Optimizing the ATLAS code with different profilers

Sami Kama[1]
on behalf of the ATLAS Collaboration[2]

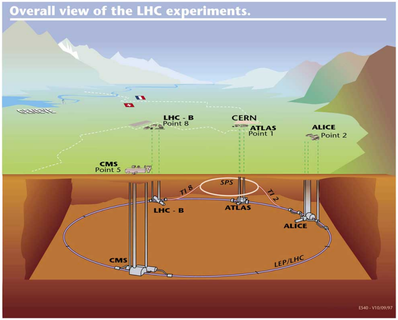
[1]Southern Methodist University, Texas

[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 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 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 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 PB of raw data.
- With the increase in LHC energy, collision rate, event complexity and trigger output ATLAS software needs to speedup considerably

ATLAS Software (ATHENA)

- 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
- Writen 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 ~2000 packages
- Producing 4k+ libraries of various sizes
- Evolving for more than 10 years
- Writen 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

Need tools to point out problematic code!

Profilers commonly used in ATLAS

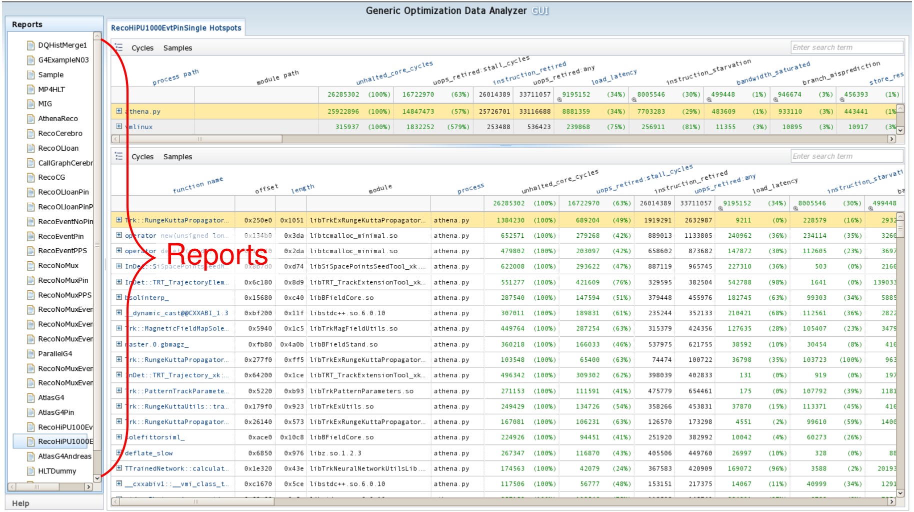
ATLAS uses various tools to profile and monitor ATHENA

- 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.

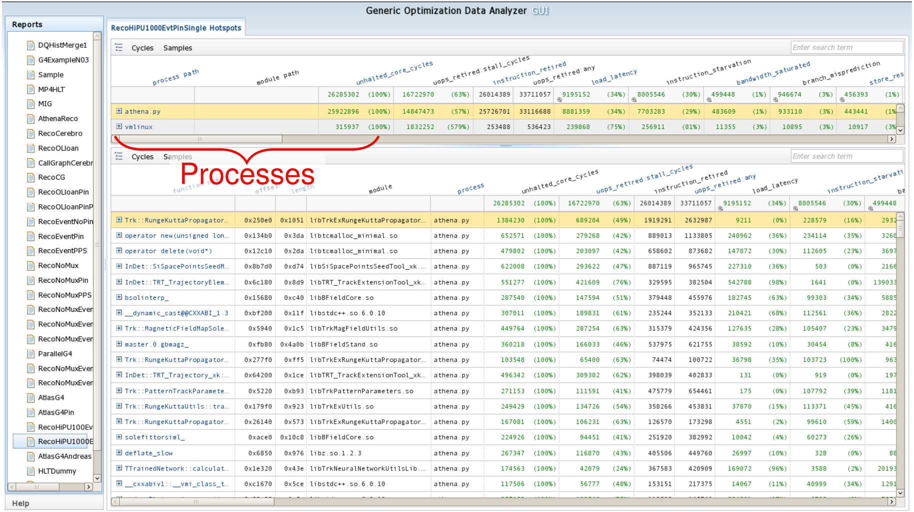
Google Data Analyzer (GOODA)

- 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

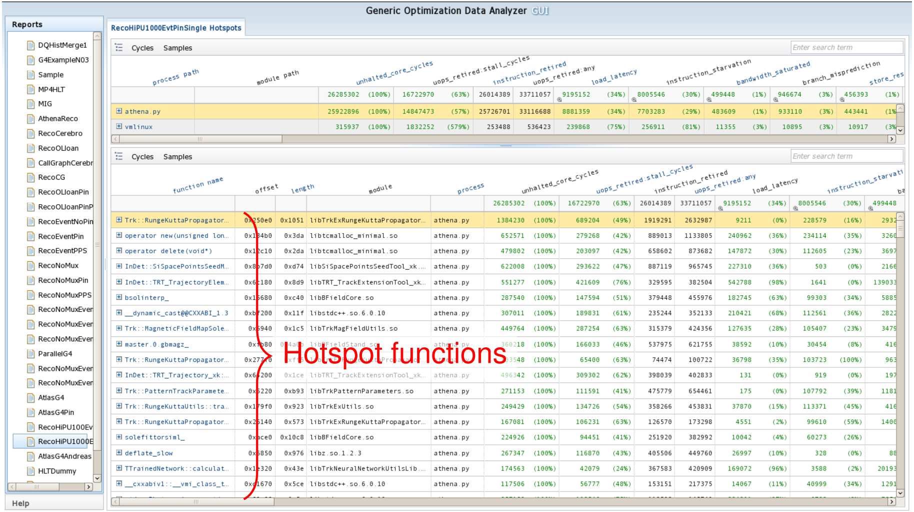
GOODA Example



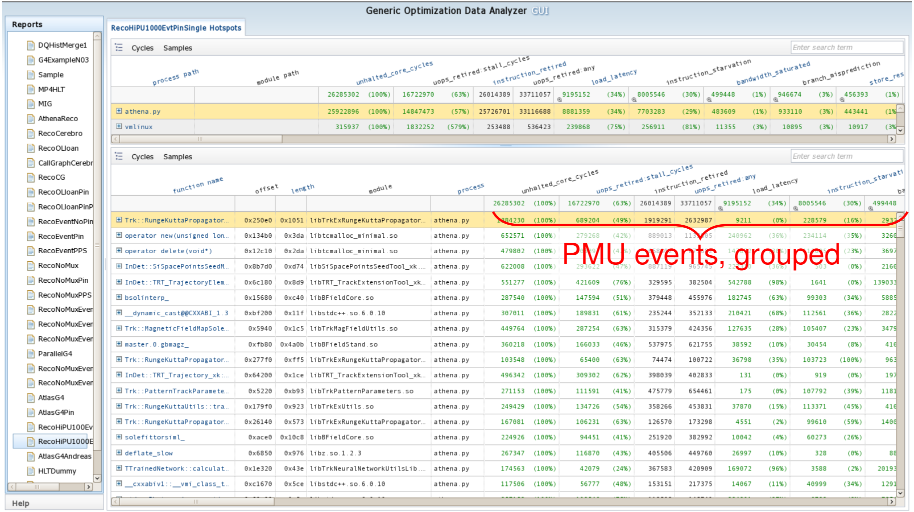
GOODA Example



GOODA Example



GOODA Example



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

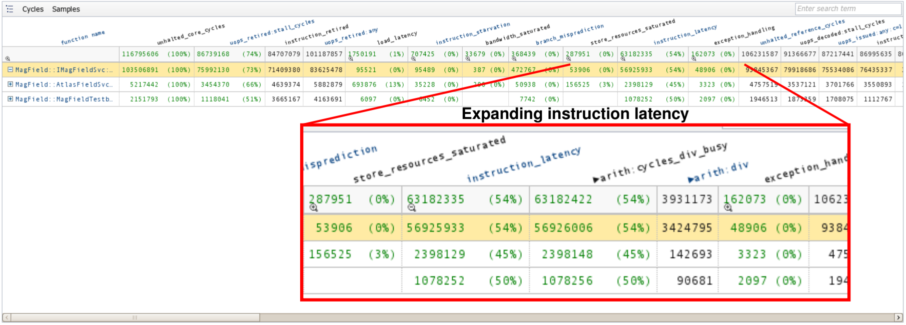
Cycles Samples

Enter search term

function name	unhalted_core_cycles	uops_retired:stall_cycles	instruction_retired	uops_retired:any	load_latency	instruction_starvation	bandwidth_saturated	branch_misprediction	store_resources_saturated	instruction_latency	exception_handling	unhalted_reference_cycles	uops_decoded:stall_cycles	uops_issued:any:c=1	instruct
	116795606 (100%)	86739168 (74%)	84707079	101187857	1750191 (1%)	707425 (0%)	33679 (0%)	368439 (0%)	287951 (0%)	63182335 (54%)	162073 (0%)	106231587	91366677	87217441	86995635
⊟ MagField::IMagFieldSvc:…	103506891 (100%)	75992130 (73%)	71409380	83625478	95521 (0%)	95489 (0%)	387 (0%)	472767 (0%)	53906 (0%)	56925933 (54%)	48906 (0%)	93845367	79918686	75534086	76435337
⊞ MagField::AtlasFieldSvc…	5217442 (100%)	3454370 (66%)	4639374	5882879	693876 (13%)	35228 (0%)	290 (0%)	50938 (0%)	156525 (3%)	2398129 (45%)	3323 (0%)	4757519	3537121	3701766	3550893
⊞ MagField::MagFieldTestb…	2151793 (100%)	1118041 (51%)	3665167	4163691	6097 (0%)	6452 (0%)		7742 (0%)		1078252 (50%)	2097 (0%)	1946513	1873359	1708075	1112767

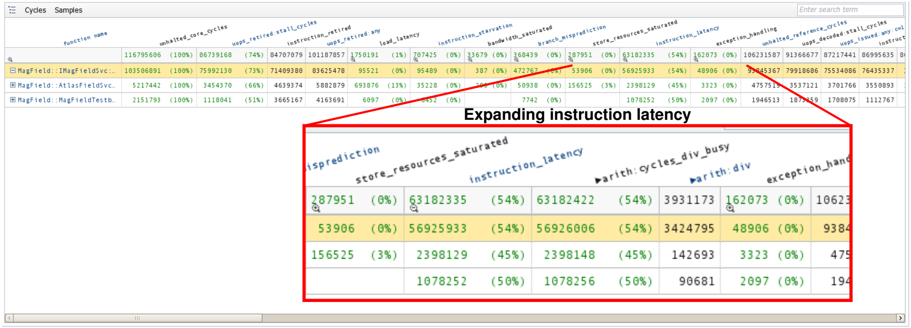
Improving new Magnetic Field

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



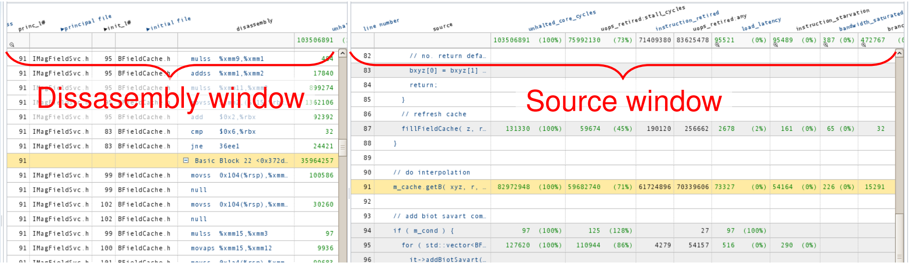
Improving new Magnetic Field

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



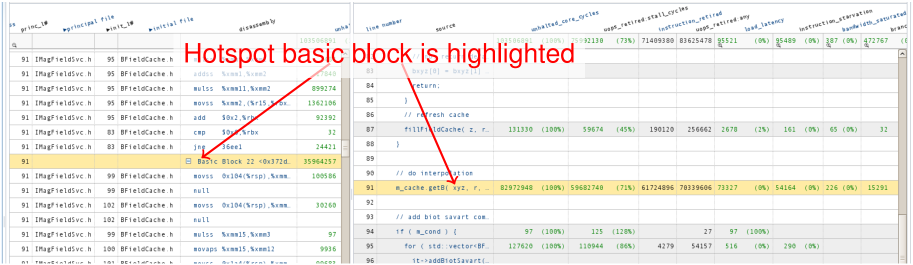
High instruction latency originating from division operations. Can drill down (double-click) for details

Magnetic field code details



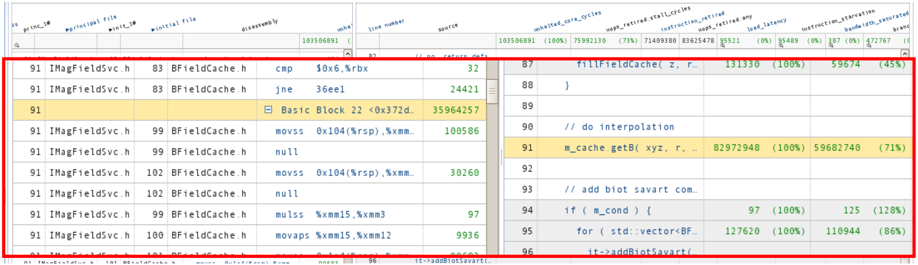
- Detailed view contains both dissassembly and source code if debug symbols and source files are available
- Events are displayed at instruction level and hottest basic block is automatically highlighted
- Debug symbols in optimized builds are skewed, instruction latency is coming from another file.

Magnetic field code details



- Detailed view contains both dissassembly 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 dissassembly 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

```
        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[0] = cc*dBdr[1] - cs*dBdr[2] - cs*dBdphi[1]/r + ss*dBdphi[2]/r + s*B[1]/r;
        deriv[1] = cs*dBdr[1] - ss*dBdr[2] + cc*dBdphi[1]/r - cs*dBdphi[2]/r - c*B[1]/r;
        deriv[2] = c*dBdz[1] - s*dBdz[2];
        deriv[3] = cs*dBdr[1] + cc*dBdr[2] - ss*dBdphi[1]/r - cs*dBdphi[2]/r - s*B[0]/r;
        deriv[4] = ss*dBdr[1] + cs*dBdr[2] + cs*dBdphi[1]/r + cc*dBdphi[2]/r + c*B[0]/r;
        deriv[5] = s*dBdz[1] + c*dBdz[2];
        deriv[6] = c*dBdr[0] - s*dBdphi[0]/r;
        deriv[7] = s*dBdr[1] + c*dBdphi[0]/r;
        deriv[8] = dBdz[0];
    }
}
```

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]) ) );
        }
        // convert to cartesian coordinates
        float cc = c*c;
        float cs = c*s;
        float ss = s*s;
        deriv[0] = cc*dBdr[1] - cs*dBdr[2] - cs*dBdphi[1]/r + ss*dBdphi[2]/r + s*B[1]/r;
        deriv[1] = cs*dBdr[1] - ss*dBdr[2] + cc*dBdphi[1]/r - cs*dBdphi[2]/r - c*B[1]/r;
        deriv[2] = c*dBdz[1] - s*dBdz[2];
        deriv[3] = cs*dBdr[1] + cc*dBdr[2] - ss*dBdphi[1]/r - cs*dBdphi[2]/r - s*B[0]/r;
        deriv[4] = ss*dBdr[1] + cs*dBdr[2] + cs*dBdphi[1]/r + cc*dBdphi[2]/r + c*B[0]/r;
        deriv[5] = s*dBdz[1] + c*dBdz[2];
        deriv[6] = c*dBdr[0] - s*dBdphi[0]/r;
        deriv[7] = s*dBdr[1] + c*dBdphi[0]/r;
        deriv[8] = dBdz[0];
    }
}
```

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

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]) ) );
        }
        // convert to cartesian coordinates
        float cc = c*c;
        float cs = c*s;
        float ss = s*s;
        deriv[0] = cc*dBdr[1] - cs*dBdr[2] - cs*dBdphi[1]/r + ss*dBdphi[2]/r + s*B[1]/r;
        deriv[1] = cs*dBdr[1] - ss*dBdr[2] + cc*dBdphi[1]/r - cs*dBdphi[2]/r - c*B[1]/r;
        deriv[2] = c*dBdz[1] - s*dBdz[2];
        deriv[3] = cs*dBdr[1] + cc*dBdr[2] - ss*dBdphi[1]/r - cs*dBdphi[2]/r - s*B[0]/r;
        deriv[4] = ss*dBdr[1] + cs*dBdr[2] + cs*dBdphi[1]/r + cc*dBdphi[2]/r + c*B[0]/r;
        deriv[5] = s*dBdz[1] + c*dBdz[2];
        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

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]) ) );
        }
        // convert to cartesian coordinates
        float cc = c*c;
        float cs = c*s;
        float ss = s*s;
        deriv[0] = cc*dBdr[1] - cs*dBdr[2] - cs*dBdphi[1]/r + ss*dBdphi[2]/r + s*B[1]/r;
        deriv[1] = cs*dBdr[1] - ss*dBdr[2] + cc*dBdphi[1]/r - cs*dBdphi[2]/r - c*B[1]/r;
        deriv[2] = c*dBdz[1] - s*dBdz[2];
        deriv[3] = cs*dBdr[1] + cc*dBdr[2] - ss*dBdphi[1]/r - cs*dBdphi[2]/r - s*B[0]/r;
        deriv[4] = ss*dBdr[1] + cs*dBdr[2] + cs*dBdphi[1]/r + cc*dBdphi[2]/r + c*B[0]/r;
        deriv[5] = s*dBdz[1] + c*dBdz[2];
        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
- 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

Hottest 10 CLHEP Functions

Calls	Instr	<instr>/call	Call rank	Function
1778523	6392431813	3594.24	1439	CLHEP::operator*(CLHEP::HepMatrix const&, CLHEP::HepSymMatrix const&)
671676353	5988139520	8.92	9	CLHEP::Hep2Vector::operator()(int) const
232093102	5956556656	25.66	27	HepGeom::Transform3D::operator()(int, int) const
285282108	3709057782	13.00	21	CLHEP::Hep3Vector::operator()(int)
15815930	3179001930	201.00	319	CLHEP::HepRotation::rotateAxes(CLHEP::Hep3Vector const&, CLHEP::Hep3Vector const&, CLHEP::Hep3Vector const&)
20529818	2422518524	118.00	267	HepGeom::Transform3D::inverse() const
31612743	2212258670	69.98	200	CLHEP::HepSymMatrix::HepSymMatrix(CLHEP::HepSymMatrix const&)
28914115	1929106393	66.72	214	CLHEP::HepVector::HepVector(int, int)
51974716	1819115060	35.00	150	HepGeom::operator*(HepGeom::Transform3D const&, HepGeom::Point3D<double> const&)
27652274	1506352669	54.47	219	CLHEP::HepVector::HepVector(CLHEP::HepVector const&)

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

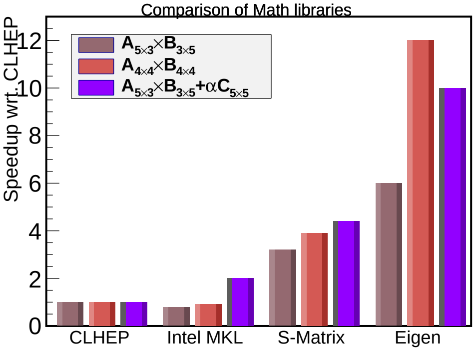
CLHEP Instrumentation

- 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 futher offline processing

```
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
 }
}
```

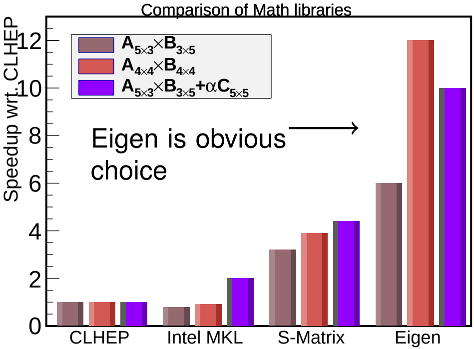
Using pin results

- 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.



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.



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 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

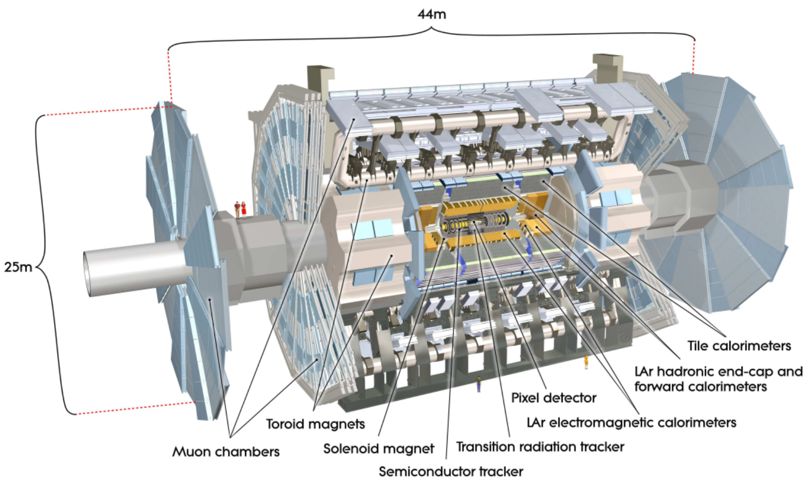
Thanks to

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

References

- GOODA home page
- Pin homepage

ATLAS


44m
25m
Tile calorimeters
LAr hadronic end-cap and
forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Muon chambers
Solenoid magnet
Transition radiation tracker
Semiconductor tracker

Instrumenting CLHEP::HepSymMatrix(const CLHEP::HepSymMatrix&)

```
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++;
  }
}
```