Experiments with multi-threaded velopixel track reconstruction

PASC Conference 2016

8.6.2016

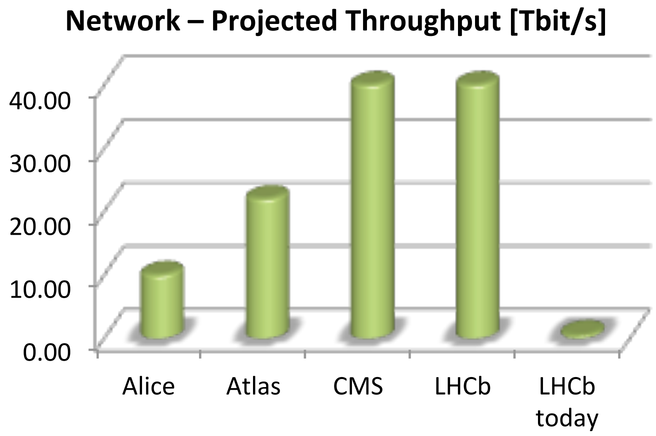
Omar Awile (omar.awile@cern.ch),

Pawel Szostek

CERNopenlab

A Thread-Parallel Implementation of High-Energy Physics Particle Tracking on Many-Core Hardware Platforms

Online Computing Challenges at the LHC



Basic constraints:

- limited money
- limited power
- limited manpower

Challenges:

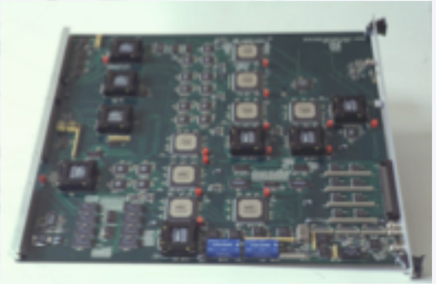
- Hard time constraints @ high throughput - 40MHz readout rate at trigger (for LHCb).
- The task of the Trigger and Data Acquisition (DAQ) is to reduce data volume for LHC experiments 100 TB/s → 100 PB/yr (factor ~25k).
- While filtering for and reconstructing interesting events.

Challenges for trigger and DAQ upgrade

For the example of LHCb

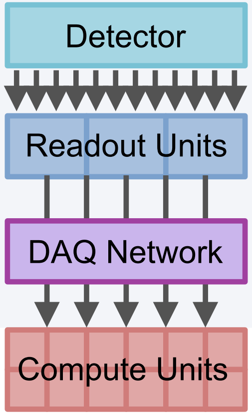
L1 Trigger

- High efficiency despite overlapping collisions add tracking information
- Flexible, robust and easy to reproduce
- Algorithms must process ~10’000 events/s



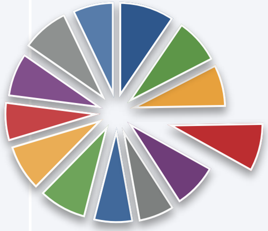
DAQ

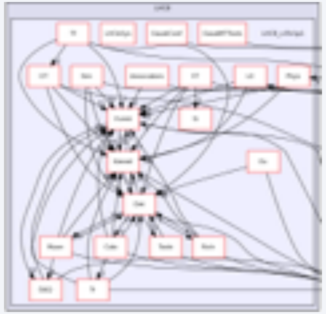
- Collision data spread over 10’000 pieces
- Data gathered onto one of 1000s compute units
- Compute units run complex filter algorithms



High-Level Trigger

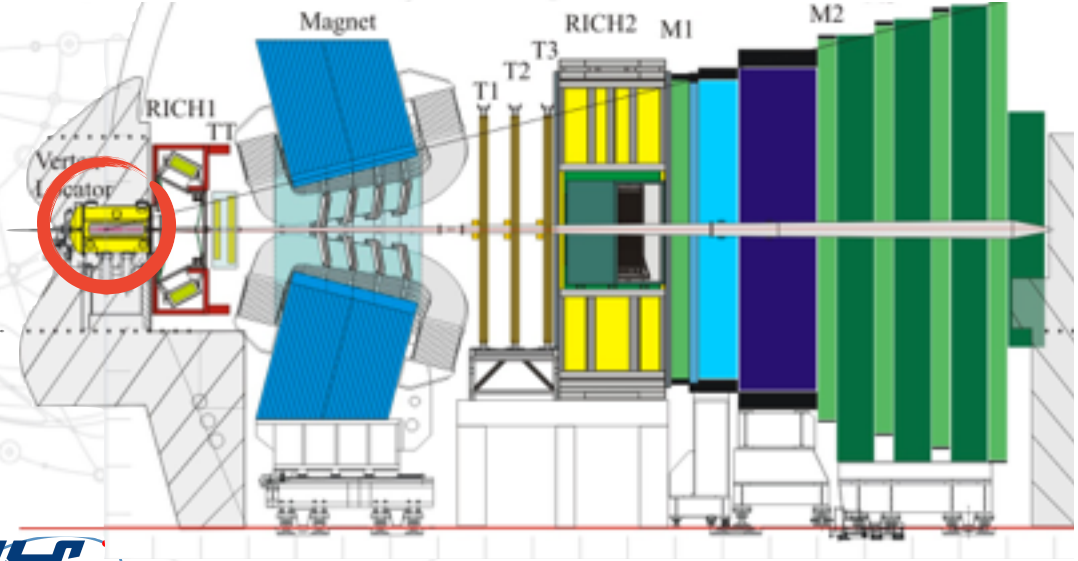
- large software infrastructure
- flat time profile
- complex and costly algorithms for reconstruction
- difficult to parallelize algorithms





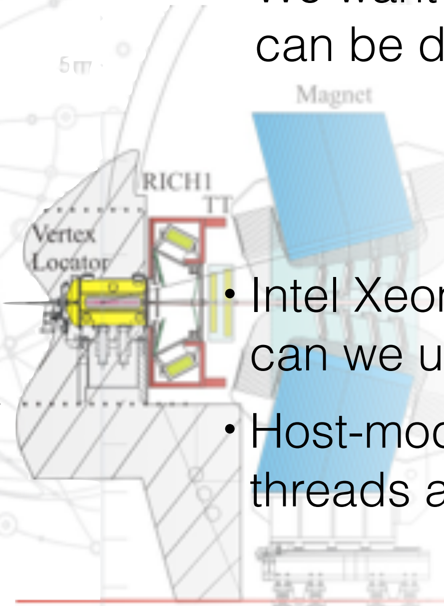
Thread-parallel track reconstruction

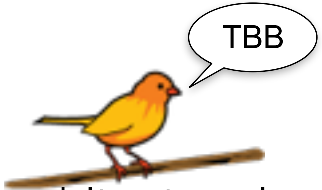
- Triggering is parallelized by running multiple (serial) instances of code
- We want to explore how track reconstruction for vertex locator data can be done on multi- and manycore CPUs - using multithreading.



Thread-parallel track reconstruction

- Triggering is parallelized by running multiple (serial) instances of code
- We want to explore how track reconstruction for vertex locator data can be done on multi- and manycore CPUs - using multithreading.

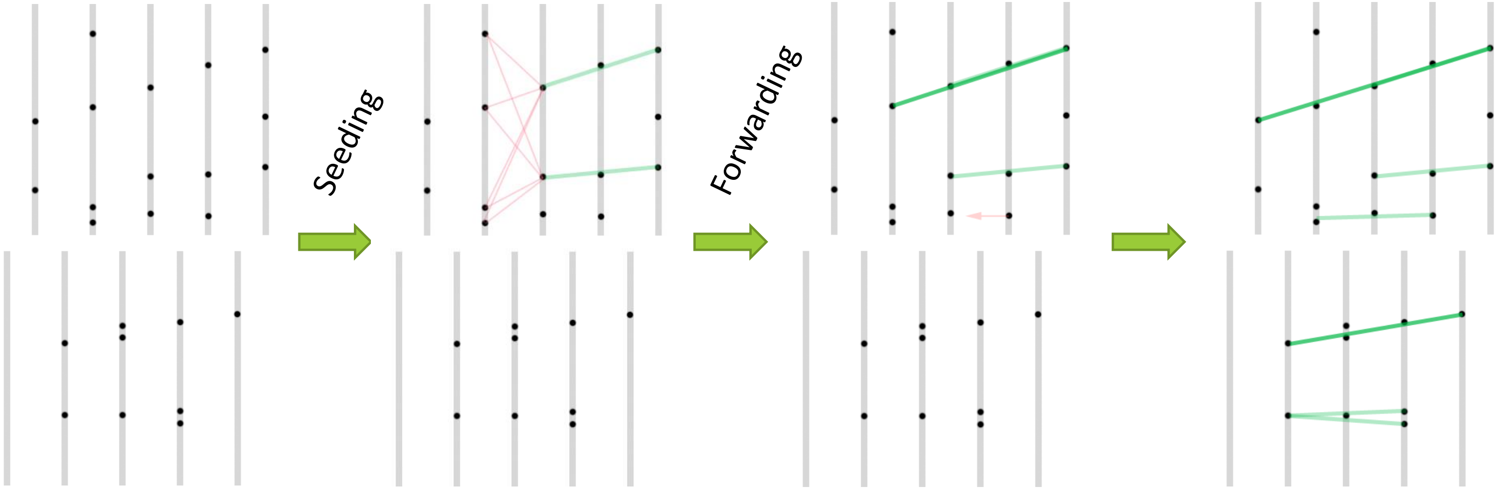




- Intel Xeon is still the predominant HW architecture in sci.comp. but can we use it more efficiently?
- Host-mode manycore processors (Knights Landing) with 100s of HW threads are around the corner, how can we scale that far?

VeloPixel track reconstruction

- Iterative algorithm that finds straight lines in collision event data in VeloPixel sub-detector.
    - Triplets of *hits* with best criterion are searched (seeding)
    - Triplets are extended to tracks if a fitting hit can be found



Daniel Campora, LHCb Computing Workshop (2015)

Using OpenMP and TBB for multilevel parallelism

- We would like to be able to compare our parallel code with a typical production run.

  —> we parallelize over events and within each event

- OpenMP uses nested parallelism, parallel for
- TBB For now mostly based on parallel_for
  - Also tested pipelining
- Used lock-free parallel implementations
  - TBB thread-safe data-structures did not perform well!

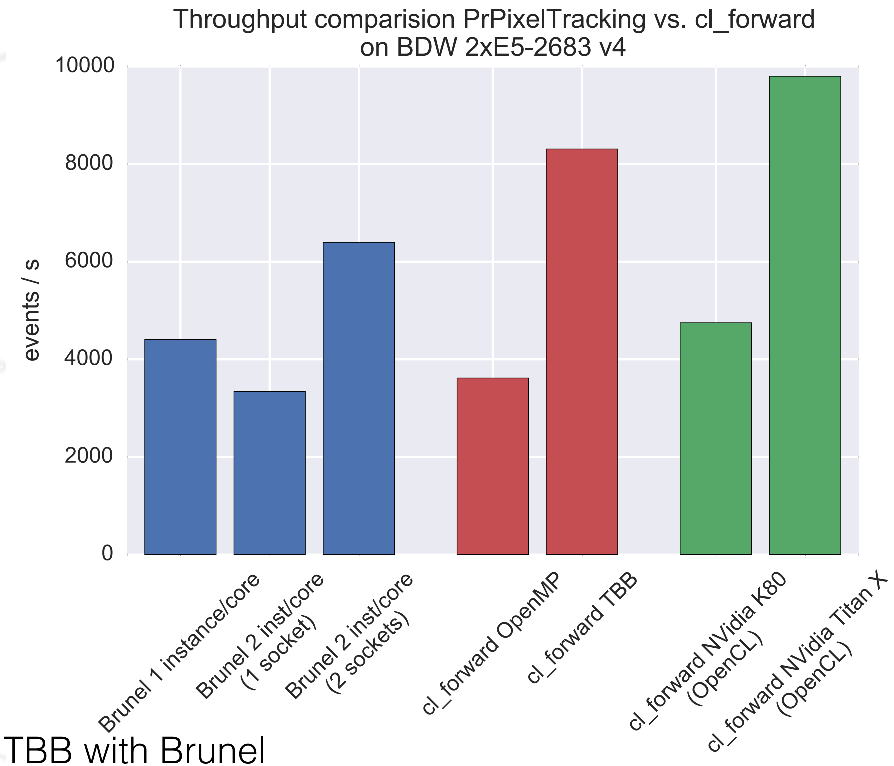
Results and Timings

Recovering track reconstruction efficiency

```
Production code aka Brunel (v50r0) PrPixel
2248492 tracks including    56641 ghosts (  2.5%). Event average   1.9%
              velo :  1937720 from  2105493 ( 92.0%)    44013 clones (  2.27%), purity: ( 99.81%),  hitEff: ( 95.40%)
              long :   672751 from   678628 ( 99.1%)    13556 clones (  2.02%), purity: ( 99.82%),  hitEff: ( 96.72%)
         long>5GeV :   446458 from   448535 ( 99.5%)     7731 clones (  1.73%), purity: ( 99.83%),  hitEff: ( 97.25%)
      long_strange :    27383 from    27846 ( 98.3%)      416 clones (  1.52%), purity: ( 99.33%),  hitEff: ( 97.51%)
 long_strange>5GeV :    13436 from    13679 ( 98.2%)      128 clones (  0.95%), purity: ( 99.16%),  hitEff: ( 98.35%)
        long_fromb :    38897 from    39148 ( 99.4%)      690 clones (  1.77%), purity: ( 99.78%),  hitEff: ( 97.15%)
   long_fromb>5GeV :    32074 from    32196 ( 99.6%)      537 clones (  1.67%), purity: ( 99.80%),  hitEff: ( 97.36%)
```

```
our code
2180404 tracks including    26268 ghosts (  1.2%). Event average   1.0%
              velo :  1923734 from  2105493 ( 91.4%)    30356 clones (  1.58%), purity: ( 99.77%),  hitEff: ( 96.06%)
              long :   671727 from   678628 ( 99.0%)     8266 clones (  1.23%), purity: ( 99.74%),  hitEff: ( 97.75%)
         long>5GeV :   445784 from   448535 ( 99.4%)     4672 clones (  1.05%), purity: ( 99.78%),  hitEff: ( 98.26%)
      long_strange :    27152 from    27846 ( 97.5%)      320 clones (  1.18%), purity: ( 99.21%),  hitEff: ( 97.81%)
 long_strange>5GeV :    13365 from    13679 ( 97.7%)      116 clones (  0.87%), purity: ( 99.06%),  hitEff: ( 98.55%)
        long_fromb :    38778 from    39148 ( 99.1%)      368 clones (  0.95%), purity: ( 99.70%),  hitEff: ( 97.94%)
   long_fromb>5GeV :    31989 from    32196 ( 99.4%)      275 clones (  0.86%), purity: ( 99.73%),  hitEff: ( 98.15%)
```

Timings

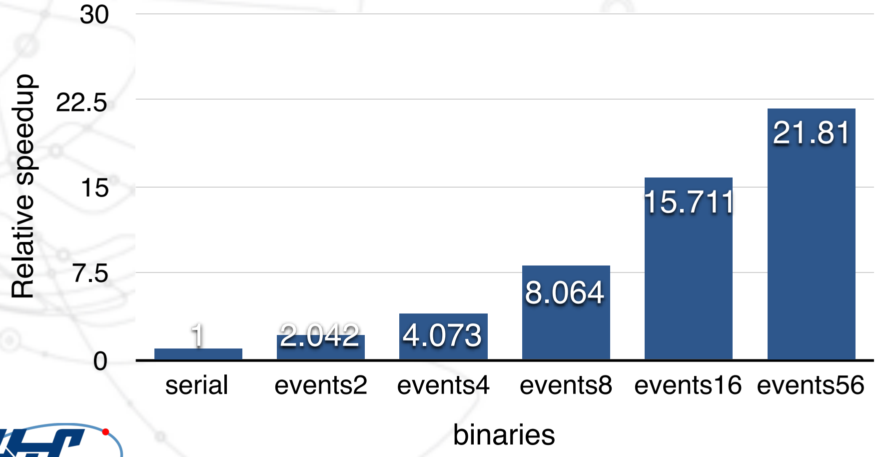


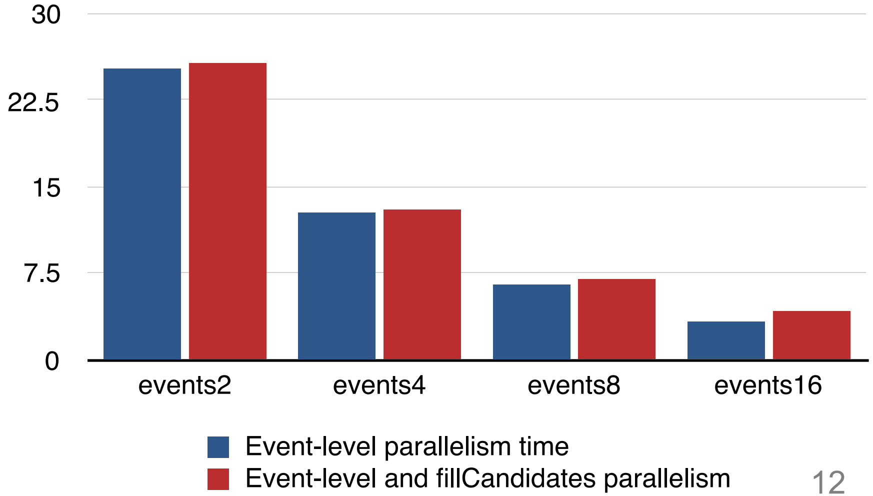
Comparing TBB with Brunel

- tbbPixel speedup no HT: 1.88
- tbbPixel speedup HT: 1.29

OpenMP Timings

- Runtime very sensitive to scheduling policies (dynamic vs static, granularities)
- Nested parallel regions often give a slow-down with respect to non-nested parallelism



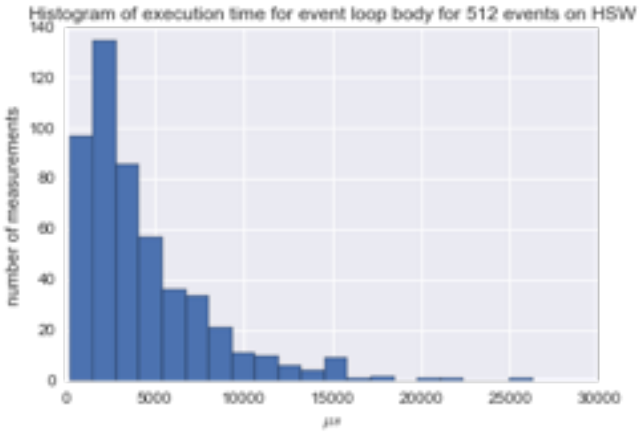


Scalability issues

- Scalability of tbbPixel (or ompPixel) is limited!
  - Event execution times vary by up to x50
    —> computational imbalance
- For now we mostly parallelized simple loops

  —> we are limited by Amdahl's law
- A majority of events are very small, loop trip-counts are very small

  —> overhead from multithreading can be significant



What next?

- Xeon-Phi Knights Landing:
  - We have started testing/benchmarking!
  - With 200+ threads scaling is a problem
- TBB Flow Graph or HPX?
  - Express our algorithm in terms of small concurrent tasks
  - Leave the rest up to scheduler
- How can we reduce computational imbalance?
  - Process “small” events only in serial freeing up resources for “big” events
- Understand scaling problems in OpenMP

Thank you!

Who are we:

CERN openlab High Throughput Computing Collaboration
Olof Bärring, Niko Neufeld
Omar Awile, Paolo Durante, Christian Färber, Karel Hà, Jon Machen (Intel), Rainer Schwemmer, Srikanth Sridharan, Paweł Szostek, Sébastien Valat, Balázs Vőneki

Backup

General structure of the code

```
for event in events:
    fillCandidates()
    for sensor in sensors://52 sensors
        trackForwarding()
        for hit in sensor.hits:
            trackCreation()
        for track in tracks:
            do_some_stuff_1()
```

```
fillCandidates():
    for sensor in sensors:
        for hit in sensor.hits:
            for hit2 in sensor.next().hits:
                do_some_stuff_2()
```

```
trackForwarding():
    for track in tracks:
        for hit in sensor.hits:
            do_some_stuff_3()
```

```
trackCreation():
    for hit in sensors[s].hits:
        for hit2 in sensors.next().hits:
            do_some_stuff_4()
```

LHCb upgrade (2020)

- LHCb studies CP-violation, rare decays, …
- The upgrade:
  - Currently readout of detector @ 1MHz
    After upgrade: 40MHz
  - Redesign of DAQ system
  - yield > 10x more events