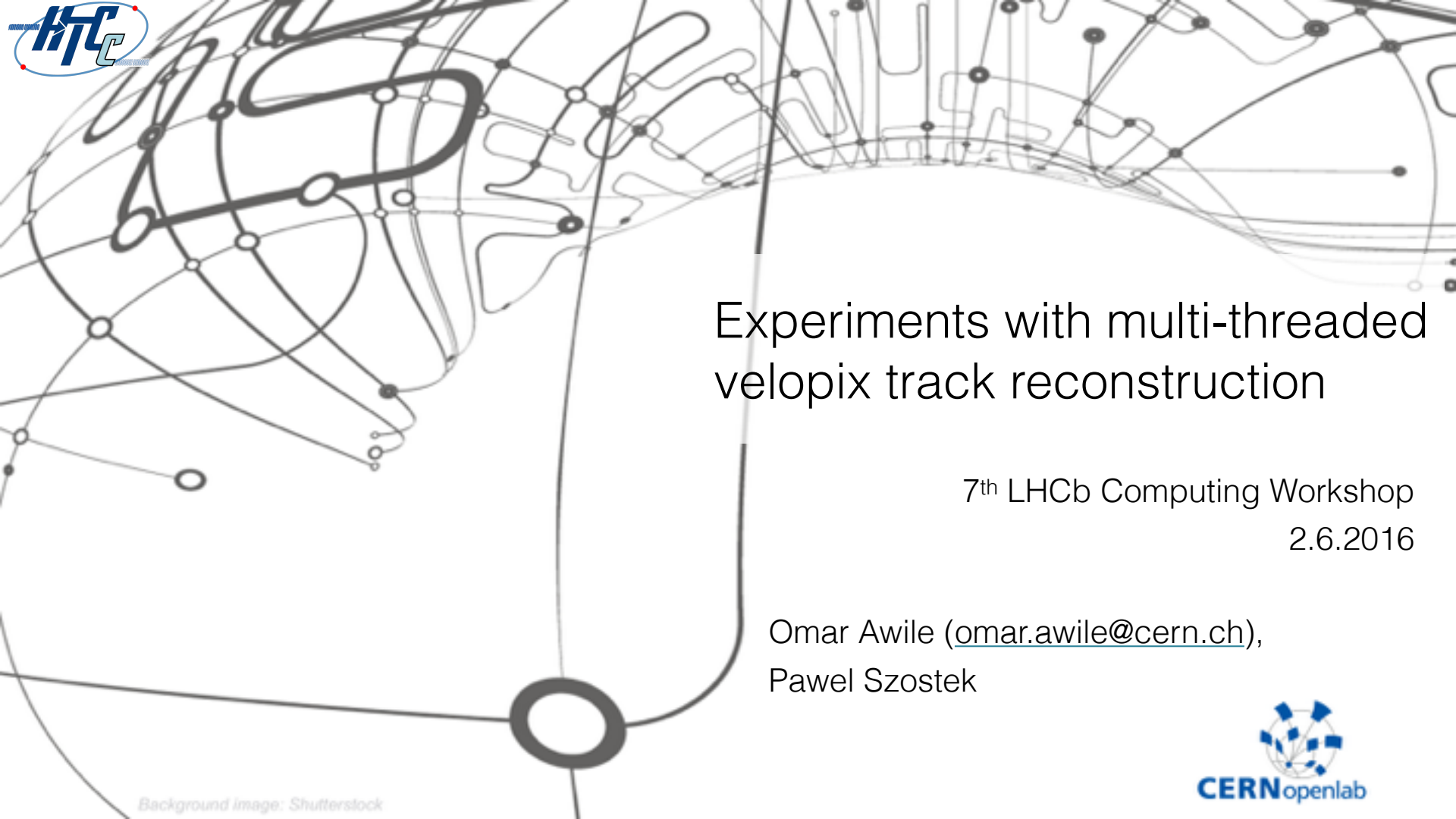


Experiments with multi-threaded velopix track reconstruction

7th LHCb Computing Workshop

2.6.2016

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

Pawel Szostek

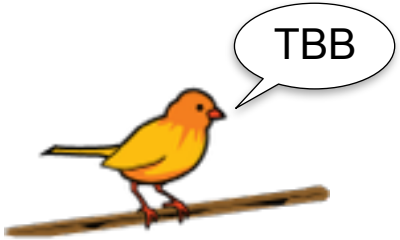
CERNopenlab

Background image: Shutterstock

Some context… and motivation

- We want to explore how velopixel track reconstruction can be done on multi- and manycore CPUs - using multithreading.

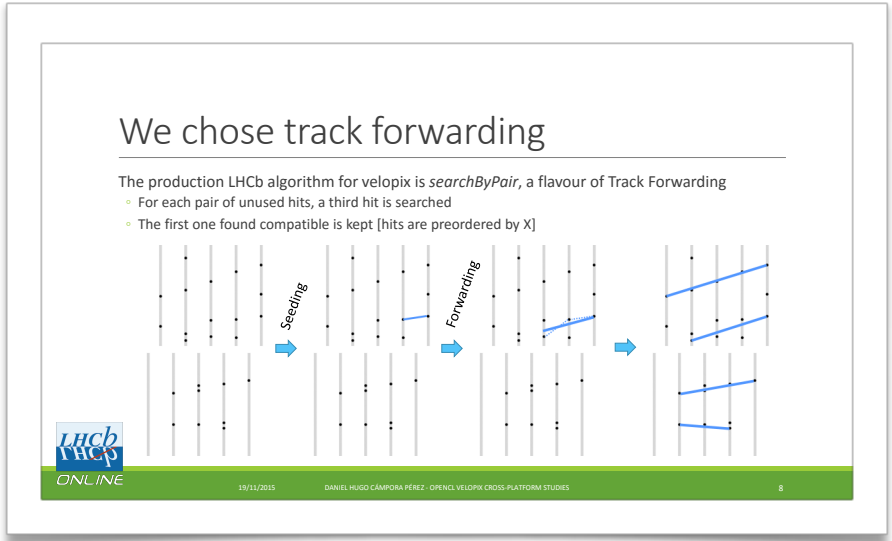
OpenMP



- 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?

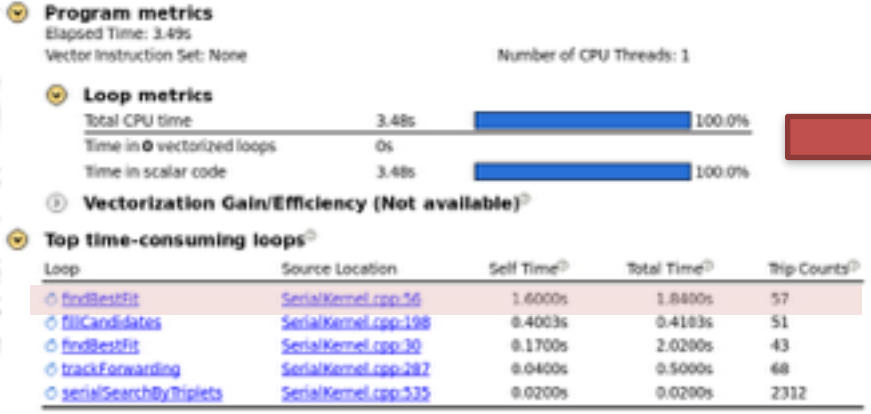
Let's not start from scratch

- We ported Daniel Campora's clPixel to serial C++ for a baseline
- From there experimented with
  - OpenMP
  - TBB
  - vectorization



vectorization

- We found a hotspot! but…
  - loop is small and contains a reduction
- Use openmp-simd reductions
- Other loops…. difficult
  - e.g. fillCandidates loop has multiple exits



Program metrics
Elapsed Time: 3.49s
Vector Instruction Set: None
Number of CPU Threads: 1

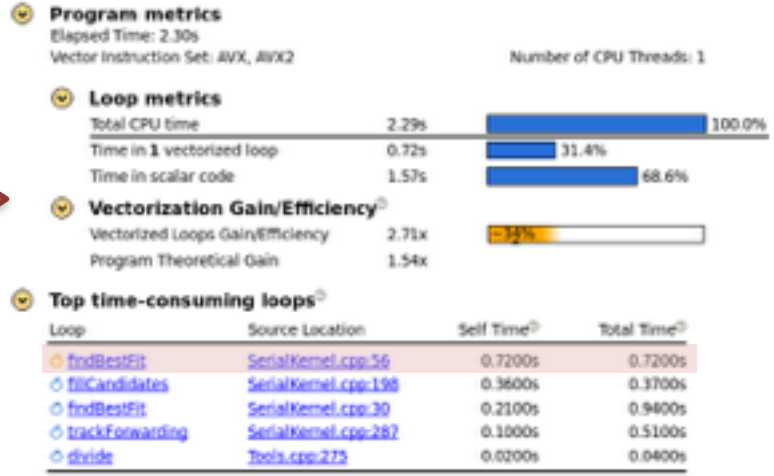
Loop metrics

Total CPU time	3.48s	100.0%
Time in 0 vectorized loops	0s	
Time in scalar code	3.48s	100.0%

Vectorization Gain/Efficiency (Not available)

Top time-consuming loops

Loop	Source Location	Self Time	Total Time	Trip Counts
findBestFit	SerialKernel.cpp:56	1.6000s	1.8400s	57
fillCandidates	SerialKernel.cpp:198	0.4003s	0.4103s	51
findBestFit	SerialKernel.cpp:30	0.1700s	2.0200s	43
trackForwarding	SerialKernel.cpp:287	0.0400s	0.5000s	68
serialSearchByTriplets	SerialKernel.cpp:535	0.0200s	0.0200s	2312



Program metrics
Elapsed Time: 2.30s
Vector Instruction Set: AVX, AVX2
Number of CPU Threads: 1

Loop metrics

Total CPU time	2.29s	100.0%
Time in 1 vectorized loop	0.72s	31.4%
Time in scalar code	1.57s	68.6%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	2.71x	34%
Program Theoretical Gain	1.54x	

Top time-consuming loops

Loop	Source Location	Self Time	Total Time
findBestFit	SerialKernel.cpp:56	0.7200s	0.7200s
fillCandidates	SerialKernel.cpp:198	0.3600s	0.3700s
findBestFit	SerialKernel.cpp:30	0.2100s	0.9400s
trackForwarding	SerialKernel.cpp:287	0.1000s	0.5100s
divide	Tools.cpp:275	0.0200s	0.0400s

Some OpenMP experiments

- Idea: “inject” **nested parallel regions** at different iteration levels
- Manipulate them using C macros (turning parallelism on and off, changing **number of threads** and **scheduling policies**)
- Find the best settings by **exploring** the **parameter space**

Using 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
- For now mostly based on TBB parallel_for
  - Also tested pipelining
- Used lock-free parallel implementations
  - TBB thread-safe data-structures did not perform well!

Results and Timings

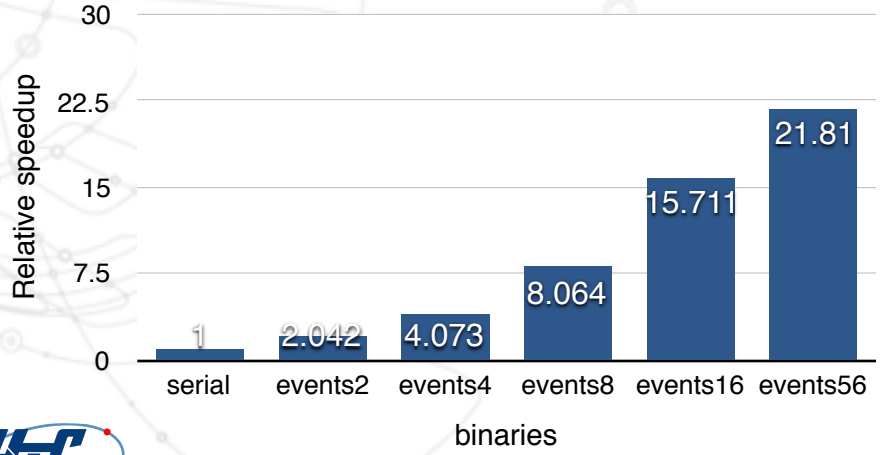
Making sure results are OK

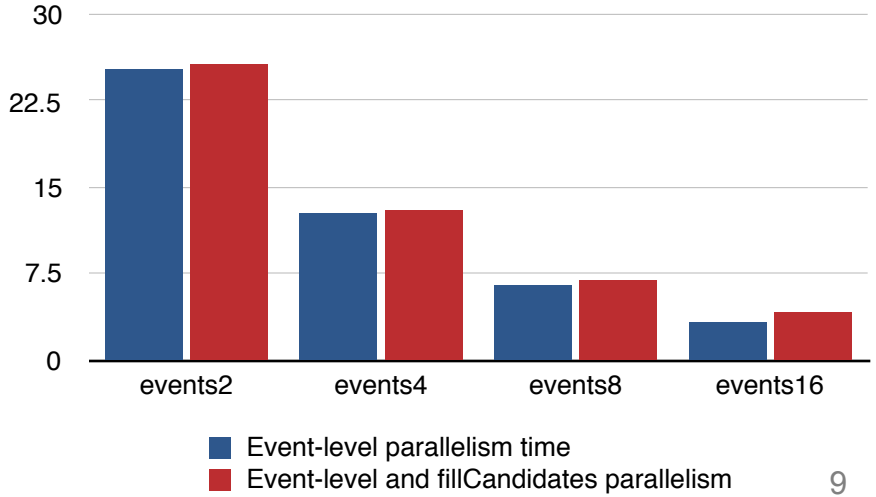
```
Brunel (v50r0) PrPixel
2248492 tracks including    56641 ghosts (  2.5%). Event average   1.9%
              velo :  1937720 from  2105493 ( 92.0%)    44013 clones (  2.27%), purity: ( 99.81%),  hitEff: ( 95.40%)
              long :   672751 from   678628 ( 99.1%)    13556 clones (  2.02%), purity: ( 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%)


(tbb|omp)Pixel
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%)
```

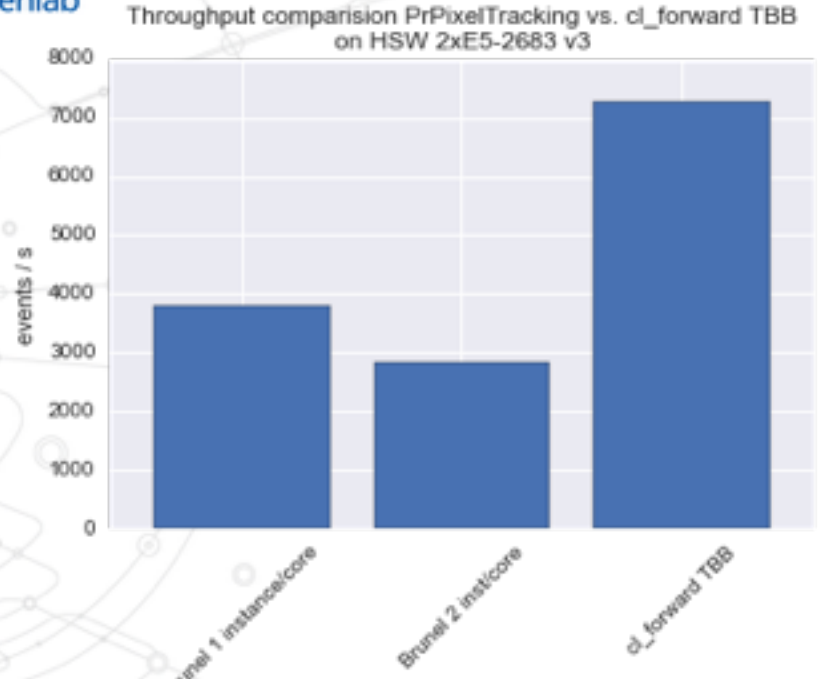
OpenMP Timings

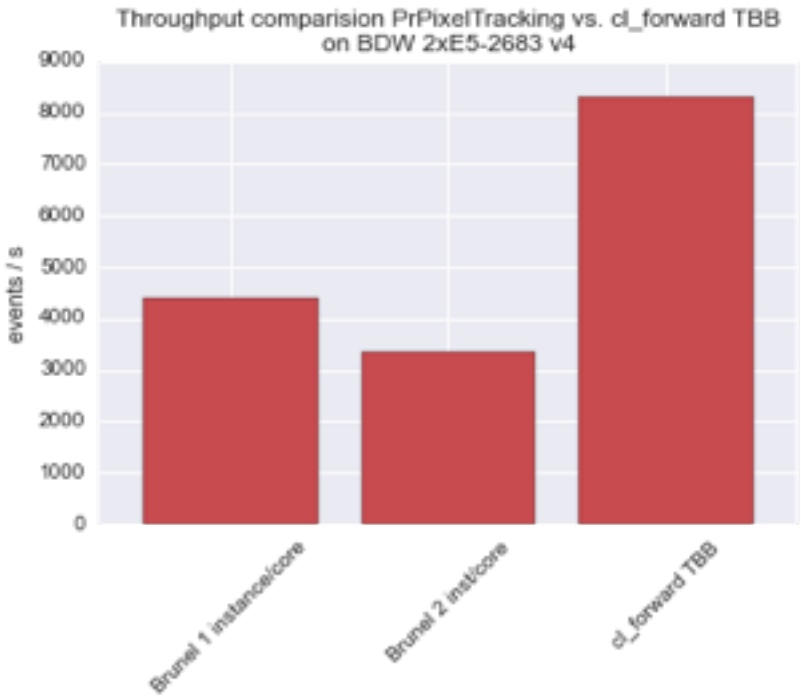
- Runtime very sensitive to scheduling policies (dynamic vs static, granularities)
- Nested parallel regions often give a slow-down with respect to non-nested parallelism





TBB Timings



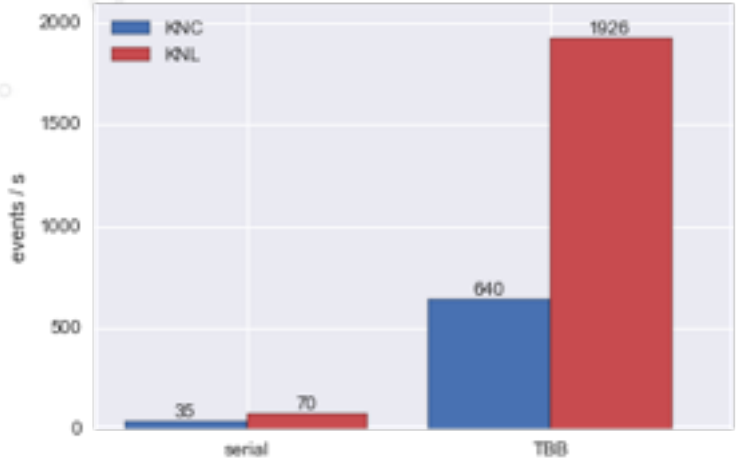


Comparing TBB with Brunel without HT (production?)

- tbbPixel speedup on HSW: 1.84
- tbbPixel speedup on BDW: 1.88

tbbPixel on the Xeon-Phi

- Very preliminary!
- When compared with KNC, KNL shows a big boost!
- Comparing with Xeon is not that easy
  - Current code does not scale to KNL (or KNC) :(



What we've learned

vectorization

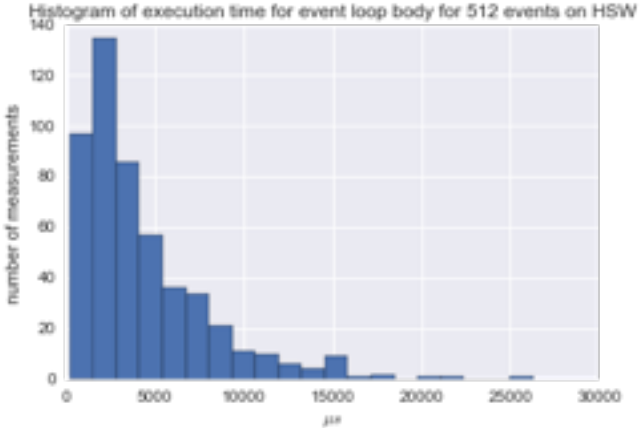
- If you can, use the Intel tools!
  - `icpc -qopt-report=5`
    Generated reports are very wordy, but can give valuable hints on where it is worth vectorizing and what could be tried
  - Intel Advisor
    Comprehensive tool for code vectorization and threading analysis

Parallelization strategies

- Scalability of tbbPixel (or ompPixel) is limited!
  - Event execution times vary by up to x1000
    —> 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



Bits and pieces

Data Generation

- For rapid prototyping we want to break out of LHCb software stack.
  - Still work with “real” data
- PrEventDumper: https://gitlab.cern.ch/oawile/PrEventDumper
- The algorithm can be controlled with a Brunel configurable parameter to output only (velopix) data or MC particle and track data (e.g. for validation).

Result validation

- Needed a simple track validation tool
- Also:
  - should be independent of Brunel
  - should be extendible
  - should work with flat data format
- EventAnalyzer: https://gitlab.cern.ch/oawile/EventAnalyzer
  - Written in python
  - returns validation in format similar to PrChecker

```
$ python2.7 validator.py -v -f results.txt
Reading data:
 done.
2248492 tracks including    56641 ghosts (  2.5%). Event average   1.9%
              velo :  1937720 from  2105493 ( 92.0%,  92.0%)  44013 clones (  2.27%), purity: ( 99.81%,  99.84%), hitEff: ( 95.40%,  95.34%)
              long :   672751 from   678628 ( 99.1%,  99.2%)  13556 clones (  2.02%), purity: ( 99.82%,  99.84%), hitEff: ( 96.72%,  96.67%)
         long>5GeV :   446458 from   448535 ( 99.5%,  99.5%)   7731 clones (  1.73%), purity: ( 99.83%,  99.86%), hitEff: ( 97.25%,  97.18%)
      long_strange :    27383 from    27846 ( 98.3%,  98.4%)    416 clones (  1.52%), purity: ( 99.33%,  99.38%), hitEff: ( 97.51%,  97.15%)
 long_strange>5GeV :    13436 from    13679 ( 98.2%,  98.2%)    128 clones (  0.95%), purity: ( 99.16%,  99.21%), hitEff: ( 98.35%,  98.04%)
        long_fromb :    38897 from    39148 ( 99.4%,  99.4%)    690 clones (  1.77%), purity: ( 99.78%,  99.84%), hitEff: ( 97.15%,  96.83%)
   long_fromb>5GeV :    32074 from    32196 ( 99.6%,  99.6%)    537 clones (  1.67%), purity: ( 99.80%,  99.86%), hitEff: ( 97.36%,  97.04%)
```

What next?

- 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!

Resources:

cl_forward: https://gitlab.cern.ch/oawile/cl_forward

PrEventDumper: https://gitlab.cern.ch/oawile/PrEventDumper

EventAnalyzer: https://gitlab.cern.ch/oawile/EventAnalyzer

Data format: https://gitlab.cern.ch/oawile/EventAnalyzer/blob/master/DATAFORMAT.md

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()
```