Porting the LHCb Stack from x86 (Intel) to aarch64 (ARM)

CHEP 2018, Sofia

Laura Promberger\textsuperscript{1} 2
Marco Clemencic\textsuperscript{1}
Ben Couturier\textsuperscript{1}
Aritz Brosa Iartza\textsuperscript{1} 3
Niko Neufeld\textsuperscript{1}

on behalf of the LHCb collaboration

July 12, 2018

\textsuperscript{1}CERN
\textsuperscript{2}Hochschule Karlsruhe - Technik und Wirtschaft
\textsuperscript{3}Universidad de Oviedo (ES)
Motivation - The Upgrade In 2021

<table>
<thead>
<tr>
<th></th>
<th>Currently (Run 2)</th>
<th>Upgrade (Run 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data acquisition rate</td>
<td>50 GB/s</td>
<td>4 TB/s</td>
</tr>
<tr>
<td>Data recording rate</td>
<td>0.7 GB/s</td>
<td>2 - 10 GB/s</td>
</tr>
</tbody>
</table>

For the upgrade

- Software needs major refactoring and usage of new technology
- New HLT farm

Goal

- Add cross-platform support to the LHCb stack
  → More flexibility with the tender for the new HLT farm
  ⇧ Biggest Problem: Vectorization
The LHCb Stack

- 5 million lines of code (experiment-specific projects)
- Multiple, large projects

For this work
- Old version of the LHCb stack (Oct 2017)
- Not multi-threaded

Structure of the stack

- Experiment-specific
- Experiment-independent
- External dependencies (LCG)
## Vectorization

<table>
<thead>
<tr>
<th>Platform</th>
<th>Vcl</th>
<th>Vc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel AVX2</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Intel AVX512</td>
<td>Yes</td>
<td>In development</td>
</tr>
<tr>
<td>PowerPC Altivec</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ARM NEON</td>
<td>No</td>
<td>In development</td>
</tr>
</tbody>
</table>

**Vectorization Style**
- Wrapper for intrinsics
- High-level, targets horizontal vectorization

**Extensibility for new intrinsics**
- Medium (no unit tests)
- Complex

→ Vcl allows 'fast' implementation of other platforms
Port to aarch64 (ARM)

LCG requires

- Changing compile flags
  - e.g. replace `-max-page-size=0x1000` by `-common-page-size=0x1000`
- Changing versions of the external dependencies
- Disabling unnecessary packages (e.g. Oracle, R)

Other projects

- Changing compile flags
- Replacement of Vc by
  - Vcl
  - Scalar code
Port to aarch64 - Problems

Default signedness of char

- Intel uses signed char
- ARM uses unsigned char

→ Use -fsigned-char to change the default to signed char

```c
// Jenkins one-at-time hash function
static unsigned int hash32( const char* key )
{
    unsigned int hash = 0;
    for ( const char* k = key; *k; ++k ) {
        hash += *k;
        hash += ( hash << 10 );
        hash ^= ( hash >> 6 );
    }
    hash += ( hash << 3 );
    hash ^= ( hash >> 11 );
    hash += ( hash << 15 );
    return hash;
}
```
Cast double to unsigned int

- Intel assembly uses `vcvttsd2si`
- ARM assembly uses `fcvtzu`

```c
if (m_xInverted == true) {
    strip = (unsigned int) floor((m_uMaxLocalu/m_pitch) +0.5);
}
```

```c
float x = -3.3;
unsigned int y = (unsigned int) x;
```

**Problem**

```c
float x = -3.3;
uint32_t y = static_cast<uint32_t> (static_cast<int> (x));
```

**Solution**
## Performance - The machines

<table>
<thead>
<tr>
<th></th>
<th>ThunderX2</th>
<th>E5-2630 v4</th>
<th>Power8+</th>
<th>Power9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture</strong></td>
<td>ARM</td>
<td>Intel</td>
<td>PowerPc</td>
<td>PowerPc</td>
</tr>
<tr>
<td><strong>Platform</strong></td>
<td>aarch64</td>
<td>x86_64</td>
<td>ppc64le</td>
<td>ppc64le</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>GCC 7.2</td>
<td>GCC 6.2</td>
<td>GCC 7.3</td>
<td>GCC 7.3</td>
</tr>
<tr>
<td><strong>Number logical cores</strong></td>
<td>224</td>
<td>40</td>
<td>128</td>
<td>176</td>
</tr>
<tr>
<td><strong>Threads per core</strong></td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td><strong>Cores per socket</strong></td>
<td>28</td>
<td>10</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td><strong>Sockets/NUMA nodes</strong></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>RAM (GB)</strong></td>
<td>256</td>
<td>64</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td><strong>Largest intrinsic set</strong></td>
<td>NEON</td>
<td>AVX2</td>
<td>Altivec</td>
<td>Altivec</td>
</tr>
<tr>
<td><strong>CPU performance</strong></td>
<td>top-notch</td>
<td>cost-efficient</td>
<td>high-tier</td>
<td>mid-tier</td>
</tr>
</tbody>
</table>
Performance - Scalability of the LHCb Stack

![Graph showing performance scalability of LHCb Stack across different systems.](image-url)
Scalability II - Cost-Performance Estimations

![Cost Performance Estimations Diagram]

- E5-2630 v4 logical cores 40 processes
- E5-2630 v4 logical cores 48 processes
- Thunder X2 logical cores 224 150 processes
- Thunder X2 logical cores 224 200 processes
- Thunder X2 logical cores 224 224 processes (extrapolation)
Outlook

- Long-term goal: Adding cross-platform support to the Run 3 LHCb stack
  - Requires a fully functioning cross-platform vectorization library

- Finding a cross-platform vectorization library
  - ROOT plans to use VecCore which has both, UMESIMD and Vc as back end
    → LHCb evaluates to switch to VecCore instead of Vc and Vcl

- New vectorization intrinsic set for ARM: SVE
  - First official date for CPU release: Fujitsu - 2021
    → Too late for LHCb Run 3
Summary

- Cross-platform support of the LHCb stack for aarch64 and ppc64le
- Biggest problem: Vectorization
  - "Hackish" workarounds of Vc just for this study
- Cost-performance estimation
  - To be considered: pricing, not multi-threaded, less vectorization on aarch64
  - ARM and Intel quite close
    → Competitive tender for real evaluation necessary
Questions?
## Vectorization

<table>
<thead>
<tr>
<th></th>
<th>Vcl</th>
<th>Vc</th>
<th>UMESIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>AVX2</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Intel</td>
<td>AVX512</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PowerPC</td>
<td>Altivec</td>
<td>No</td>
<td>Early Example</td>
</tr>
<tr>
<td>ARM</td>
<td>NEON</td>
<td>No</td>
<td>Early Example</td>
</tr>
</tbody>
</table>

- **Vectorization Style**: Wrapper for intrinsics
- **Extensibility for new intrinsics**: Medium (no unit tests)

- **High-level, targets horizontal vectorization**
- **Wrapper for intrinsics**
- **easy (unit tests available)**
Performance - Scalability of the LHCb Stack normalized

![Graph showing scalability of the LHCb Stack normalized with different hardware configurations: Thunder X2, Gcc 7.2, CentOS; E5-2630 v4, Gcc 6.2, CentOS; POWER8+, Gcc 7.3, CentOS; POWER9, Gcc 7.3, RHEL.](image-url)