Software framework for the LHCb upgrade

Sébastien Ponce, Stefan Roiser, Concezio Bozzi, on behalf of the LHCb Collaboration
sebastien.ponce@cern.ch
Outline

Context

Main directions

Monitoring the progress

Current state

References
LHCb Run 3 landscape

• Upgrade of the detector itself to take more luminosity (x5)
  • still 30MHz collisions
  • more pile-up (now 5.5, was 1.1)
LHCb Run 3 landscape

- Upgrade of the detector itself to take more luminosity (×5)
  - still 30MHz collisions
  - more pile-up (now 5.5, was 1.1)
- New trigger system
  - no hardware, fully software
  - input rate ×30!

### Experiment

**Hardware trigger**
- 1MHz
- Software Trigger
  - 2 levels (HLT1, HLT2)

**Software framework for the LHCb upgrade**

---

Run2

- 30MHz
- Hardware trigger
- Software Trigger
  - 2 levels (HLT1, HLT2)

---

Run3

- 30MHz
- Software Trigger
  - 2 levels (HLT1, HLT2)
How to make it possible?

• Hardware evolution\(^1\)
  • Moore’s law still holding
  • in numbers of transistors

• Better use of hardware
  • more parallelization
  • many cores, vectorization, hyperscalar features, ...

• Improving our software
  • using latest programming techniques
  • improving our memory accesses
  • reengineering some algorithms\([1]\)

---

\(^1\) Data source: https://github.com/karlrupp/microprocessor-trend-data, modified to only show transistors
Adapting to many-core

- Multi-core is over, we have now many-core\(^2\)
  - easily 40, up to 100 logical CPU cores
- Consequently multi-process is becoming hard
  - as memory pressure is too high
- LHCb software has opted for multithreading
  - parallelizing at the granularity of events
  - running as many events in parallel as cores

\(^2\)Data source: https://github.com/karlrupp/microprocessor-trend-data, modified to only show transistors
How to handle thread safety?

- Use a thread safe framework
  - Gaudi hive, and its new “functional” algorithms
- Rely on modern C++ standards
  - making extensive use of constness in particular
- Go step by step through the code
  - starting small, learning the techniques
  - spreading the knowledge
Optimization opportunities

- Careful memory usage
  - as memory has become slow compared to CPU
  - and thus cache misses are costful

- Vectorization
  - vector units are getting wider
  - but making good usage of them requires rethinking our data models

- C++ new features
  - many are related to optimization
Spreading the knowledge

- Regular tutorial sessions in the collaboration
  - core C++ programming
  - LHCb framework and coding practices
  - optimization tools, collaborative tools
- Regular hackathons
  - with support of experts
Monitoring the progress

- Many tools available to measure performance
  - including internal tracing of the code
- Regular (nightly) builds and runs[5]
  - checking validity on all platform
  - checking raw performance
  - recording all performance indicators
    - e.g. cache misses, vectorization levels
- all history available to extract trends
lhcb-head - build: 1900 (2018-06-28)

head of everything against Gaudi mastermaster and LCG_93 (ROOT 6.12.06)
available on: cvmfs

<table>
<thead>
<tr>
<th>Project</th>
<th>Version</th>
<th>x86_64-sl6-gcc7-opt</th>
<th>x86_64-sl6-gcc7-dbgs</th>
<th>x86_64-centos7-gcc7-opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCG</td>
<td>93</td>
<td>build</td>
<td>tests</td>
<td>build</td>
</tr>
<tr>
<td>Gaudi</td>
<td>master</td>
<td>build</td>
<td>tests</td>
<td>build</td>
</tr>
<tr>
<td>Online</td>
<td>HEAD</td>
<td>build</td>
<td>tests</td>
<td>build</td>
</tr>
<tr>
<td>LHCb</td>
<td>HEAD</td>
<td>build</td>
<td>tests</td>
<td>build</td>
</tr>
<tr>
<td>Lbcom</td>
<td>HEAD</td>
<td>build</td>
<td>tests</td>
<td>build</td>
</tr>
<tr>
<td>Boole</td>
<td>HEAD</td>
<td>build</td>
<td>tests</td>
<td>build</td>
</tr>
<tr>
<td>Rec</td>
<td>HEAD</td>
<td>build (24)</td>
<td>tests</td>
<td>build</td>
</tr>
<tr>
<td>Brunel</td>
<td>HEAD</td>
<td>tests (3)</td>
<td>build (21)</td>
<td>build</td>
</tr>
</tbody>
</table>
Monitoring the progress

- Many tools available to measure performance
  - including internal tracing of the code
- Regular (nightly) builds and runs[5]
  - checking validity on all platform
  - checking raw performance
  - recording all performance indicators
    - e.g. cache misses, vectorization levels
- all history available to extract trends
Monitoring the progress

- Many tools available to measure performance
  - including internal tracing of the code
- Regular (nightly) builds and runs[5]
  - checking validity on all platform
  - checking raw performance
  - recording all performance indicators
    - e.g. cache misses, vectorization levels
- all history available to extract trends
What we’ve achieved so far

- 2016: first thread safe code in LHCb
- May 2017: first benchmark with multithreading
  - 500 evts/s/node
- November 2017: scalable code
  - 3000 evts/s/node
- Feb 2018: realistic reconstruction with cuts
  - 12400 evts/s/node
- June 2018: latest improvements
  - 14800 evts/s/node
The graph shows the relation between the level of parallelization (number of threads times number of jobs) and the events per second (events/s). The y-axis represents the number of events per second, ranging from 0 to 3000. The x-axis represents the level of parallelization, ranging from 0 to 60.

Different lines and markers represent various parallelization levels and strategies:

- **non Hive** (black dashed line)
- **Hive MJ** (grey line)
- **best perf** (black solid line)
- **12 thr/job** (green square)
- **16 thr/job** (green circle)
- **20 thr/job** (orange square)
- **24 thr/job** (orange circle)
- **28 thr/job** (red square)
- **32 thr/job** (red circle)

The graph illustrates how the efficiency of different parallelization strategies changes with varying levels of parallelization.
What we’ve achieved so far

• 2016 : first thread safe code in LHCb
• May 2017 : first benchmark with multithreading
  • 500 evts/s/node
• November 2017 : scalable code
  • 3000 evts/s/node
• Feb 2018 : realistic reconstruction with cuts
  • 12400 evts/s/node
• June 2018 : latest improvements
  • 14800 evts/s/node
Max throughput for 20 threads, 2 jobs = 12400.3 evt/s/node

Events / s / node

Level of parallelization (nb threads x nb jobs)
What we’ve achieved so far

• 2016 : first thread safe code in LHCb
• May 2017 : first benchmark with multithreading
  • 500 evts/s/node
• November 2017 : scalable code
  • 3000 evts/s/node
• Feb 2018 : realistic reconstruction with cuts
  • 12400 evts/s/node
• June 2018 : latest improvements
  • 14800 evts/s/node
Current state: halfway there

- Infrastructure is in place
- Training effort is starting to pay off
- We have a running HLT1 sequence
- Computing performance has improved a lot
  - we started around 3 KHz/node
  - we are 8 to 15 KHz/node depending on cuts
The road to 30MHz

- Ongoing effort needs to continue
  - getting more and more momentum thanks to knowledge spread
- HLT1 and core framework are not all of it
  - trigger selections need to be reworked [7]
  - HLT2 needs the same optimizations as HLT1
  - conditions [3], geometry [4, 6], scheduling, .... [8, 2]
  - the event model has to be reviewed
- All this is already ongoing
- Expected to pay off in some months
Conclusion

- LHCb’s software is undergoing a major modernization to face Run3 challenges
- Best usage of modern hardware and parallelization are the main directions
- On top of proactive dissemination of knowledge
- Light is visible at the end of the tunnel
  - and is growing steadily!
References (I)

Johannes Albrecht, Ben Couturier, Christoph Hasse, and Sebastien Ponce.
New approaches for track reconstruction in Lhcb’s vertex locator.

Concezio Bozzi and Stefan Roiser.
Towards a computing model for the Lhcb upgrade.

Marco Clemencic.
A git-based conditions database backend for Lhcb.

Ben Couturier, Chris Burr, Lucia Grillo, Marco Clemencic, Markus Frank, and Silvia Borghi.
Perspectives for the migration of the Lhcb geometry to the dd4hep toolkit.
References (II)

Ben Couturier and Maciej Pawel Szymanski.
Improvements to the lhcb software performance testing infrastructure using message queues and big data technologies.

Agnieszka Dziurda, Francesco Polci, Giulio Dujany, Jean-Francois Marchand, Lucia Grillo, and Paras Naik.
Lhcb full-detector real-time alignment and calibration: latest developments and perspectives.

Conor Fitzpatrick, Rosen Matev, and Sascha Stahl.
A 30mhz software trigger for the lhcb upgrade.

Aritz Brosa Iartza, Ben Couturier, Laura Promberger, Marco Clemencic, and Niko Neufeld.
Porting the lhcb stack from x86 (intel) to aarch64 (arm).