Parallel Track Reconstruction in CMS
Using the Cellular Automaton Approach

Daniel Funke, Thomas Hauth, Vincenzo Innocente, Günter Quast, Peter Sanders and Dennis Schieferdecker | October 15, 2013
Motivation

Increase in LHC’s luminosity and energy will change events from this

20-30 simultaneous pp collisions $\Rightarrow \approx 100$ tracks per event
Motivation

Increase in LHC’s luminosity and energy will change events from this to this

80-100 simultaneous pp collisions $\Rightarrow$ more than 1000 tracks per event
Motivation

Challenges:

- Increased combinatoric complexity
- Stagnating CPU clock speed
  ⇒ New technologies: multi-core, vector units, GPGPUs
- Heterogeneous CMS computing environment ⇒ transparent solution

Approach:

- Parallelism on intra- and inter-event level
- Simple geometric calculations and data structures
- OpenCL: open framework for CPU and GPU computing
  ⇒ one code, all platforms – ideal for CMS environment
- Cellular automaton: reconstruct tracks by joining compatible hit triplets
  ⇒ efficient and effective criteria for valid triplet combinations
  ⇒ fast triplet finding algorithm
Motivation

Challenges:

- Increased combinatoric complexity
- Stagnating CPU clock speed
  ⇒ New technologies: multi-core, vector units, GPGPUs
- Heterogeneous CMS computing environment ⇒ transparent solution

Approach:

- Parallelism on intra- and inter-event level
- Simple geometric calculations and data structures
- OpenCL: open framework for CPU and GPU computing
  ⇒ one code, all platforms – ideal for CMS environment
- Cellular automaton: reconstruct tracks by joining compatible hit triplets
  ⇒ efficient and effective criteria for valid triplet combinations
  ⇒ fast triplet finding algorithm
Problem Space

Reduce three dimensional problem to two dimensions

\[ x = r \cdot \sin \phi \quad \text{and} \quad y = r \cdot \cos \phi \]

Barrel

- Detector layer prescribes \( r_{\text{layer}} \).
- \((\phi, z)\) describe hit.

Endcap

- Detector layer prescribes \( z_{\text{layer}} \).
- \((\phi, r)\) describe hit.
Problem Space

Reduce three dimensional problem to two dimensions

\[ x = r \cdot \sin \phi \quad \text{and} \quad y = r \cdot \cos \phi \]

Barrel

Following work considers barrel layers

- Detector layer prescribes \( \eta_{\text{layer}} \).
- \((\phi, z)\) describe hit.
Algorithm Overview

- **Grid data structure**: queries for hits within predicted search range
- Simple and local computations for predicting search range for hit pairs and triplets
- Address peculiarities of OpenCL
  - No dynamic memory allocation
  - Penalty for diverging threads
- Fine-grained workload distribution
- Physical studies for triplet joining
  ⇒ not yet implemented in OpenCL
Two-Pass Scheme

Problem:
- OpenCL: no dynamic memory allocation within kernel
- Potentially huge number of outputs

Approach: Two-pass scheme

1. Count number of valid items
2. Store valid items appropriately

- If validity is expensive to determine
  ⇒ "oracle"-bitstring: reuse validity check result in store function
- All presented algorithms follow this two-pass scheme
Two-Pass Scheme

Problem:
- OpenCL: no dynamic memory allocation within kernel
- Potentially huge number of outputs

Approach: Two-pass scheme

1. Count number of valid items
2. Store valid items appropriately

- If validity is expensive to determine
  ⇒ „oracle“-bitstring: reuse validity check result in store function
- All presented algorithms follow this two-pass scheme
Grid Data Structure

- CMSSW: hits stored in $k$-d tree
- **Uniform grid**: more suitable for GPU construction and retrieval

- Ex situ construction with two pass algorithm
- One detector layer per work-group
- Simultaneous grid building for all layers
- Concurrent processing of multiple events
- Local memory use if grid granularity permits
Pair Building

For hit in second layer: find compatible hit in first layer

- Predict $z$-range based on maximum distance of track to origin
- Calculate $\phi$-range based on minimum transverse momentum $p_T$

$z$-prediction

$\phi$-prediction

$r_{\text{min}} \propto \min p_T \Rightarrow \Delta \phi \geq |\alpha - \beta|$
Triplet Prediction

For hit pair: find compatible hits in third layer

- $z$-range prediction based on straight line extrapolation
  + parameter to account for bending and multiple scattering
- Prediction of $\phi$-range similar to pair building
  $\Rightarrow$ move origin of coordinate system to hit in first layer

$z$-prediction

$\phi$-prediction
Triplet Filtering

Discard **fake** triplets: not belonging to a particle’s trajectory

- Computationally inexpensive criteria to identify valid triplets
- Cutoff values derived from simulated events for each layer configuration

Transverse bending

\[ |\phi' - \phi| \leq d\phi \]

Longitudinal bending

\[ \left| \frac{\theta'}{\theta} - 1 \right| \leq d\theta \]

Transverse impact parameter

Riemann fit method
Triplet Joining

Two hit triplets can be joined if
- both have two hits in common
- their difference in momentum is bounded by
\[ \left| \frac{q}{p} - \frac{q'}{p'} \right| \leq dp \]
- the difference between the normal vectors to their trajectory is bounded by
\[ |n - n'| \leq dx \]
Evaluation

Physics performance measures:
( obtained by matching algorithm output to simulated truth)
- Efficiency = \( \frac{n_{\text{valid}}}{n_{\text{simulated}}} \)
- Fake Rate = \( \frac{n_{\text{fakes}}}{n_{\text{found triplets}}} \)
- Clone Rate = \( \frac{n_{\text{clones}}}{n_{\text{found triplets}}} \)
- Background = \( n_{\text{fakes}} \)

Runtime performance measures:
- Kernel time: similar to CPU time
- Wall time: includes overhead due to OpenCL, data transfers, ...
- Speedup measured as ratio := \( \frac{\text{baseline algorithm}}{\text{new algorithm}} \)
Physics Performance – Setup

Realistic events:
- QCD "bread-and-butter" events and $t\bar{t}$ events with complex topology
- 2000 events, $\sqrt{s} = 14$ TeV, $p_T \geq 1$ GeV $c^{-1}$, barrel only
- Average of 120 tracks per event

Artificial events:
- Algorithmic performance evaluated with $[1 \ldots 4096]$ muon tracks
- Origin at (0,0), $p_T \in [1, 10]$ GeV $c^{-1}$, $\eta \in [-1, 1]$
- Triplet finding in pixel barrel layers evaluated
Algorithmic Performance - Setup

CPU:
- Core i7-3930K (6 cores, 3.20GHz)
- 500 EUR, 154 GFLOPS, 1.2 GFLOPS W$^{-1}$
- SLC 6.4, Intel OpenCL SDK 2012, OpenCL 1.1, GCC 4.7.2

GPU:
- GeForce GTX 660
- 250 EUR, 1881.6 GFLOPS, 13.4 GFLOPS W$^{-1}$
- Ubuntu 12.04, NVIDIA driver 319.23, OpenCL 1.1, GCC 4.7.2

CMSSW:
- CMSSW 6.0.0, SLC 6.4, GCC 4.6.2
- Single threaded application ⇒ only one CPU core used
- Initial seeding step in pixel barrel evaluated ⇒ sophisticated calculations: multiple scattering, bending corrections
Physics Performance – Triplet Finding

- ≈ 80% efficiency throughout detector \( \sim \) order of CMSSW initial seeding
  \( \Rightarrow \) good result considering simplicity of approach

- High fake rate for layer 4+ \( \Rightarrow \) less precise silicon strip dets.
  \( \Rightarrow \) looser cuts
  \( \Rightarrow \) multiple triplet finding passes with increasingly looser cuts
Combination of triplets from seeding in layers 1-2-3 and 2-3-4:

- **Same hit cut** eliminates most fake combinations ⇒ computationally inexpensive
- **≈ 95% efficiency** for this step, **60% fake rate** ⇒ reduce fake triplets
Algorithmic Performance – #Tracks

For > 500 \text{tracks/event}:
- OCL on GPU outperforms OCL on CPU up to factor 64
- OCL on CPU (6 cores) \approx same performance as CMSSW (1 core)
Algorithmic Performance – Grid

Finer-grained grids:

+ Reduced combinatorics in pair building and triplet prediction
  - Data structure too large for fast local memory of GPU
  ⇒ Performance penalty in grid building and pair generation

| time [ms] | Processing Time over Tracks |
|-----------|-----------------------------|
| coarse grid - CPU | medium grid - CPU | fine grid - CPU |
| coarse grid - GPU | medium grid - GPU | fine grid - GPU |

| tracks / event | ratio |
|----------------|-------|
| 0.0 | 0.5 | 1.0 | 1.5 | 2.0 | 2.5 |

Daniel Funke – Parallel Track Reconstruction

October 15, 2013 18/19
Conclusions

Triplet Finding
- Parallel triplet finding algorithm implemented with OpenCL
- Validation of physical performance with $\approx 80\%$ efficiency
- Favorable runtime benchmarks for events with $> 500$ tracks
  $\Rightarrow$ Speedup of up to 64 on GPU compared to CPU
- Processing of multiple events required to fully exploit GPUs

Triplet Joining
- Suitable efficiently computable criteria identified
- Overall efficiency of 75% and reasonable fake rejection

Future Work
- Implement triplet joining in OpenCL
- Extend geometric calculations to endcaps
- Evaluate CMSSW framework integration
Conclusions

Triplet Finding

- Parallel triplet finding algorithm implemented with OpenCL
- Validation of physical performance with $\approx 80\%$ efficiency
- Favorable runtime benchmarks for events with $>500$ tracks
  $\Rightarrow$ Speedup of up to 64 on GPU compared to CPU
- Processing of multiple events required to fully exploit GPUs

Triplet Joining

- Suitable efficiently computable criteria identified
- Overall efficiency of 75\% and reasonable fake rejection

Future Work

- Implement triplet joining in OpenCL
- Extend geometric calculations to endcaps
- Evaluate CMSSW framework integration
Backup
Influence of Work-Group Size

- GPU very sensitive to work-group size – CPU not (bad auto-vectorization)
- GPU outperforms CPU up to factor $\approx 64$
Runtime over Events

Concurrent processing of events amortizes OpenCL overhead
⇒ essential for GPU usage

100 tracks per event

1000 tracks per event

Open question: How to realize multiple concurrent events in framework?
⇒ Heuristic based on expected tracks/event
IO requires large portion of runtime on GPU up to ≈ 256 tracks per event, then triplet prediction takes over.

Grid building time amortizes for larger events (≈ 256 tracks)
IO transfer negligible on CPU

Grid data structure building dominates runtime for events $< \approx 128$ tracks
High efficiency of $\approx 98\%$

For $> 100$ tracks from origin: very high occupancy in detector
$\Rightarrow$ high fake rate expected