Multi-threaded ATLAS Simulation on Intel Knights Landing Processors

Steve Farrell, Paolo Calafiura, Charles Leggett, Vakho Tsulaia, Andrea Dotti,
on behalf of the ATLAS collaboration

CHEP 2016
San Francisco

Berkeley Lab
Ernest Orlando Lawrence Berkeley National Laboratory

Sep 30, 2016

Overview

- Many-integrated-core (MIC) architectures
  - Intel Xeon Phi product family
  - Knights Landing processors
  - MIC-equipped supercomputers
- Atlas multi-threaded simulation
  - Design and parallelism
- Performance measurements
  - Throughput and memory scaling
  - CPU profiling studies

Setting the stage

- The multi-core era is not news anymore, but we're seeing some significant shifts in processor trends as time evolves
  - Increasing number of cores with transistor scaling
  - Less memory per core (in practice) due to RAM costs
  - Slower, less-sophisticated cores due to power concerns
  - Increasing capabilities (and importance) of vector processing
- Nvidia general-purpose GPUs are an "extreme" example
  - Highly parallel, simple cores
  - Requires highly adapted code and use of non-trivial libraries/APIs (e.g. CUDA)
- Intel's answer: a highly parallel many-core Linux device
  - "A supercomputer on a chip" with a familiar programming model

Intel Many-Integrated-Core architecture

- A "supercomputer on a chip"
  - Lots of threads, wide vector registers, with low power footprint
  - Particularly suited to highly-parallel, CPU-bound applications

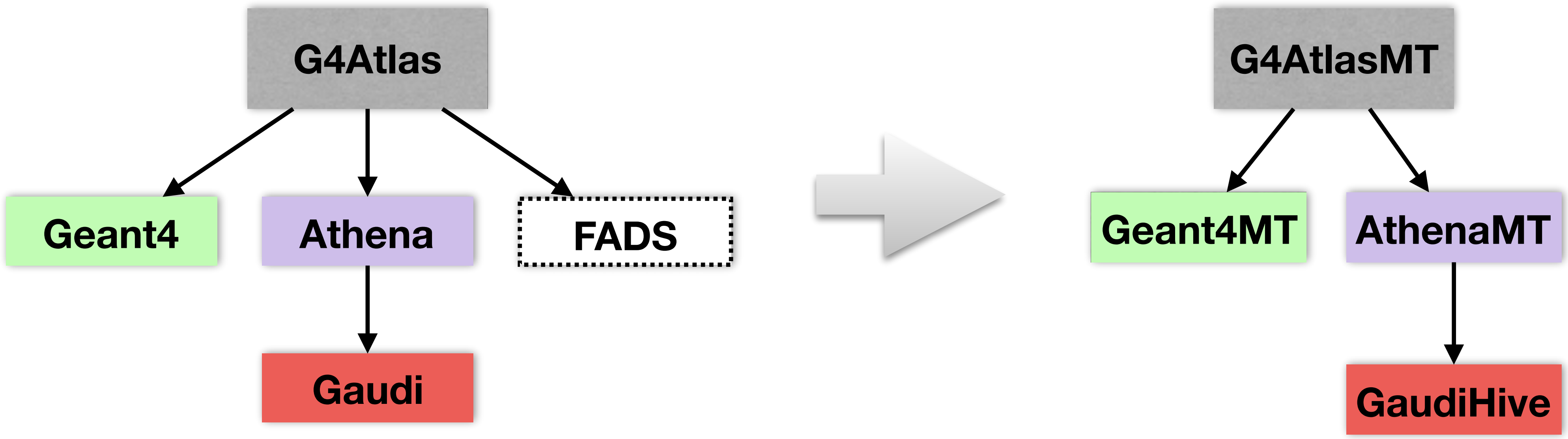
- The Xeon Phi product line:

Knights Corner (KNC)	**Knights Landing (KNL)**	**Knights Hill (KNH)**
previous generation	current generation	maybe 2017
57-61 Pentium cores (~1GHz)	72 Airmont cores (3x faster)	60-72 Silvermont cores
6-16 GB on-chip RAM	8-16 GB MCDRAM	???
coprocessor only	up to 384 GB RAM	
	host or coprocessor	

- Supercomputers:

• Tianhe-2 @ NSCC-GZ • Stampede @ TACC	• Cori @ NERSC • Theta @ ANL	• Aurora @ ANL

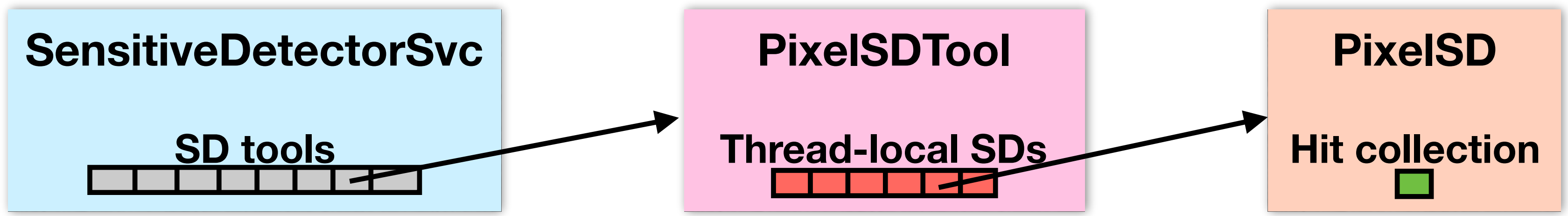
Multi-threaded ATLAS simulation



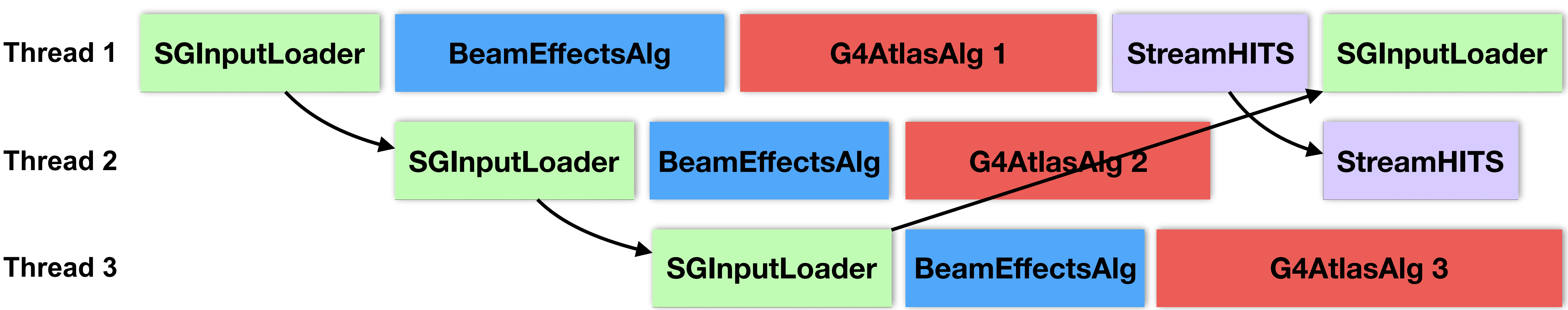
- The time is ripe for multi-threading
  - Multi-threaded version of Gaudi being integrated into AthenaMT framework
  - Multi-threaded version of Geant4 available and shown to perform well
  - Overhaul of ATLAS simulation infrastructure with thread-safety in mind
    - See Andrea Di Simone's presentation this week
- Some challenges
  - Marriage of dependencies with different models of concurrency
    - Gaudi's task-parallelism with Intel's Threading Building Block
    - Geant4's master-worker event-parallelism with pthreads and thread-local-storage
  - Mechanisms needed to setup and manage thread-local Geant4 workspace
  - A lot of legacy simulation and core code which needs thread-safety updates/rewrites

Thread-safe design

- **Geant4 components vs. Athena components**
  - Thread-shared Athena components create and manage thread-local Geant4 components



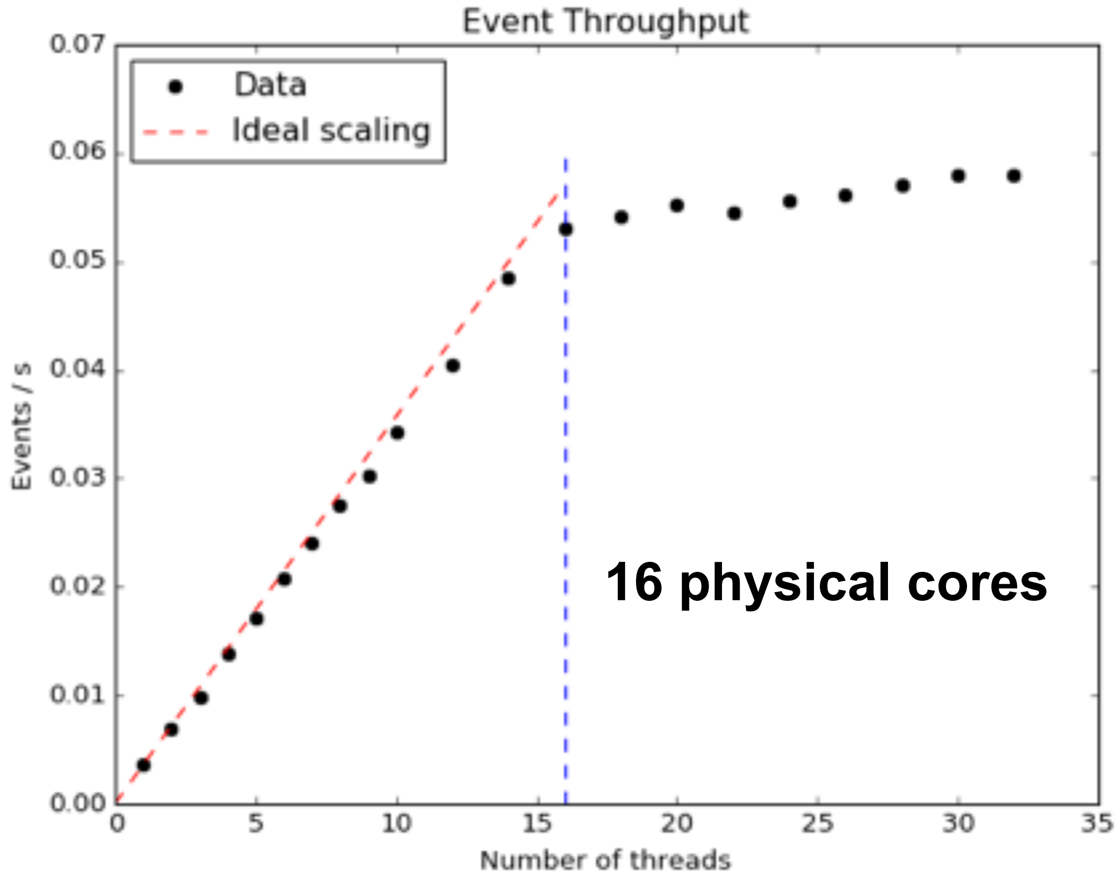
- **Thread setup/teardown mechanism**
  - ThreadPoolSvc supports ThreadInitTools invoked simultaneously on all worker threads before and after the event loop
    - Used to initialize the Geant4 thread-local workspaces (geo, physics, etc.)
- **Execution and scheduling**
  - Event-processing algorithms are cloned to execute concurrently on each worker thread
    - G4AtlasAlg handles bulk of processing by passing one event to Geant4
    - BeamEffectsAlg applies some corrections/smearing to the input generated event
  - Two I/O algorithms are serialized due to thread-unsafe POOL layer: SGInputLoader, StreamHITS

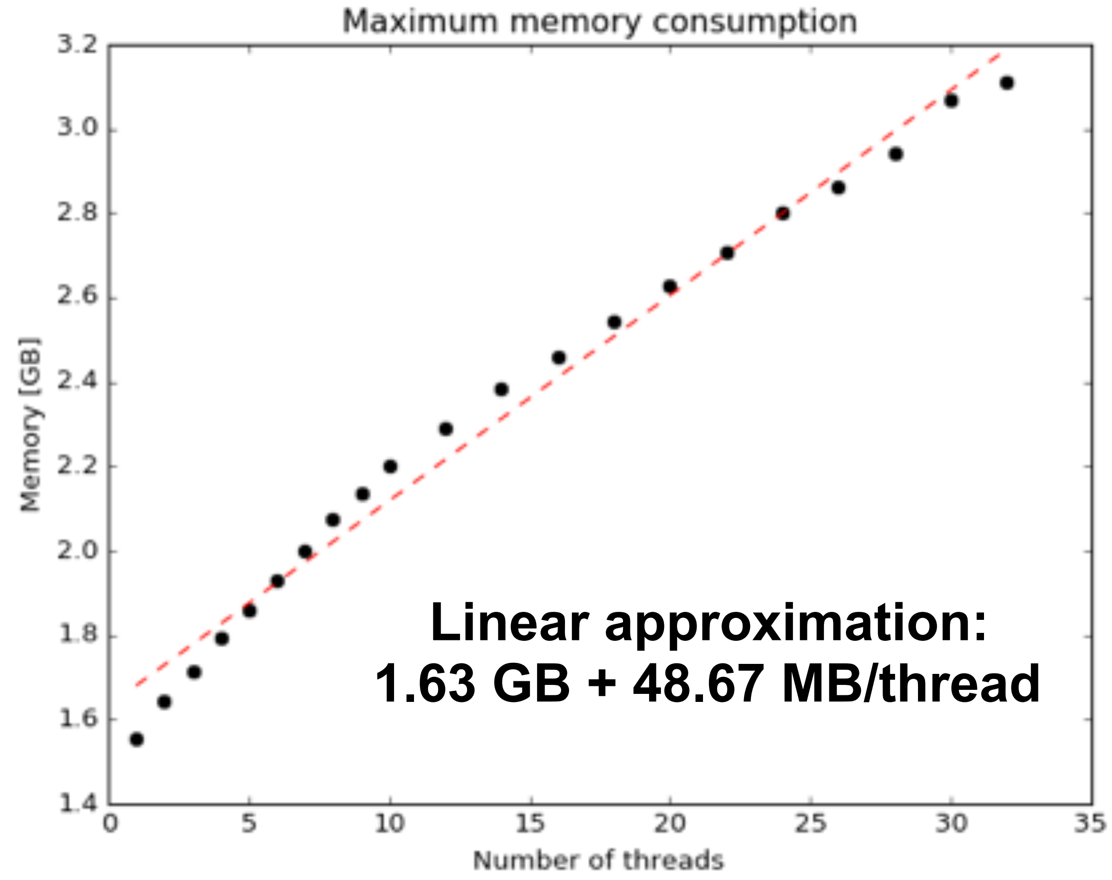


Status of the migration

- **Multi-threaded full Geant4 simulation nearly complete**
  - Geometry, physics, most sensitive detectors were straight-forward
    - including custom endcap calorimeter geometry
  - User actions working, though design somewhat complicated by our requirements and could possibly be simplified
    - a lot of our customized event handling happens here
  - Preliminary version of truth code works
    - though we're in the progress of updating the implementation
  - Magnetic field is working
    - we use a thread-shared field service with thread-local caching
- **Few missing features still in progress**
  - LAr sensitive detectors are highly complicated and not yet thread-safe
  - Some of the filtering mechanisms not yet working in MT
  - Frozen calorimeter showers implemented and in testing
- **Additional things that will require more work**
  - Fast-simulations like FastCaloSim (AF2) and FATRAS
  - Multi-threading in the Integrated Simulation Framework (ISF)
  - Full validation of the multi-threaded simulation

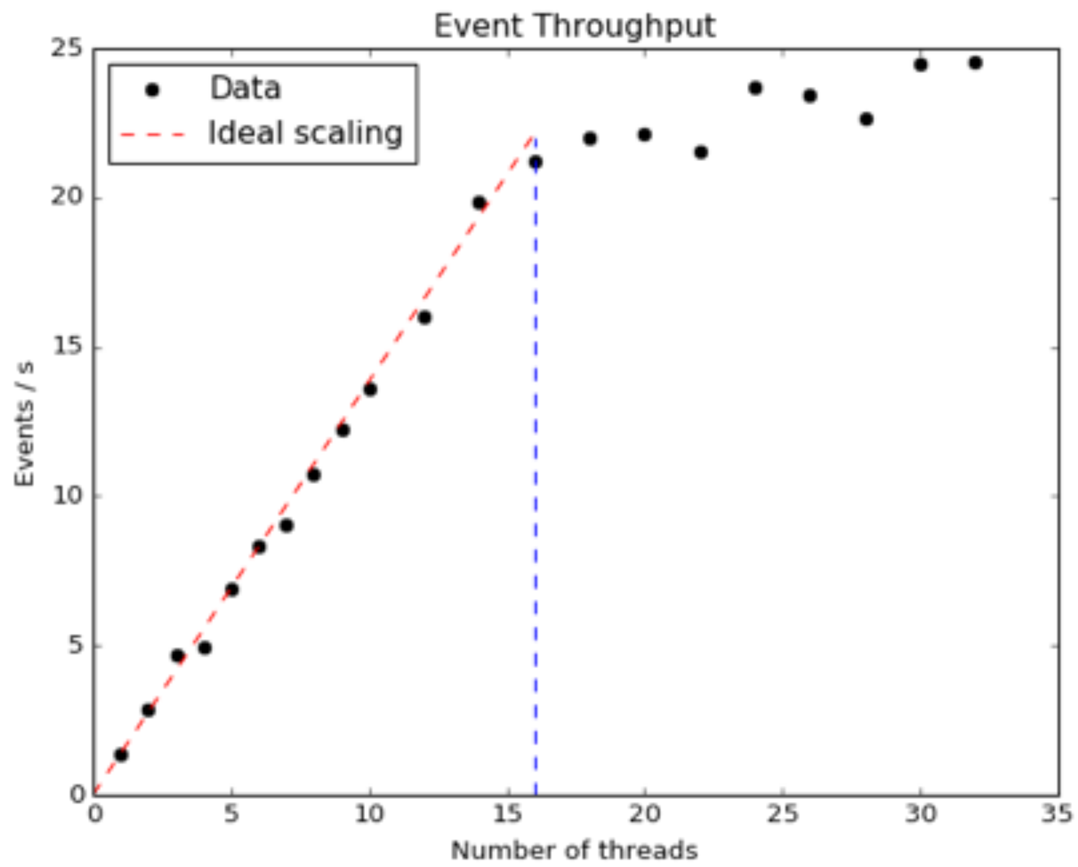
Scaling on a Xeon - ttbar sample

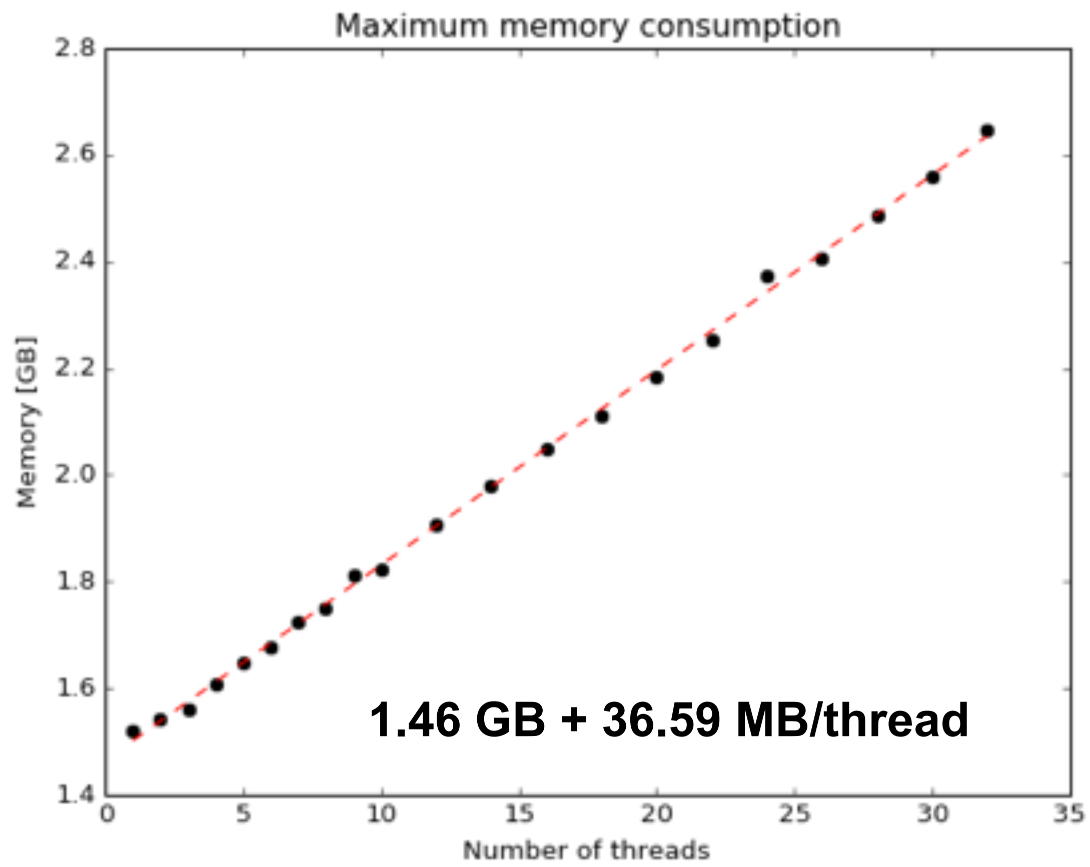




- Event throughput scales very well up to the physical number of cores, and plateaus quite abruptly in hyper-threading regime
- Memory scales nicely, showing excellent savings from sharing across threads
- Unfortunately, this sample is difficult to test with on a KNL due to long event processing times, so we switch to a faster single-muon simulated sample

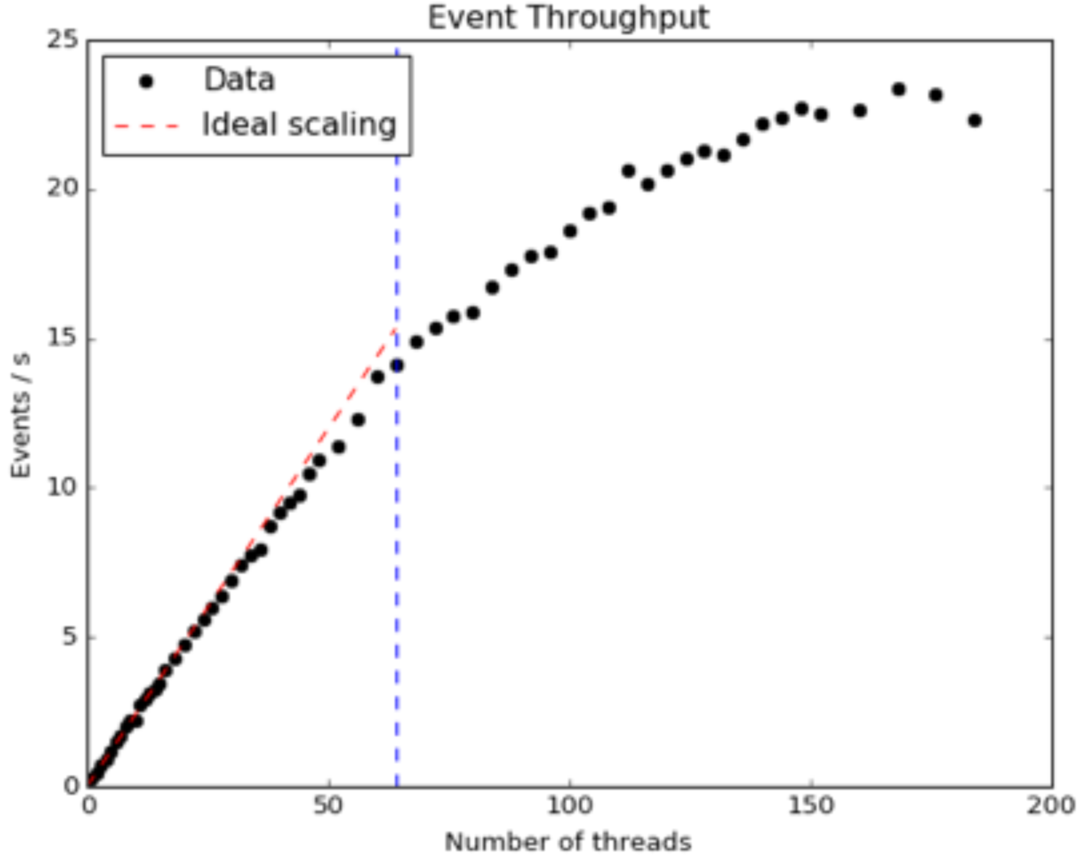
Scaling on a Xeon - single-muon sample

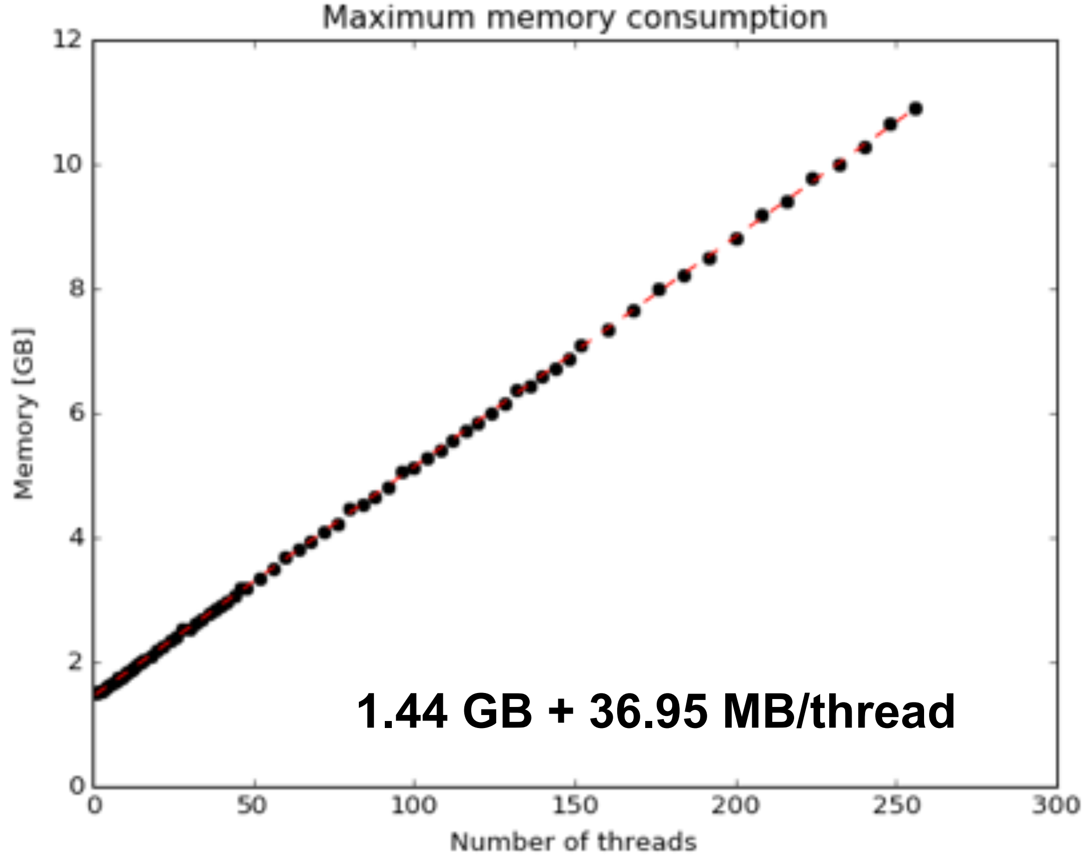




- As with the ttbar sample, the scaling with the single-muon sample is excellent up to the physical number of cores
- The memory scaling is also good again
- The characteristics of these results reasonably agree with the ttbar sample, which gives some confidence that we can continue making measurements with the single-muon sample

Scaling on a Xeon Phi - single-muon sample





- Throughput scaling is nearly perfect up to the physical number of cores, with a lot of improvement gained in the hyper-threading regime
- Throughput maxes out around 170 threads, but starts to turn down above that
- Memory continues to scale very well over the entire thread scaling range
- Maximum throughput achieved on KNL is fairly consistent with maximum throughput on the 16-core Xeon

Xeon vs. Xeon Phi performance

Threads	Xeon throughput [events/s]	Xeon Phi throughput [events/s]	Throughput ratio (KNL slowdown)
1	1.38	0.24	5.78
4	4.88	0.91	5.39
6	8.24	1.47	5.62
8	10.54	1.78	5.91
12	15.17	2.76	5.50
16	20.50	3.68	5.57
24	22.51	5.16	4.36
32	24.24	7.24	3.35

- Per-core performance is about 5.5 times worse on KNL compared to Ivy-bridge Xeon.

Profiling the application

- Using VTune, we can start to understand the performance differences between the Xeon and Xeon Phi architectures
    - These results measured with a Zμμ sample and a single worker thread

Architecture	CPI rate	Front-end bound	ICache misses	Bad speculation	Back-end bound
KNL	3.0	60.2%	0.96	2.4%	18.6%
Haswell	0.9	31.5%	0.086	11.7%	27.6%

- On KNL, the application seems to be held up in the instruction front-end, with a high clocks-per-instruction rate of 3.0!
    - High rate of instruction cache misses
    - Seems to be due to relatively poor handling of large ATLAS+G4 code size

Application hotspots

- **Hotspots on a Haswell machine** (Zμμ sample, single worker thread):

Function	Clockticks	Instructions Retired	CPI Rate	Front-End Bound(%)	Bad Speculation(%)	Back-End Bound(%)
G4PhysicsVector::Value	132,160,198,240	107,000,160,500	1.235	0.164	9.0%	0.557
__sin_avx	129,540,194,310	71,740,107,610	1.806	0.344	26.9%	0.197
sincos	118,360,177,540	44,420,066,630	2.665	0.414	27.6%	0.170
__cos_avx	117,680,176,520	25,380,038,070	4.637	0.459	24.2%	0.203
LArWheelCalculator_Impl::DistanceCalculatorSaggingOff::	110,760,166,140	196,640,294,960	0.563	0.051	8.8%	0.355
__ieee754_log_avx	75,140,112,710	36,560,054,840	2.055	0.338	22.5%	0.283
__ieee754_atan2_avx	72,300,108,450	56,620,084,930	1.277	0.373	21.3%	0.185
G4Navigator::LocateGlobalPointAndSetup	67,680,101,520	49,380,074,070	1.371	0.313	6.2%	0.471
LArWheelCalculator::parameterized_sin	67,460,101,190	92,560,138,840	0.729	0.076	24.9%	0.284
MagField::AtlasFieldSvc::getField	58,240,087,360	57,800,086,700	1.008	0.144	19.5%	0.472

- **Hotspots on a KNL machine** (same config):

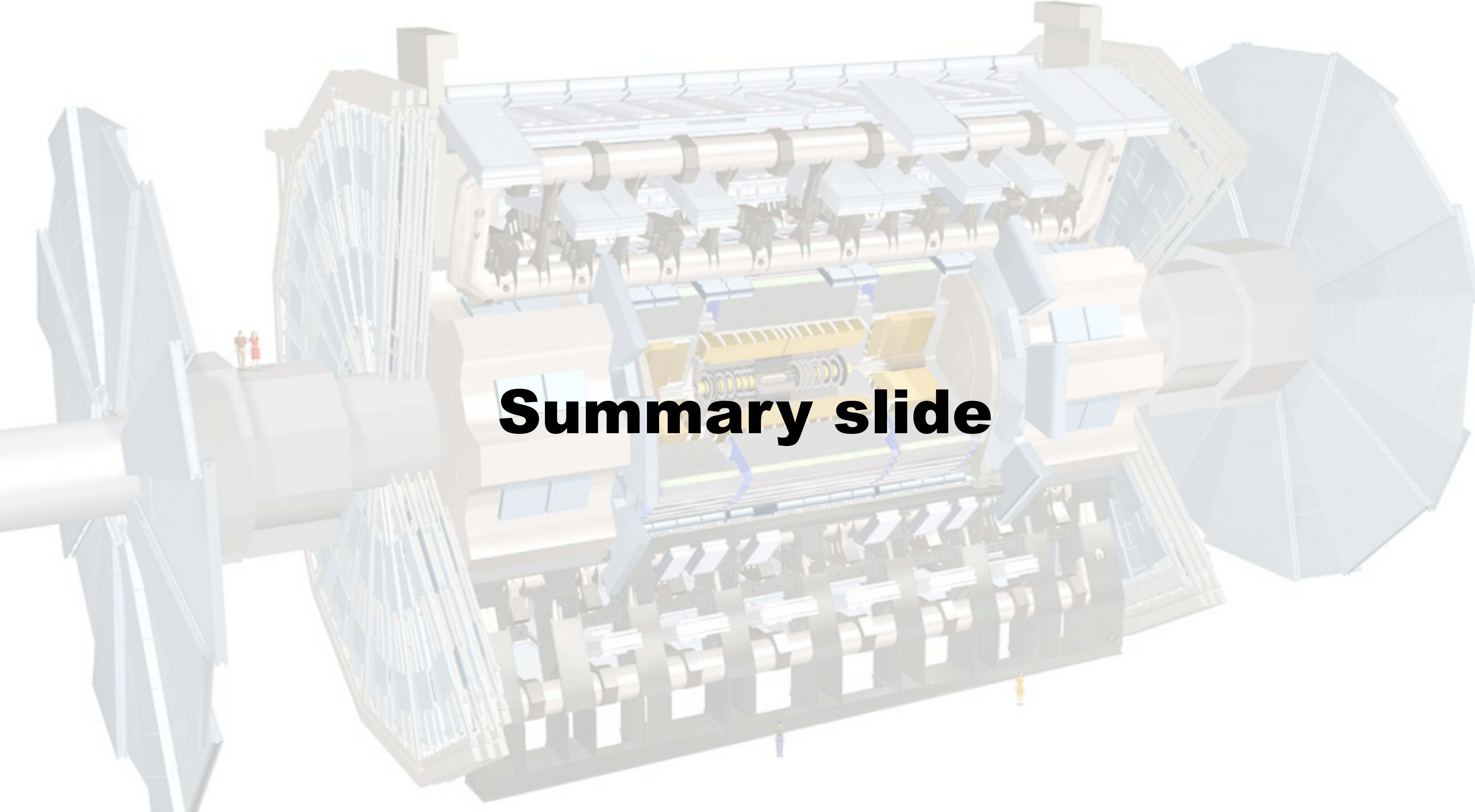
Function	Clockticks	Instructions Retired	CPI Rate	Front-End Bound(%)	Bad Speculation(%)	Back-End Bound(%)
G4PhysicsVector::Value	333,820,500,730	82,760,124,140	4.034	0.567	2.8%	0.270
LArWheelCalculator::parameterized_sin	257,140,385,710	189,920,284,880	1.354	0.456	0.3%	0.125
LArWheelCalculator_Impl::DistanceCalculatorSaggingOff::	235,160,352,740	127,340,191,010	1.847	0.161	4.1%	0.443
G4Navigator::LocateGlobalPointAndSetup	194,320,291,480	42,160,063,240	4.609	0.620	1.2%	0.246
__tls_get_addr	193,620,290,430	49,360,074,040	3.923	0.638	1.4%	0.207
G4SteppingManager::DefinePhysicalStepLength	189,300,283,950	64,240,096,360	2.947	0.658	2.2%	0.150
__sin_avx	174,540,261,810	58,140,087,210	3.002	0.441	1.3%	0.354
G4Navigator::ComputeStep	166,540,249,810	41,240,061,860	4.038	0.767	0.3%	0.093
MagField::AtlasFieldSvc::getField	158,660,237,990	55,860,083,790	2.840	0.451	1.9%	0.333
BFieldCache::getB	156,520,234,780	106,760,160,140	1.466	0.159	0.7%	0.501

- The lists are fairly similar
  - The KNL slowdown doesn't seem to be due to any particular piece of code, but rather a global slowdown of the entire codebase

Conclusion

- ATLAS can now run a nearly complete multi-threaded simulation setup in AthenaMT
  - Throughput and memory scaling performance look quite good so far
- Intel Xeon Phi architectures appear to be a reasonable target resource for such an application
  - The x86 compatibility promise from Intel has been fulfilled
- Knights Landing machines give throughput comparable to a 16-core Ivy Bridge
  - We seem to be limited by CPU front-end, probably due to poor code layout
  - There's still some room for improvement to improve scaling for certain configurations beyond 180 threads on the KNL
- It's clear that we'll be able to utilize NERSC's Cori Phase II for ATLAS simulation
  - but to use it effectively we've still got some work to do

Summary slide



ATLAS MT simulation on KNL

- ATLAS simulation is being migrated to multi-threading
  - Event-level parallelism based on Geant4 and AthenaMT
  - Nearly complete full simulation configuration (G4AtlasMT) now ready
- Intel's new Knights Landing generation of Intel Xeon Phi processors is a good target for this type of application
  - Highly parallel architecture for CPU-heavy code
- G4AtlasMT shows good scaling performance on both Xeon and Xeon Phi architectures