Optimizing the ATLAS code with different profilers

Sami Kama\textsuperscript{1}

on behalf of the ATLAS Collaboration\textsuperscript{2}

\textsuperscript{1}Southern Methodist University, Texas

\textsuperscript{2}https://cdsweb.cern.ch/record/1386334

Advanced Computing and Analysis Techniques in Physics, Beijing/China, May 2013
Introduction

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(20) 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(600) Hz at \( \sim 1.6 \text{ MB/ev} \)
  - Trigger will be upgraded to 1 kHz output in 2015
- Selected events are stored and processed offline in more detail.
- Both offline processing and online selection is done with the same software using 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 speed up considerably
Comprised of more than 6 million lines of C++ and Python code with a small amount of FORTRAN code

Spread over ~2000 packages

Producing 4k+ libraries of various sizes

Evolving for more than 10 years

Written by people with various levels of programming knowledge, some experts, some first timers

Detailed knowledge of packages is frequently lost due to authors changing topics, institutes or leaving the field.

Configuration is done in Python

64-bit application consumes ~4 GB memory
  - big challenge for many profilers
ATLAS Software (ATHENA)

- Comprised 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 for more than 10 years
- Written by people with various levels of programming knowledge, some experts, some first timers
- Detailed knowledge of 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 call graphs 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

Optimizing the ATLAS code with profilers

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

#### Hotspot functions

- **Trk::RungeKuttaPropagator**: 0x156e8
- **operator new(ulonglong)**: 0x1c0a0
- **Int::SpacePointsSeed**: 0x16f70
- **Int::TRT_trajectoryTime**: 0x16e0c
- **Int::Dynamic_calo**: 0x184b8
- **Trk::MagneticFieldMapSole**: 0x1a648
- **Object::genstage**: 0x1a1e8
- **Trk::RungeKutta**: 0x27b0a
- **Trk::PatternTrackParameters**: 0x2d22b
- **Trk::RungeKuttaUtil::tra**: 0x1d7c9
- **sofitters**: 0x3f2c8
- **define_event**: 0x3f569
- **TrainedNetwork::calculation**: 0x3f528
- **cXnablal::__vm_class_t**: 0x1a378
### PMU events, grouped

**Generic Optimization Data Analyzer (GODA)**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Offset</th>
<th>Length</th>
<th>Module</th>
<th>Process</th>
<th>Unhalted Core Cycles</th>
<th>Unhalted Core Cycles</th>
<th>Unhalted Core Cycles</th>
<th>Unhalted Core Cycles</th>
<th>Unhalted Core Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trk::RungeKuttaPropagator</td>
<td>0x235e0</td>
<td>0x1051</td>
<td>libTrk::RungeKuttaPropagator</td>
<td>athena.py</td>
<td>2628392 (100%)</td>
<td>2628392 (100%)</td>
<td>2628392 (100%)</td>
<td>2628392 (100%)</td>
<td>2628392 (100%)</td>
</tr>
<tr>
<td>operator new(unsigned lon)</td>
<td>0x13060</td>
<td>0x320</td>
<td>libTrk::allocateMinimal.so</td>
<td>athena.py</td>
<td>6520151 (100%)</td>
<td>6520151 (100%)</td>
<td>6520151 (100%)</td>
<td>6520151 (100%)</td>
<td>6520151 (100%)</td>
</tr>
<tr>
<td>operator delete(void*)</td>
<td>0x12c10</td>
<td>0x2da</td>
<td>libTrk::minimal.so</td>
<td>athena.py</td>
<td>478002 (100%)</td>
<td>478002 (100%)</td>
<td>478002 (100%)</td>
<td>478002 (100%)</td>
<td>478002 (100%)</td>
</tr>
<tr>
<td>InDet::SpacePointsSeed::</td>
<td>0x207e0</td>
<td>0x374</td>
<td>libSpacePointsSeedTool.so</td>
<td>athena.py</td>
<td>622000 (100%)</td>
<td>622000 (100%)</td>
<td>622000 (100%)</td>
<td>622000 (100%)</td>
<td>622000 (100%)</td>
</tr>
<tr>
<td>InDet::TRT_Trajectory::</td>
<td>0x2c180</td>
<td>0x209</td>
<td>libTRT::TrackExtensionTool.so</td>
<td>athena.py</td>
<td>551277 (100%)</td>
<td>551277 (100%)</td>
<td>551277 (100%)</td>
<td>551277 (100%)</td>
<td>551277 (100%)</td>
</tr>
<tr>
<td>IsolInterm</td>
<td>0x126f0</td>
<td>0x440</td>
<td>libISF::core.so</td>
<td>athena.py</td>
<td>207540 (100%)</td>
<td>207540 (100%)</td>
<td>207540 (100%)</td>
<td>207540 (100%)</td>
<td>207540 (100%)</td>
</tr>
<tr>
<td><em>dynamic_catch</em></td>
<td>0x2f200</td>
<td>0x11f</td>
<td>libISF::core.so</td>
<td>athena.py</td>
<td>307972 (100%)</td>
<td>307972 (100%)</td>
<td>307972 (100%)</td>
<td>307972 (100%)</td>
<td>307972 (100%)</td>
</tr>
<tr>
<td>Trk::MagneticFieldMap::</td>
<td>0x29400</td>
<td>0x1c5</td>
<td>libTrk::MagFieldUtils.so</td>
<td>athena.py</td>
<td>449764 (100%)</td>
<td>449764 (100%)</td>
<td>449764 (100%)</td>
<td>449764 (100%)</td>
<td>449764 (100%)</td>
</tr>
<tr>
<td>master 0::manage</td>
<td>0x1060</td>
<td>0x4b0</td>
<td>libISF::Fields.so</td>
<td>athena.py</td>
<td>369184 (100%)</td>
<td>369184 (100%)</td>
<td>369184 (100%)</td>
<td>369184 (100%)</td>
<td>369184 (100%)</td>
</tr>
<tr>
<td>Trk::RungeKuttaPropagator</td>
<td>0x27790</td>
<td>0x2ff</td>
<td>libTrk::RungeKuttaPropagator</td>
<td>athena.py</td>
<td>103543 (100%)</td>
<td>103543 (100%)</td>
<td>103543 (100%)</td>
<td>103543 (100%)</td>
<td>103543 (100%)</td>
</tr>
<tr>
<td>InDet::TRT_Trajectory::</td>
<td>0x24200</td>
<td>0x209</td>
<td>libTRT::TrackExtensionTool.so</td>
<td>athena.py</td>
<td>496342 (100%)</td>
<td>496342 (100%)</td>
<td>496342 (100%)</td>
<td>496342 (100%)</td>
<td>496342 (100%)</td>
</tr>
<tr>
<td>Trk::PatternTrack::</td>
<td>0x23220</td>
<td>0x203</td>
<td>libTrk::Pattern::</td>
<td>athena.py</td>
<td>273135 (100%)</td>
<td>273135 (100%)</td>
<td>273135 (100%)</td>
<td>273135 (100%)</td>
<td>273135 (100%)</td>
</tr>
<tr>
<td>Trk::RungeKutta::</td>
<td>0x270f0</td>
<td>0x522</td>
<td>libTrk::RungeKutta::</td>
<td>athena.py</td>
<td>249522 (100%)</td>
<td>249522 (100%)</td>
<td>249522 (100%)</td>
<td>249522 (100%)</td>
<td>249522 (100%)</td>
</tr>
<tr>
<td>soLexToSS::</td>
<td>0x2a2e0</td>
<td>0x300</td>
<td>libISF::Core.so</td>
<td>athena.py</td>
<td>224126 (100%)</td>
<td>224126 (100%)</td>
<td>224126 (100%)</td>
<td>224126 (100%)</td>
<td>224126 (100%)</td>
</tr>
<tr>
<td>deflatter::</td>
<td>0x2659</td>
<td>0x20e</td>
<td>libISF::so_1.2::</td>
<td>athena.py</td>
<td>266747 (100%)</td>
<td>266747 (100%)</td>
<td>266747 (100%)</td>
<td>266747 (100%)</td>
<td>266747 (100%)</td>
</tr>
<tr>
<td>TtrainNetwork::</td>
<td>0x21520</td>
<td>0x4d4</td>
<td>libTrk::Neural::</td>
<td>athena.py</td>
<td>174563 (100%)</td>
<td>174563 (100%)</td>
<td>174563 (100%)</td>
<td>174563 (100%)</td>
<td>174563 (100%)</td>
</tr>
<tr>
<td>cxxabvt::</td>
<td>0x2e70</td>
<td>0x320</td>
<td>libISF::Core.so</td>
<td>athena.py</td>
<td>117906 (100%)</td>
<td>117906 (100%)</td>
<td>117906 (100%)</td>
<td>117906 (100%)</td>
<td>117906 (100%)</td>
</tr>
</tbody>
</table>
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 math 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 code, several calls deep
  - 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

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Samples</th>
<th>unhalted_core_cycles</th>
<th>unhalted_core_recloth</th>
<th>instruction_retired</th>
<th>instruction_retired_any</th>
<th>load_likely</th>
<th>load_likely_any</th>
<th>instruction_stall</th>
<th>instruction_stall_all</th>
<th>branch_disambiguation</th>
<th>branch_disambiguation_any</th>
<th>store_retired</th>
<th>exception_handling</th>
<th>unhalted_reference_cycles</th>
<th>unhalted_exception_stall</th>
<th>unretired_instructions</th>
<th>unretired_instructions_any</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>789012</td>
<td>345678</td>
<td>987654</td>
<td>123456</td>
<td>789012</td>
<td>345678</td>
<td>987654</td>
<td>123456</td>
<td>789012</td>
<td>345678</td>
<td>987654</td>
<td>123456</td>
<td>789012</td>
<td>345678</td>
<td>987654</td>
<td>123456</td>
<td>789012</td>
</tr>
</tbody>
</table>
Improving new Magnetic Field

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

Expanding instruction latency
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.
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]) +
                   fr*( gphi*(m_field[6][j]-m_field[2][j]) +
                      fphi*(m_field[7][j]-m_field[3][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]) ) +
                     fphi*(m_field[3][j]-m_field[1][j]) ) +
                     fphi*(m_field[7][j]-m_field[3][j]) );
    dBdphi[j] = sphi*( gz*( gr*(m_field[1][j]-m_field[0][j]) +
                           fr*(m_field[3][j]-m_field[2][j]) ) +
                           fr*(m_field[7][j]-m_field[5][j]) ) +
                           fr*(m_field[7][j]-m_field[5][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];
```
float dDz[3], dBdr[3], dBdphi[3];
for (int j = 0; j < 3; j++) { // Bz, Br, Bphi components
    dDz[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] = dDz[0];

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

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

Dot products of two vectors!
Fixing the problem

```
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]) ) ) ;
}
```

Dot products of two vectors!

```
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
- Great documentation and user community (PinHeads)
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 hard to know from inspecting the code the ratios of different operations and matrix/vector sizes which happen when processing real events
- 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
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 3x3, 3x5, 5x3 and 5x5 matrices.
- This information is used for testing different vector math libraries.
- 4x4 is 3D rotation(3x3) plus translation and vectorizes better.
- Eigen performed best and is currently being implemented.

<table>
<thead>
<tr>
<th>Speedup wrt CLHEP</th>
<th>CLHEP</th>
<th>Intel MKL</th>
<th>S-Matrix</th>
<th>Eigen</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{5 \times 3} \times B_{3 \times 5}$</td>
<td>0.7</td>
<td>2.3</td>
<td>4.0</td>
<td>5.0</td>
</tr>
<tr>
<td>$A_{4 \times 4} \times B_{4 \times 4}$</td>
<td>1.3</td>
<td>4.0</td>
<td>6.0</td>
<td>7.0</td>
</tr>
<tr>
<td>$A_{5 \times 3} \times B_{3 \times 5} + \alpha C_{5 \times 5}$</td>
<td>2.0</td>
<td>5.0</td>
<td>7.0</td>
<td>8.0</td>
</tr>
</tbody>
</table>
Using pin results

- From pin instrumentation we observed frequent use of 3x3, 3x5, 5x3 and 5x5 matrices.
- This information is used for testing different vector math libraries.
- 4x4 is 3D rotation(3x3) plus translation and vectorizes better.
- Eigen performed best and is currently being implemented.

Comparison of Math libraries

- **Eigen** is obvious choice.
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 easier.

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
References

- GOODA home page
- Pin homepage
VOID CLHEPHepSymMatrixCopyConst(ADDRINT addr,
    std::map<int,uint64_t> &counts,
    CLHEP::HepSymMatrix const& par1){
    int hash=calcHash(par1.num_row(),par1.num_col());
    std::map<int,uint64_t>::iterator it;
    if((it=counts.find(hash))==counts.end()){
        counts[hash]=1;
    }else{
        it->second++;
    }
}