Optimizing the ATLAS code with different profilers

Sami Kama$^1$
on behalf of the ATLAS Collaboration$^2$

$^1$Southern Methodist University, Texas

$^2$https://cdsweb.cern.ch/record/1386334

Advanced Computing and Analysis Techniques in Physics, Beijing/China, May 2013
Large Hadron Collider

- Located at CERN near Geneva
- 27 km circumference and \( \sim 100 \text{ m} \) below surface
- It is operational since 2010.
- There are 4 detectors located on it, ATLAS is one of two large general purpose detectors
- It is shutdown for two years for upgrades and maintenance on March 2013
- It will operate at higher beam energy and higher luminosity after shutdown
ATLAS

- ATLAS is composed of different co-centric cylindrical detectors of \( \sim 150 \text{ M} \) readout channels.
- It has a three-level trigger system:
  - Level-1 trigger is hardware based and located in the detector. It reduces 40 MHz input rate to 75 kHz.
  - Level 2 and Level 3 are software based triggers running on \( \sim 16k \) core pc farm, reducing final event rate to 300 Hz at \( \sim 1.6 \text{ MB/ev} \).
  - Trigger will be upgraded to 1 kHz output in 2015.
- Selected events are stored and offline processed in more detail.
- Both offline processing and online selection is done with the same software with a different configuration.
- So far ATLAS stored and processed \( \sim 22 \text{ PB} \) of raw data.
- With the increase in LHC energy, collision rate, event complexity and trigger output ATLAS software needs to speedup up to 3 times.
ATLAS Software (ATHENA)

- Composed of more than 6 million lines of C++ and Python code with a small amount of FORTRAN code
- Spread over $\sim 2000$ packages
- Producing 4k+ libraries of various sizes
- Evolving since more than 10 years
- Written by people with various levels of programming knowledge, some experts, some first timers
- Detailed knowledge in packages is frequently lost due to authors changing topics, institutes or leaving the field.
- Configuration is done in Python
- 64-bit application consumes $\sim 4$ GB memory
  - big challenge for many profilers
ATLAS Software (ATHENA)

- Composed of more than 6 million lines of C++ and Python code with a small amount of FORTRAN code
- Spread over \( \sim 2000 \) packages
- Producing 4k+ libraries of various sizes
- Evolving since more than 10 years
- Written by people with various levels of programming knowledge, some experts, some first timers
- Detailed knowledge in packages is frequently lost due to authors changing topics, institutes or leaving the field.
- Configuration is done in Python
- 64-bit application consumes \( \sim 4 \) GB memory
  - big challenge for many profilers

Need tools to point out problematic code!
Profilers commonly used in ATLAS

ATLAS uses various tools to profile and monitor ATHENA

- PerfMon to collect coarse level resource utilization information from ATHENA instrumentation.
- Valgrind suite to check leaks, extract callgraphs and detailed CPU utilization
- GOODA to investigate most detailed CPU utilization
- Pin Tools to do detailed code instrumentation to study parameter ranges
- Other tools such as Intel Vtune, Papi, igprof from CMS, Google perf tools etc.
Google Data Analyzer (GOODA)

- Open source, developed by a collaboration between ATLAS and Google
- Uses Linux perf tool to configure and collect detailed performance monitoring unit (PMU) information from hardware monitoring units inside CPUs
- Analyzes the monitoring data and creates spreadsheets that can be displayed in web browsers.
- Gives detailed information about performance bottlenecks
Profiling

GOODA Example

Sami Kama (Southern Methodist University) Optimizing the ATLAS code with profilers

ACAT, May 2013
### Profiling

**GOODA Example**

#### Processes

<table>
<thead>
<tr>
<th>Process Path</th>
<th>Cycles</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>athena.py</td>
<td>2625392</td>
<td>0.71%</td>
</tr>
<tr>
<td>delinux</td>
<td>315037</td>
<td>0.85%</td>
</tr>
</tbody>
</table>

**Generic Optimization Data Analyzer**

<table>
<thead>
<tr>
<th>Function</th>
<th>Process</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>2625392</td>
<td>0.71%</td>
</tr>
<tr>
<td>module</td>
<td>315037</td>
<td>0.85%</td>
</tr>
</tbody>
</table>

**Optimizing the ATLAS code with profilers**

Sami Kama (Southern Methodist University)
### GOODA Example

#### Hotspot functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Offset</th>
<th>Length</th>
<th>Module Path</th>
<th>Process Path</th>
<th>Cycles</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trk::RangeKuttaPropagato...</td>
<td>0x10e6</td>
<td>0x1051</td>
<td>Trk::RangeKuttaPropagato...</td>
<td>athena.py</td>
<td>0x3430</td>
<td>0x2904</td>
</tr>
<tr>
<td>Operator new(unsig...</td>
<td>0x10a8</td>
<td>0x2da</td>
<td>Operator new(unsig...</td>
<td>athena.py</td>
<td>0x3417</td>
<td>0x2904</td>
</tr>
<tr>
<td>Mr::SpacePointsSeed...</td>
<td>0x2709</td>
<td>0x274f</td>
<td>Mr::SpacePointsSeed...</td>
<td>athena.py</td>
<td>0x3417</td>
<td>0x2904</td>
</tr>
<tr>
<td>Mr::Tetrahedron ...</td>
<td>0x21ab</td>
<td>0x2bf</td>
<td>Mr::Tetrahedron ...</td>
<td>athena.py</td>
<td>0x3417</td>
<td>0x2904</td>
</tr>
</tbody>
</table>

Optimizing the ATLAS code with profilers

ACAT, May 2013
## GOODA Example

### Profiling

**GOODA Example**

**PMU events, grouped**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Offset</th>
<th>Length</th>
<th>Module Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trk:RungeKuttaPropagator</td>
<td>0x2f40</td>
<td>0x177</td>
<td>athena.py</td>
</tr>
<tr>
<td>operator new(unsigned long)</td>
<td>0x1840</td>
<td>0x3d</td>
<td>athena.py</td>
</tr>
<tr>
<td>operator delete(void*)</td>
<td>0x12c0</td>
<td>0x2d</td>
<td>athena.py</td>
</tr>
<tr>
<td>Indet::ISpacePointsSeedInit</td>
<td>0x2f70</td>
<td>0x7e</td>
<td>athena.py</td>
</tr>
<tr>
<td>Indet::TFT_TrajectoryCell</td>
<td>0x8c0</td>
<td>0x6f</td>
<td>athena.py</td>
</tr>
<tr>
<td>IsolInterno</td>
<td>0x1560</td>
<td>0x40</td>
<td>athena.py</td>
</tr>
<tr>
<td><em>dynamic_cast&lt;CAxonI</em> 3</td>
<td>0x200</td>
<td>0x77</td>
<td>athena.py</td>
</tr>
<tr>
<td>Trk::MagneticFieldMapSole</td>
<td>0x1840</td>
<td>0x50</td>
<td>athena.py</td>
</tr>
<tr>
<td>master &amp; gromacs</td>
<td>0x1c0</td>
<td>0x40</td>
<td>athena.py</td>
</tr>
<tr>
<td>Trk::RungeKuttaPropagator</td>
<td>0x2770</td>
<td>0x55</td>
<td>athena.py</td>
</tr>
<tr>
<td>Indet::TFT_TrajectoryCell</td>
<td>0x6420</td>
<td>0x50</td>
<td>athena.py</td>
</tr>
<tr>
<td>Trk::PatternTrackParameter</td>
<td>0x3280</td>
<td>0x80</td>
<td>athena.py</td>
</tr>
<tr>
<td>Trk::RungeKuttaUtils::tra</td>
<td>0x17f9</td>
<td>0x22</td>
<td>athena.py</td>
</tr>
<tr>
<td>Trk::RungeKuttaPropagator</td>
<td>0x2610</td>
<td>0x37</td>
<td>athena.py</td>
</tr>
<tr>
<td>softmax::softmax</td>
<td>0x2800</td>
<td>0x50</td>
<td>athena.py</td>
</tr>
<tr>
<td>deflate:: deflate</td>
<td>0x659</td>
<td>0x76</td>
<td>deflate.c</td>
</tr>
</tbody>
</table>

### Optimizing the ATLAS code with profilers

**ACAT, May 2013 7 / 18**

Sami Kama  (Southern Methodist University)
Hottest functions

- Tracking code, particularly Runge-Kutta methods
  - suffering from instruction starvation
  - mostly composed of vector and matrix operations
  - Vectorization helps, up to 2.5x speedup from manual vectorization in certain points is achieved, need vectorized vector libs

- Memory allocation and de-allocation
  - too many new() and deletes
  - Event Data Model (EDM) change is underway

- Magnetic field code
  - suffering from load latency and instruction latency
  - was written in fortran, several calls deep in fortran
  - re-written in C++, already was about 2x faster than fortran implementation
  - C++ code profiled again to optimize further
Improving new Magnetic Field

Profiled a special test code that queries the magnetic field code randomly.
Improving new Magnetic Field

Profiled a special test code that queries the magnetic field code randomly.

Expanding instruction latency

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Cycles</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagField INSGFieldSrc</td>
<td>103586651 (100%)</td>
<td>7592330 (73%)</td>
</tr>
<tr>
<td>MagField: AtlasFieldSrc</td>
<td>5217442 (100%)</td>
<td>3454370 (66%)</td>
</tr>
<tr>
<td>MagField: MagFieldTestb</td>
<td>2311073 (100%)</td>
<td>1118041 (51%)</td>
</tr>
</tbody>
</table>

Sami Kama (Southern Methodist University) Optimizing the ATLAS code with profilers

ACAT, May 2013
Improving new Magnetic Field

Profiled a special test code that queries the magnetic field code randomly.

High instruction latency originating from division operations. Can drill down (double-click) for details.
Magnetic field code details

- Detailed view contains both disassembly and source code if debug symbols and source files are available.
- Events are displayed at instruction level and hottest basic block is automatically highlighted.
- Debug symbols in optimized builds are skewed, instruction latency is coming from another file.
**Magnetic field code details**

- Detailed view contains both disassembly and source code if debug symbols and source files are available.
- Events are displayed at instruction level and hottest basic block is automatically highlighted.
- Debug symbols in optimized builds are skewed, instruction latency is coming from another file.
Magnetic field code details

- Detailed view contains both disassembly and source code if debug symbols and source files are available.
- Events are displayed at instruction level and hottest basic block is automatically highlighted.
- Debug symbols in optimized builds are skewed, instruction latency is coming from another file.
Fixing the problem

```c
float dBdz[3], dBdr[3], dBdphi[3];
for ( int j = 0; j < 3; j++ ) { // Bz, Br, Bphi components
    dBdz[j] = sz*( gr*( gphi*(m_field[4][j]-m_field[0][j]) +
                   fphi*(m_field[5][j]-m_field[1][j]) ) +
                   fr*( gphi*(m_field[6][j]-m_field[2][j]) +
                        fphi*(m_field[7][j]-m_field[3][j]) ) );
    dBdr[j] = sr*( gz*( gphi*(m_field[2][j]-m_field[0][j]) +
                     fphi*(m_field[3][j]-m_field[1][j]) ) +
                     fz*( gphi*(m_field[6][j]-m_field[4][j]) +
                          fphi*(m_field[7][j]-m_field[5][j]) ) );
    dBdphi[j] = sphi*( gz*( gr*(m_field[1][j]-m_field[0][j]) +
                            fr*(m_field[3][j]-m_field[2][j]) ) +
                            fz*( gr*(m_field[5][j]-m_field[4][j]) +
                                 fr*(m_field[7][j]-m_field[6][j]) ) );
}
// convert to cartesian coordinates
float cc = c*c;
float cs = c*s;
float ss = s*s;
deriv[6] = c*dBdr[0] - s*dBdphi[0]/r;
deriv[7] = s*dBdr[0] + c*dBdphi[0]/r;
deriv[8] = dBdz[0];
```
Fixing the problem

```c
float dBdz[3], dBdr[3], dBdphi[3];
for ( int j = 0; j < 3; j++ ) { // Bz, Br, Bphi components
    dBdz[j] = sz*( gr*( gphi*(m_field[4][j]-m_field[0][j]) +
                      fphi*(m_field[5][j]-m_field[1][j]) ) +
                      fr*( gphi*(m_field[6][j]-m_field[2][j]) +
                      fphi*(m_field[7][j]-m_field[3][j]) ) )
    dBdr[j] = sr*( gz*( gphi*(m_field[2][j]-m_field[0][j]) +
                      fphi*(m_field[3][j]-m_field[1][j]) ) +
                      fz*( gphi*(m_field[6][j]-m_field[4][j]) +
                      fphi*(m_field[7][j]-m_field[5][j]) ) )
    dBdphi[j] = sphi*( gz*( gr*(m_field[1][j]-m_field[0][j]) +
                         fr*(m_field[3][j]-m_field[2][j]) ) +
                      fz*( gr*(m_field[5][j]-m_field[4][j]) +
                         fr*(m_field[7][j]-m_field[6][j]) ) )
}
// convert to cartesian coordinates
float cc = c*c;
float cs = c*s;
float ss = s*s;
derv[6] = c*dBdr[0] - s*dBdphi[0]/r;
derv[7] = s*dBdr[1] + c*dBdphi[0]/r;
derv[8] = dBdz[0];
```

Lots of $1/r$, replace with inverse multiplication.
Fixing the problem

```c
float dBdz[3], dBdr[3], dBdphi[3];
for ( int j = 0; j < 3; j++ ) { // Bz, Br, Bphi components
    dBdz[j] = sz*( gr*( gphi*(m_field[4][j]-m_field[0][j]) +
                     fphi*(m_field[5][j]-m_field[1][j]) ) +
                 fr*( gphi*(m_field[6][j]-m_field[2][j]) +
                     fphi*(m_field[7][j]-m_field[3][j]) ) );
    dBdr[j] = sr*( gz*( gphi*(m_field[2][j]-m_field[0][j]) +
                     fphi*(m_field[3][j]-m_field[1][j]) ) +
                 fz*( gphi*(m_field[6][j]-m_field[4][j]) +
                     fphi*(m_field[7][j]-m_field[5][j]) ) );
    dBdphi[j] = sph*( gz*( gr*(m_field[1][j]-m_field[0][j]) +
                         fr*(m_field[3][j]-m_field[2][j]) ) +
               fz*( gr*(m_field[5][j]-m_field[4][j]) +
                     fr*(m_field[7][j]-m_field[6][j]) ) );
}

// convert to cartesian coordinates
float cc = c*c;
float cs = c*s;
float ss = s*s;

deriv[6] = c*dBdr[0] - s*dBdphi[0]/r;
deriv[7] = s*dBdr[0] + c*dBdphi[0]/r;
deriv[8] = dBdz[0];
```

Dot products of two vectors!

Lots of $1/r$, replace with inverse multiplication.
Fixing the problem

```c
float dBdz[3], dBdr[3], dBdphi[3];
for ( int j = 0; j < 3; j++ ) { // Bz, Br, Bphi components
    dBdz[j] = sz*( gr*( gphi*(m_field[4][j]-m_field[0][j]) +
                    fphi*(m_field[5][j]-m_field[1][j]) ) +
                    fr*( gphi*(m_field[6][j]-m_field[2][j]) +
                    fphi*(m_field[7][j]-m_field[3][j]) ) );
    dBdr[j] = sr*( gz*( gphi*(m_field[2][j]-m_field[0][j]) +
                     fphi*(m_field[3][j]-m_field[1][j]) ) +
                     fz*( gphi*(m_field[6][j]-m_field[4][j]) +
                     fphi*(m_field[7][j]-m_field[5][j]) ) );
    dBdphi[j] = sphi*( gz*( gr*(m_field[1][j]-m_field[0][j]) +
                      fr*(m_field[3][j]-m_field[2][j]) ) +
                      fz*( gr*(m_field[5][j]-m_field[4][j]) +
                      fr*(m_field[7][j]-m_field[6][j]) ) );
}

// convert to cartesian coordinates
float cc = c*c;
float cs = c*s;
float ss = s*s;

deriv[6] = c*dBdr[0] - s*dBdphi[0]/r;
deriv[7] = s*dBdr[1] + c*dBdphi[0]/r;
deriv[8] = dBdz[0];
```

DOT products of two vectors!

Lots of $1/r$, replace with inverse multiplication

40% more speedup after replacement, 5% – 20% global speedup with new code. Vectorization is yet to come.
Pin Tools

Pin is a dynamic binary instrumentation framework from Intel

- instrumentation is done on binary at run-time, eliminates need to modify or recompile the code
- can instrument from instruction level to function level, supports dynamically generated code
- can access function parameters and register contents
- can work with threaded programs
- has limited access to symbol and debug information
- creates a copy of the binary, inspects applications instructions and inserts calls to analysis functions
- used in computer architecture, security, emulation and parallel program analysis tools such as Intel’s Parallel Inspector, Parallel Amplifier, Trace Analyzer and Collector, CMP$im and many others
Improving Tracking Code

- Tracking is mostly based on vector and matrix operations
- CLHEP library is used for matrix and vector representations and operations
  - CLHEP is not performance optimized and does not vectorize well.
  - It is very hard to figure out dimensions and operations from the code
- Another, vectorized vector math library is required
  - There are many libraries, which is the best?
- Pin is used to instrument CLHEP classes and operations to extract information on most commonly used objects
<table>
<thead>
<tr>
<th>Calls</th>
<th>Instr</th>
<th>&lt;instr&gt;/call</th>
<th>Call rank</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1778523</td>
<td>6392431813</td>
<td>3594.24</td>
<td>1439</td>
<td>CLHEP::operator*( CLHEP::HepMatrix const&amp;, CLHEP::HepSymMatrix const&amp;)</td>
</tr>
<tr>
<td>671676353</td>
<td>5988139520</td>
<td>8.92</td>
<td>9</td>
<td>CLHEP::Hep2Vector::operator()(int) const</td>
</tr>
<tr>
<td>232093102</td>
<td>5956556656</td>
<td>25.66</td>
<td>27</td>
<td>HepGeom::Transform3D::operator()(int, int) const</td>
</tr>
<tr>
<td>285282108</td>
<td>3709057782</td>
<td>13.00</td>
<td>21</td>
<td>CLHEP::Hep3Vector::operator()(int)</td>
</tr>
<tr>
<td>15815930</td>
<td>3179001930</td>
<td>201.00</td>
<td>319</td>
<td>CLHEP::HepRotation::rotateAxes( CLHEP::Hep3Vector const&amp;, CLHEP::Hep3Vector const&amp;, CLHEP::Hep3Vector const&amp;)</td>
</tr>
<tr>
<td>20529818</td>
<td>2422518524</td>
<td>118.00</td>
<td>267</td>
<td>HepGeom::Transform3D::inverse() const</td>
</tr>
<tr>
<td>31612743</td>
<td>2212258670</td>
<td>69.98</td>
<td>200</td>
<td>CLHEP::HepSymMatrix::HepSymMatrix( CLHEP::HepSymMatrix const&amp;)</td>
</tr>
<tr>
<td>28914115</td>
<td>1929106393</td>
<td>66.72</td>
<td>214</td>
<td>CLHEP::HepVector::HepVector(int, int)</td>
</tr>
<tr>
<td>51974716</td>
<td>1819115060</td>
<td>35.00</td>
<td>150</td>
<td>HepGeom::operator*( HepGeom::Transform3D const&amp;, HepGeom::Point3D&lt; double &gt; const&amp;)</td>
</tr>
<tr>
<td>27652274</td>
<td>1506352669</td>
<td>54.47</td>
<td>219</td>
<td>CLHEP::HepVector::HepVector( CLHEP::HepVector const&amp;)</td>
</tr>
</tbody>
</table>

In order to determine a suitable replacement, these routines are instrumented with Pin and function properties are queried.
CLHEP Instrumentation

- Each hot function is instrumented by its respective analysis routine
- These functions analyzed the call parameters for hottest functions for each call and produced output to further offline processing

```c
void InstFunc(ADDRINT addr, std::string& msg, par1 v1, par2 v2){
    //do stuff
} //analysis code

VOID InstHOOK(RTN rtn, VOID *v){//called for each routine
    RTN_Open(rtn);//read the routine from binary
    std::string *msg=new std::string;//extra param for analysis func
    if (RTN_Name(rtn).compare("<mangledName>") == 0) {
        RTN_InsertCall(rtn, IPOINT_BEFORE, (AFUNPTR)InstFunc,
                        IARG_RETURN_IP,
                        IARG_PTR, msg,
                        IARG_FUNCARG_ENTRYPOINT_VALUE, 1,//func param1
                        IARG_FUNCARG_ENTRYPOINT_VALUE, 2,//func param2
                        IARG_END);//instrument <mangledName> function
    }
}
```
Using pin results

- From pin instrumentation we observed frequent use of 3x5, 5x3 and 5x5 matrices.
- This information is used for testing different vector math libraries.
- 4x4 test is added to estimate performance in square matrices.
- Eigen performed best and is currently being implemented.
Using pin results

- From pin instrumentation we observed frequent use of 3x5, 5x3 and 5x5 matrices.
- This information is used for testing different vector math libraries.
- 4x4 test is added to estimate performance in square matrices.
- Eigen performed best and is currently being implemented.

Comparison of Math libraries

Eigen is obvious choice
Summary

- ATLAS has successfully identified points to improve in its huge codebase using profilers such as GOODA and Pin tools.
- GOODA is an open source, performance profiling tool giving valuable insight about bottlenecks in the program.
- Pin is very good at finding out the details of actually executed code. It makes analysis of large code bases very easy.
- Information gained from Pin enabled us to choose optimal vector library.
- Some results are already implemented and improved performance up to 20%.
- Studies are ongoing, many more improvements to come.
Thank you for your attention

Thanks to

- Graeme A. Stewart
- Rolf Seuster
- Roberto A Vitillo