORE GPUS

Giuseppe Cerati², Peter Elmer³, Brian Gravelle⁵, Matti Kortelainen², Vyacheslav Krutelyov⁴, Steven Lantz¹, Mario Masciovecchio⁴, Kevin McDermott¹, Boyana Norris⁵, Allison Reinsvold Hall², Micheal Reid¹, Daniel Riley¹, Matevž Tadel⁴, Peter Wittich¹, Bei Wang³, Frank Würthwein⁴, and Avraham Yagil⁴

¹Cornell University, ²Fermi National Accelerator Laboratory, ³Princeton University, ⁴UC San Diego, ⁵University of Oregon

Connecting The Dots Workshop, April 22, 2020
Overview

• Motivation
  • Towards mkFit at High Level Trigger (HLT)
• Unpacking of raw data format from CMS Silicon Strip Tracker (CMS SST)
  • Structure-of-array (SoA) format
• Parallel “Three Threshold” Clustering Algorithm
• Enabling Multi-event Processing
  • OpenMP nested parallelism in multicore CPU
  • CUDA streams in GPU
• Performance Results
  • Single event
  • Throughput
• Summary and plans
Motivation

• Exponential growth in reconstruction time at increasing pile-up (PU). Tracking is by far the most time-consuming step of CMS reconstruction
  • Thus, tracking codes are a clear target for moving to GPU
• Current target: deploy as much as possible into the HLT for Run3 (2021-2024)
  • Major CMS efforts include the Patatrack project (pixel reconstruction and tracking) and mkFit (Kalman filter tracking)
Towards mkFit at HLT

• Tracking on GPUs enables new trigger capabilities and thereby physics reach of CMS
  • Global tracking instead of regions-of-interest driven
  • Tracking on a much larger fraction of Level-1 triggered events

• To avoid I/O and memory bottlenecks, the full tracking chain starting from the RAW data needs to be included

• This presentation considers the unpacking, clustering and transformation of the silicon strip tracker data as needed for tracking
  • Memory bandwidth limited, so not well suited to GPU
  • Currently CMS uses a serial algorithm. We investigated a parallelized and vectorized implementation for CPU and GPU
Event-by-event raw SST data are organized by Front End Driver (FED), in total 440.

Each FED has 96 optical channels and 256 strips per channel.
- Each event has ~10M strips, 5% occupancy (for PU50)

ADC values in a channel are usually compressed via zero suppression.
- Think of each FED’s ADCs as a sparse matrix: channel = row, strips = column
- Zero suppression scheme is a variant of CSR (compressed sparse row)

Pre-measured calibration data are available for each detector strip. This includes:
- Good or bad; gain value (i.e., the ratio of ADC value to charge); noise (in ADC units)
Unpacking Raw to SoA Data Format

- SoA data format is more efficient on CPU SIMD register and with coalesced memory access on GPU
- Unpacking transforms all event-by-event raw data and calibration data in SoA format

### Channel 0

| Channel length | Additional info in channel header | Start strip # | Count | ADC | ADC | ADC | Start strip # | Count | ADC | ADC | ADC | ...
|----------------|----------------------------------|---------------|-------|-----|-----|-----|---------------|-------|-----|-----|-----|-------|

### Channel M

| Channel length | Additional info in channel header | Start strip # | Count | ADC | ADC | ADC | Start strip # | Count | ADC | ADC | ADC | ...
|----------------|----------------------------------|---------------|-------|-----|-----|-----|---------------|-------|-----|-----|-----|-------|

M (~40k) is the number of channels in the tracker

ADC: 0 0 0 0 \cdots N-1

strip #: 0 0 0 0 \cdots N-1

N (~500k) is the number of active strips in the tracker
Parallel “Three Threshold” Clustering

- **Find seed strips** *(operations parallel over all ADCs read out)*:
  - Make a mask for seed candidates that exceed the “seed threshold”
  - Remove neighboring seed candidates
  - Make an array of the indices of seed candidates
- **Form clusters** *(operations parallel over clusters)*:
  - Determine the left and right boundaries by scanning for consecutive strips above the channel threshold
  - Compute the cluster charge, centroid and check against “cluster threshold”
Enabling Multi-event Processing

  - Use CUDA streams to launch multiple events concurrently on the same device
    - Maximize GPU utilization
    - Reduce data transfer overhead by overlapping communication and computation
  - Use some of the PATATRACK GPU infrastructure ([https://github.com/cms-patatrace/cmssw](https://github.com/cms-patatrace/cmssw)), especially caching memory allocator

- **CPU: Nested Parallelism in OpenMP**
  - Two levels of parallel-for loops: an outer level to handle different events, and an inner level to handle sets of strips within a single event
  - Vectorization via the OpenMP simd clause in the inner loop
  - Ensure the right numbers of threads with proper thread affinities are given at each level
    
    ```
    export OMP_NESTED=TRUE
    export OMP_NUM_THREADS=n,m
    export OMP_PLACES=cores
    export OMP_PROC_BIND=spread, close
    export KMP_HOT_TEAMS_MODE=1 (Intel only)
    export KMP_HOT_TEAMS_MAX_LEVEL=2 (Intel only)
    ```
  - Use first touch policy for NUMA aware memory placement
## Throughput

<table>
<thead>
<tr>
<th>Events Currency n (parallelization within one event m)</th>
<th>Iterations</th>
<th>Total Events</th>
<th>CPU Time (seconds)</th>
<th>CPU Throughput (events/s)</th>
<th>GPU Time (seconds)</th>
<th>GPU Throughput (events/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (28)</td>
<td>840</td>
<td>840</td>
<td>1.70</td>
<td>492</td>
<td>1.36</td>
<td>615</td>
</tr>
<tr>
<td>2 (14)</td>
<td>420</td>
<td>840</td>
<td>2.10</td>
<td>400</td>
<td>1.29</td>
<td>649</td>
</tr>
<tr>
<td>4 (7)</td>
<td>210</td>
<td>840</td>
<td>1.67</td>
<td>502</td>
<td>1.41</td>
<td>595</td>
</tr>
<tr>
<td>7 (4)</td>
<td>120</td>
<td>840</td>
<td>1.57</td>
<td>532</td>
<td>1.46</td>
<td>574</td>
</tr>
<tr>
<td>14 (2)</td>
<td>60</td>
<td>840</td>
<td>1.50</td>
<td>560</td>
<td>1.53</td>
<td>548</td>
</tr>
<tr>
<td>28 (1)</td>
<td>30</td>
<td>840</td>
<td>2.10</td>
<td>400</td>
<td>1.54</td>
<td>546</td>
</tr>
</tbody>
</table>

- System: Tigergpu at Princeton Research Computing
  - CPU: 2.4 GHz Xeon Broadwell E5-2680 v4, 2x14 cores
  - GPU: 1329 MHz P100, 56 multiprocessors

- We use the same TTBar PU70 event data to mimic the performance behavior of running 840 total events
- Throughput: Total events #/time
- Each CPU test uses all nxm=28 cores with nested parallelism. For example 2(14) means we have 2 concurrent events, where each event is processed by 14 threads.
- CPU throughput: 400-560 (events/s), GPU throughput: 546-649 (events/s)
Summary and Plans

• Summary
  • Enables parallelization and vectorization of the CMS strip tracker clustering algorithm
  • Demonstrates very encouraging performance results on both multicore CPU and GPU

• Plans
  • Integration with CMSSW
    • GPU kernels should be reusable with minimal changes
      • Use the PATATRACK infrastructure where possible
    • Validation: expect to produce the identical results to the original CMSSW algorithm
  • mkFit hit producer
    • Convert clusters to global hit coordinates
  • Also investigating feasibility of OpenCL implementation for FPGA
BACKUP