ATLAS Offline Software
Performance Monitoring and Optimization

Neelima Chauhan, Gunjan Kabra, Thomas Kittelmann, Robert Langenberg, Rocco Mandrysch, Andreas Salzburger, Rolf Seuster, Elmar Ritsch, Graeme Stewart, Niels van Eldik, Roberto Vitillo

CHEP 2013
14.10 - 18.10.2013
1. Introduction
2. Performance measurements
3. Performance comparison studies
4. Summary and Conclusion
Why do we need software performance improvements?

- LHC will run again in 2015 with:
  - ATLAS trigger rate: $\sim 1$ kHz (2012: $\lesssim 400$ Hz)
  - 14 TeV and 25 ns bunch spacing
  - average 25 to 40 interactions per bunch crossing

- For reconstruction software:
  - processing time per event needs to be improved substantially
  - memory consumption needs to be decreased

- Several projects for speed improvement are in progress

- Performance studies of two of these projects will be discussed here
Performance measurements

- Profiling shows highest CPU/event consumers are algorithms for track reconstruction.
- Several math operations are executed in these algorithms:
  - vector/matrix operations via CLHEP framework
  - trigonometric functions via standard GNU libm math libraries
- The operations were monitored via:
  - Intel Pin tool
  - PAPI
- For further studies: CPU performance comparison between different math libraries were made in simple test framework.
## Linear algebra libraries

<table>
<thead>
<tr>
<th>CLHEP</th>
<th>Eigen</th>
<th>SMatrix</th>
<th>Intel Math Kernel Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>• C++ utility classes for HEP</td>
<td>• C++ templates (headers only)</td>
<td>• Implemented in ROOT as expression templates</td>
<td>• BLAS and LAPACK</td>
</tr>
<tr>
<td>• support SIMD vectorization</td>
<td></td>
<td></td>
<td>• highly optimized</td>
</tr>
</tbody>
</table>

- All libraries support matrices and vectors with all sizes
- CLHEP is not maintained anymore and known to be not well performed
Performance measurements
Monitoring tools

Pin tool

- Dynamic binary instrumentation framework
  - includes API for abstracting underlying instruction set idiosyncracies
    → no recompilation needed
  - can inject code at the level of functions or instructions

- Underlying tool used by Intel Parallel Inspector and Amplifier

- http://www.pintool.org/
Results with Pin of CLHEP functions

- Monitor calls of CLHEP functions:
  - during reconstruction job
  - with 2012 data sample
    (events passed any Jet, Tau or Missing ET trigger chain)

Five CLHEP functions with highest number of calls:

<table>
<thead>
<tr>
<th>Function</th>
<th>Calls/Evt</th>
</tr>
</thead>
<tbody>
<tr>
<td>HepVector::~HepVector()</td>
<td>3691535</td>
</tr>
<tr>
<td>HepSymMatrix::HepSymMatrix(HepSymMatrix const &amp;)</td>
<td>1702193</td>
</tr>
<tr>
<td>HepVector::HepVector(int, int)</td>
<td>1593544</td>
</tr>
<tr>
<td>operator*(HepMatrix const&amp;, HepSymMatrix const&amp;)</td>
<td>93120</td>
</tr>
<tr>
<td>operator*(HepMatrix const&amp;, HepVector const&amp;)</td>
<td>42918</td>
</tr>
</tbody>
</table>
Results with Pin of CLHEP functions

'HepMatrix*HepSymMatrix' arguments with highest number of calls:

<table>
<thead>
<tr>
<th>1st Argument</th>
<th>2nd Argument</th>
<th>Calls/Evt</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3</td>
<td>3 × 3</td>
<td>29333</td>
</tr>
<tr>
<td>3 × 2</td>
<td>2 × 2</td>
<td>28139</td>
</tr>
<tr>
<td>3 × 5</td>
<td>5 × 5</td>
<td>13003</td>
</tr>
</tbody>
</table>

'HepSymMatrix*HepVector' arguments with highest number of calls:

<table>
<thead>
<tr>
<th>1st Argument</th>
<th>2nd Argument</th>
<th>Calls/Evt</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 × 3</td>
<td>3</td>
<td>23676</td>
</tr>
<tr>
<td>3 × 5</td>
<td>3</td>
<td>11802</td>
</tr>
<tr>
<td>1 × 5</td>
<td>5</td>
<td>4718</td>
</tr>
</tbody>
</table>
PAPI (Performance API)

- Platform-independent interface for hardware performance counters such as: floating point instructions, level 1 cache misses, single/double precision vector/SIMD instructions
- Contains low- and high-level sets of routines for accessing counters:
  - low level: controls and provides access to all counters
  - high level: easily allows one to start, stop and read the counters
- http://icl.cs.utk.edu/papi/
Results with PAPI of matrix/vector operations

- Monitor in a simple test framework floating point operation of several matrix/vector calls with PAPI
- Compare CLHEP with other classes: Eigen and SMatrix

Floating operations of 3-dimensional vector/matrix calls:

<table>
<thead>
<tr>
<th>Operations</th>
<th>CLHEP</th>
<th>Eigen</th>
<th>SMatrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix allocation</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Vector allocation</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Vector + Vector</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Matrix × Vector</td>
<td>18</td>
<td>15</td>
<td>n./a.</td>
</tr>
<tr>
<td>Matrix × Matrix</td>
<td>54</td>
<td>47</td>
<td>46</td>
</tr>
</tbody>
</table>

- Matrix × Vector is not direct available in SMatrix (ROOT v5.34.04)
Further studies: speed comparison measurements

- Evaluated in a small test framework:
  - CPU time of different matrix multiplications
  - comparison studies between: CLHEP, Eigen, SMatrix, MKL and hand written C++ operation
    (non vectorized: BasMult & vectorized: OptMult)
- Compiler setup: gcc 4.7.2 and '-O3' for vectorization
- Implemented matrix multiplications:
  - $4 \times 4$ with square matrices
  - rectangular matrices: $A_{5\times3} \times B_{3\times5}$
  - template expression: $C_{5\times5} = \alpha A_{5}B_{3\times5} + \beta C_{5\times5}$
Speed comparison with $4 \times 4$ square matrices

**Speedup factor w.r.t. CLHEP:**

- Hand vectorized operation is the fastest
Speed comparison with rectangular matrices: $A_{5\times3} \times B_{3\times5}$

Speedup factor w.r.t. CLHEP:

- Without vectorization
Speed comparison with expression templates:

\[ C_{5 \times 5} = \alpha A_{5 \times 5} B_{3 \times 5} + \beta C_{5 \times 5} \]

**Speedup factor w.r.t. CLHEP:**

- Without vectorization

![Graph showing speedup factors for different libraries besides CLHEP]
Conclusion of speed comparison studies: matrix operations

- Hand vectorized operation is the fastest, but hand written code is less maintainable and error prone.
- Eigen is the fastest linear algebra library.
- ATLAS decided to replace CLHEP with Eigen for linear algebra operation for track reconstruction.
Trigonometric functions

- GNU libm used as default for trigonometric functions in ATLAS software
- Monitored calls and instructions with Pin during reconstruction job with 2012 data sample

Results with Pin and test framework:

<table>
<thead>
<tr>
<th>Function</th>
<th>M Call/Evt</th>
<th>Time/Calls [ns]</th>
<th>Time/Evt [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp</td>
<td>3.4</td>
<td>146</td>
<td>0.496</td>
</tr>
<tr>
<td>cos</td>
<td>2.5</td>
<td>149</td>
<td>0.373</td>
</tr>
<tr>
<td>sin</td>
<td>2.2</td>
<td>149</td>
<td>0.328</td>
</tr>
<tr>
<td>atanf</td>
<td>2.1</td>
<td>22</td>
<td>0.0462</td>
</tr>
<tr>
<td>sincosf</td>
<td>2.1</td>
<td>24</td>
<td>0.050</td>
</tr>
</tbody>
</table>

- Total times of all trigonometric functions per event: 2.037 s of 14.41 s
CPU time comparison study with alternative math libraries

- **VDT**
  - developed by CMS
  - inlined functions, designed for auto-vectorization with fast approximative calculation
  - used with different API: `fast_func(arg)` or as 'drop in' replacement with `LD_PRELOAD` (used in this studies; which disables inlining)
  - further detailed information in Danilo Piparo’s talk

- **libimf**
  - performance optimized library by Intel (Version 2013)
  - can be used as 'drop in' replacement with `LD_PRELOAD`
    (use multiple code path for SSE and AVX instructions)

- CPU time comparison study: running reconstruction job with 2012 data sample with GNU libm, VDT and libimf.
Results of CPU time comparison study:

<table>
<thead>
<tr>
<th>Math library</th>
<th>Relative to GNU libm</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU libm</td>
<td>1.000</td>
</tr>
<tr>
<td>VDT</td>
<td>0.923</td>
</tr>
<tr>
<td>libimf</td>
<td>0.919</td>
</tr>
</tbody>
</table>

Conclusion:
- libimf provides the fastest trigonometric functions
- ATLAS decided to replace GNU libm with libimf, but keep VDT available in ATLAS software
PAPI is an analysis API for hardware performance counters

Pin provides detailed information about how ATLAS software uses CLHEP and trigonometric functions

Comparison studies showed:
- Eigen is the fastest library for matrix and vector operations
- libmf is the fastest library for trigonometric functions

ATLAS decided to replace:
- CLHEP with Eigen for linear algebra operations for track reconstruction
- GNU libm with libimf for trigonometric functions

Detail information about upgrades in tracking algorithms:

Talk by Robert Langenberg on Tuesday
Backup slides
- CPU time breakdown per domain depending on the day after the release build during the night
- Measure while processing a data sample from 2011
- CPU time breakdown per domain depending on the day after the release build during the night
- Measure while processing a data sample from 2011
Results with PAPI of matrix/vector operations

- Monitor in a simple test framework floating point operation of several matrix/vector calls with PAPI
- Compare CLHEP with other classes: Eigen and SMatrix

Floating operations of 4-dimensional vector/matrix

<table>
<thead>
<tr>
<th>Operations</th>
<th>CLHEP</th>
<th>Eigen</th>
<th>SMatrix/SVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix allocation</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Vector allocation</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Vector + Vector</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Matrix × Vector</td>
<td>32</td>
<td>15</td>
<td>-</td>
</tr>
<tr>
<td>Matrix × Matrix</td>
<td>128</td>
<td>58</td>
<td>112</td>
</tr>
</tbody>
</table>

- Matrix × Vector is not available in SMatrix
Summary and Conclusion

Speed comparison with $4 \times 4$ square matrices

- Additionally: setup of matrix multiplication with 'std::vectors'
  - basic setup (not vectorized)
  - optimized setup: vectorized without horizontal sums

Basic Multiplication (BasMult):

```c++
for(int i = 0; i < 16; i+=4){
    for(int j = 0; j < 4; j++){
        z[i+j] = x[i] * y[j] + x[i+1] * y[4 + j] \
                 + x[i+2] * y[8 + j] + x[i+3] * y[12 + j];
    }
}
```

Optimized Multiplication (OptMult):

```c++
for(int i = 0; i < 16; i+=4){
    Vec4d r1 = Vec4d(x[i]) * Vec4d(y);
    for(int j = 1; j < 4; j++){
        r1 += Vec4d(x[i+j]) * Vec4d(&y[j*4]);
    }
    r1.store(&z[i]);
}
```
CLHEP

- CLHEP - A Class Library for High Energy Physics
- http://proj-clhep.web.cern.ch/proj-clhep/
- A set of HEP-specific utility classes such as random generators, physics vectors, geometry and linear algebra
- CLHEP provides a generic interface for any-dimension matrix/vector

Problem:
- Not maintained anymore
- Not well performed (especially matrix operations)
Eigen

- [ ] http://eigen.tuxfamily.org/
- [ ] Pure C++ template library
  - [ ] header only $\rightarrow$ no binary to compile/install
  - [ ] Opensource: MPL2
- [ ] It supports:
  - [ ] all matrix sizes
  - [ ] SIMD vectorization
  - [ ] compilers (gcc, icc, clang, ...)
- [ ] It is optimized for
  - [ ] small fixed-size matrices
  - [ ] arbitrarily large dynamic size matrices
SMatrix

- ROOT C++ package for high performance vector and matrix computations
- Implemented as expression templates
- Provide matrix/vector classes of arbitrary dimensions and type
- Classes are templated on the dimension of the matrix/vector and on the scalar type
- Problem:
  - Supports only symmetric matrices
  - Not complete linear algebra package such as Intel MKL or Eigen
Intel Math Kernel Library (MKL)

- Includes:
  - Basic Linear Algebra Subprograms (BLAS)
  - LAPACK routines for solving systems of linear equations
- Optimized:
  - on modern Intel processors
  - for large matrices and BLAS operations: $C = \alpha AB + \beta C$