Geant4 On Intel Xeon Phi

A. Dotti on behalf of Geant4 collaboration
Wim Lavrijsen for LBL CRD group
Concurrency Forum Annual Meeting, 2014

SLAC NATIONAL ACCELERATOR LABORATORY

Introduction

SLAC

- Porting of the code to Intel Xeon Phi
- Results
- Checkpointing
- Towards a complete system: G4-ATLAS on MIC, first impressions
  - Collaboration SLAC - Berkeley for ATLAS

Introduction

SLAC National Accelerator Laboratory

Photo Credit: Peter Ginter

Porting of the code / configuration

SLAC

- MT functionalities in Geant4 using POSIX standards: simplified porting to other architectures/OS
- Intel Compiler already supported since long
- **No need to change a single line of code to (cross-)compile for MIC architecture**

- Geant4 **configuration** system based on cmake
  - After few iterations with experts, in version 10.0 cmake scripts are enough "platform agnostic"
  - Cross-compilation via cmake **toolchain** file (e.g. specify compiler, add -mmic option, specify linker)
- In the plans: provide instructions/support files to users

First round of optimizations

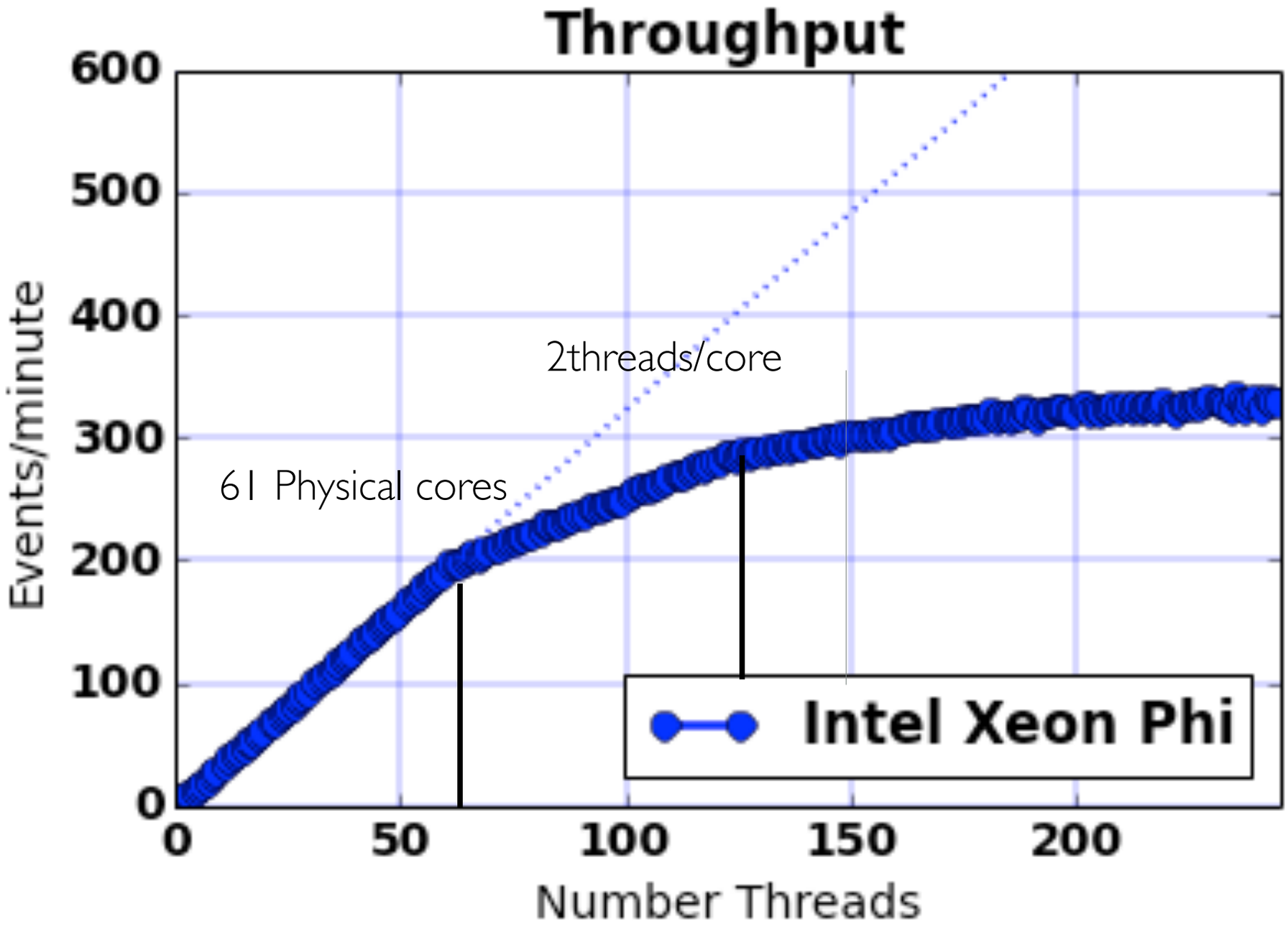
SLAC

- **First goal is to make efficient use of large core-count of Phi**
  - Need to fit application in card memory budget (16GB)
  - Use semi-realistic application as demonstration: CMS geometry, full physics, simple uniform B-Field

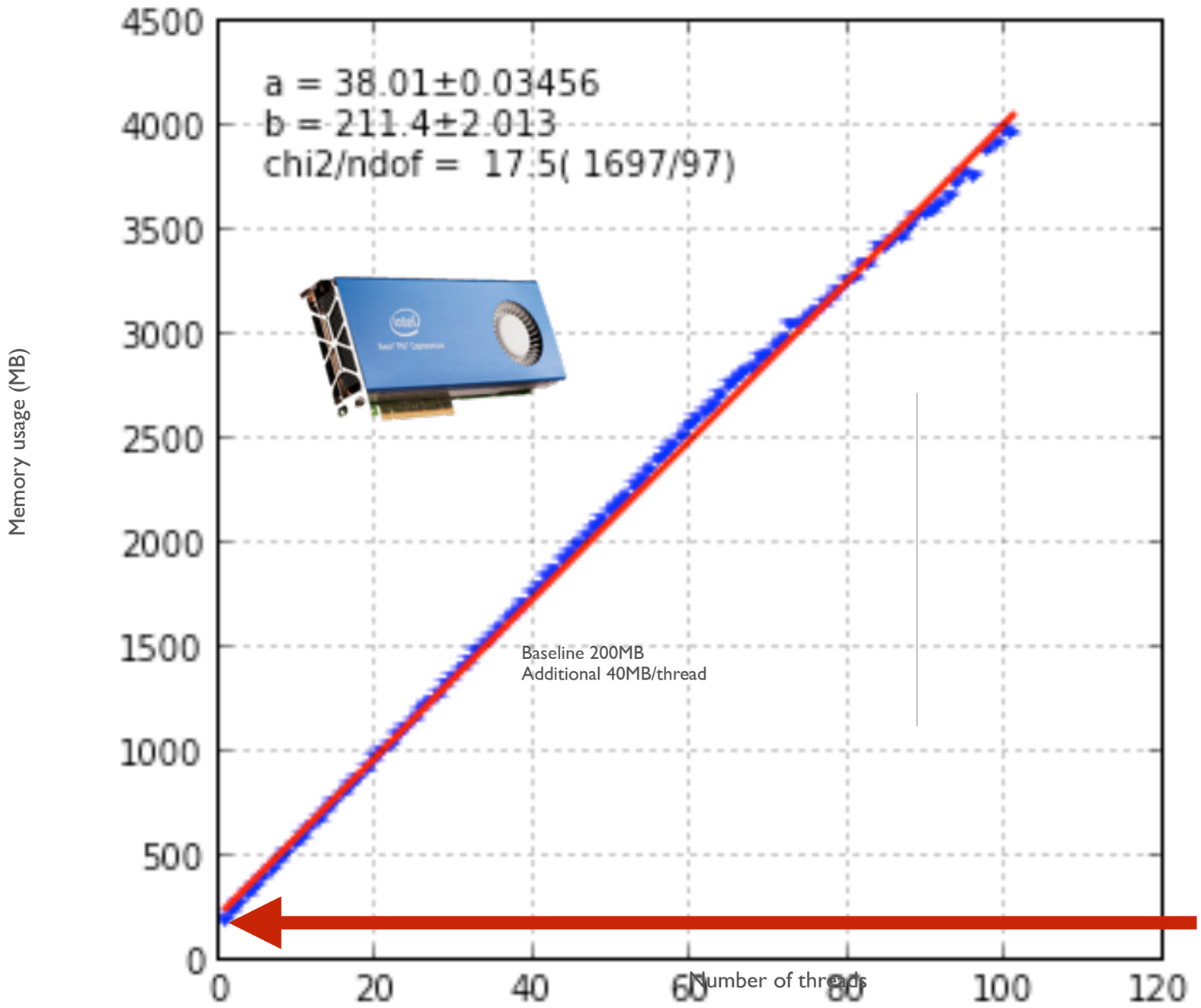
- For Geant4 Version 10 focus on reducing thread memory footprint: currently ~40MB/thread
- **Can run in ~10GB with max thread counts (244)**
- We see opportunities to further substantially reduce memory usage in next Geant4 versions (i.e. hadronic processes data tables)
  - Work already started

Results

Results: linearity



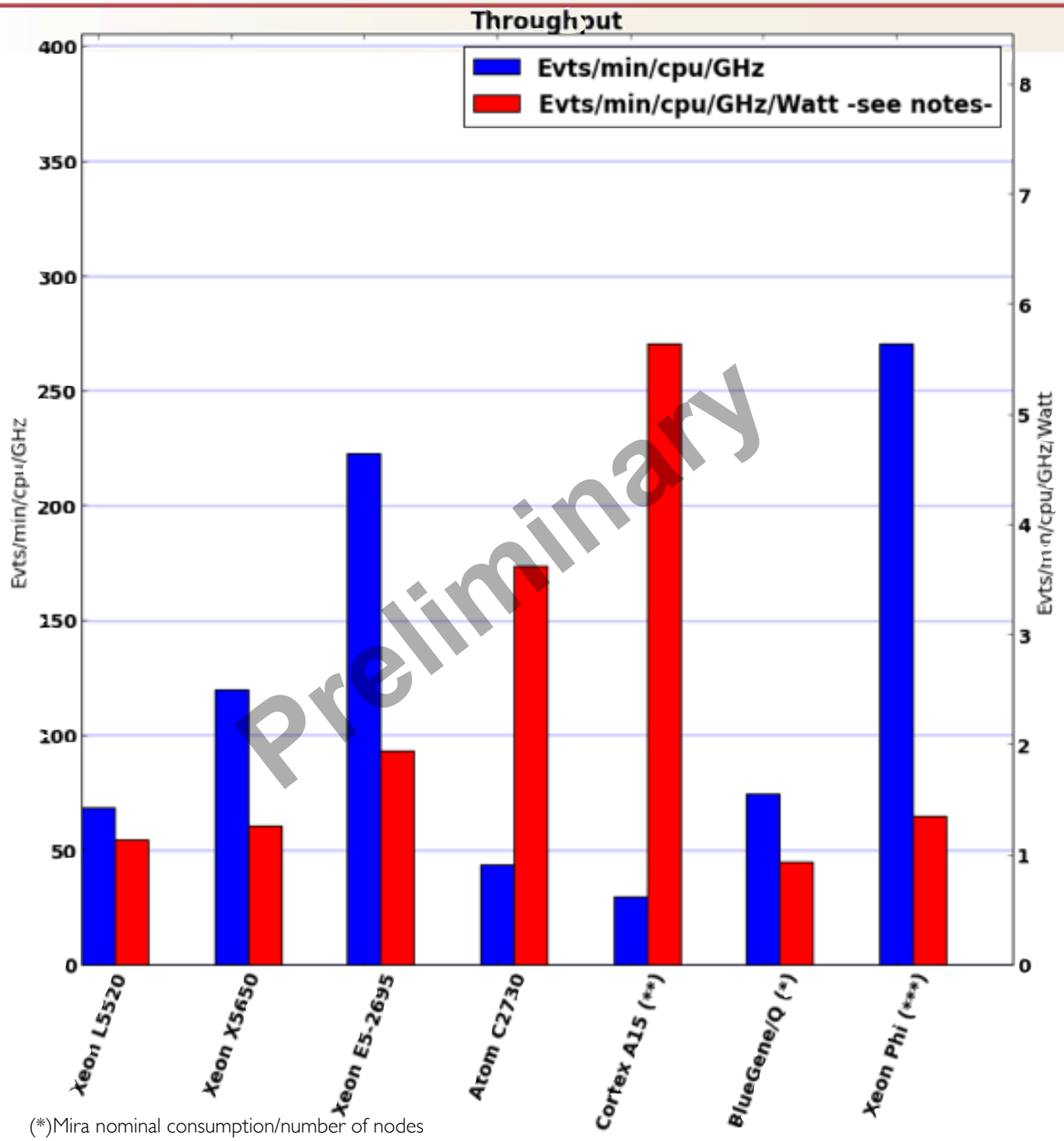
Results: memory usage



Slope: 38MB/thread

Baseline: 0 thread memory consumption

Absolute throughput



(*)Mira nominal consumption/number of nodes

(**) Measured for a ODROID-XU+E evaluation board

(***)Power consumption measured via "Intel Xeon Phi Coprocessor Status Panel"

All other are max TDP specifications

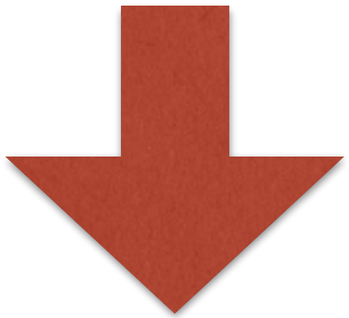
- Throughput normalized per GHz and "socket" (or node / card)
- Not a measure of the absolute performance of a system
- Also reported Throughput/Watt: not realistic (mainly not counting server, very rough!) only to give an idea of what we are talking about
- What is the best "metrics" to compare different architectures?

Absolute performances:

```
====== Max Events/min/cpu =======
154.4619 Intel Xeon L5520@2.27GHz
319.7392 Intel Xeon X5650@2.67GHz
534.6305 Intel Xeon E5-2695 v2@2.40GHz
73.8040 Intel Atom C2730@1.7GHz
46.8705 Exynos 5410 Octa Cortex-A15@1.6GHz
119.2088 BlueGene/Q@1.6GHz
334.4548 Intel Xeon Phi 7120P@1.238GHz
```

Reminders

- **Fully reproducible:** given an event and its initial seed the RNG history is independent of the number of threads and order in which these are simulated
  - Corollary 1: given the seeds, sequential and MT builds are equivalent
  - Corollary 2: being able to reproduce a single event in a dedicated job (i.e. crashes)
- MT functionality introduces **minimal overhead** (~1%) w.r.t. sequential
- Very good **linear speedup** up to very large number of threads O(100)
- Good **memory reduction**: only 30-50MB/thread (depends on application)



- Simulation on Xeon Phi co-processor may be a valuable economic option in some cases

Under Development and near future plans

SLAC

- Test differences of Static Vs Shared library builds on MIC (observed 20% improvements w/ static libs on standard architecture)
- Test PGO optimization options (another 20% observed on host)
- Locking of threads to logical cores (increase cache efficiencies)
- **All these will benefit also traditional architectures**

Under testing and plans (continue)

SLAC

- At the moment prefer hybrid computing model between Host and MIC
  - No need to modify code
  - Better performance results w.r.t. offload
- **Plans is to use MPI to coordinate two jobs:** one on the CPU one for each MIC (hybrid MPI/MT application already demonstrated)

- Collaboration with R&D department of Colfax International for

  tion studies

COLFAX
Customized Solutions

http://research.colfaxinternational.com

Check-pointing on MIC

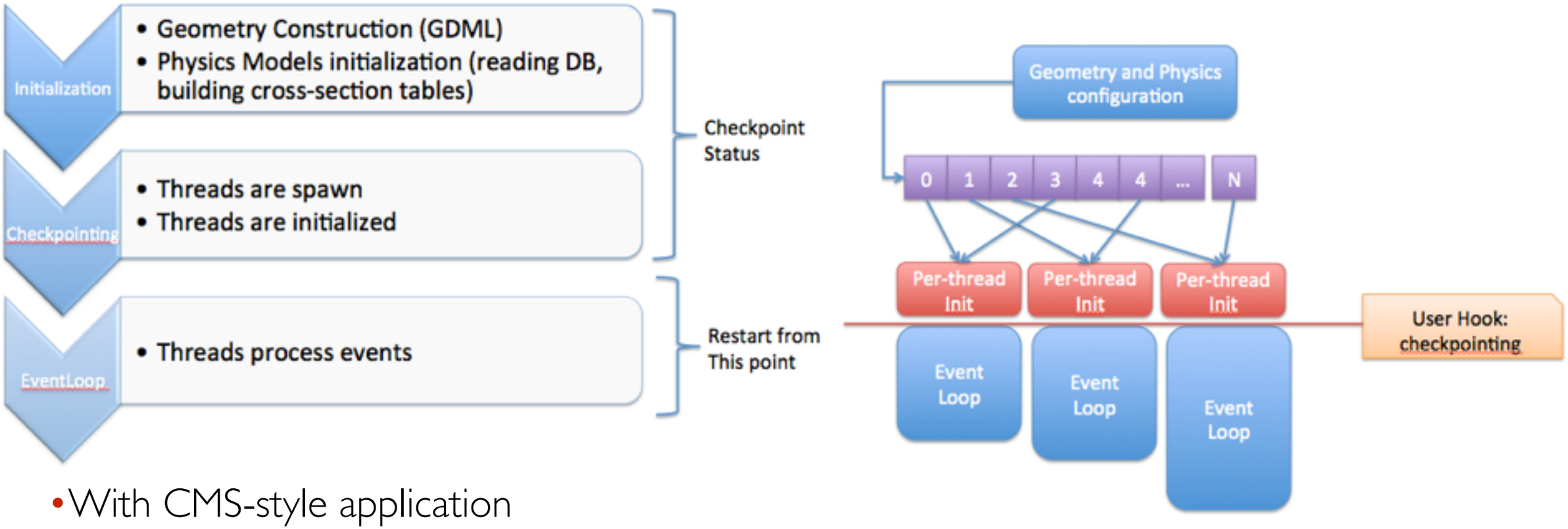
Use cases for checkpointing

SLAC

- **Allows for controlled dump of memory to file and "replay" of application from checkpoint file**
- Work done in collaboration with P. Elmer (CMS)
- Based on DMTCP (Distributed MultiThreaded CheckPointing) library from G. Cooperman et al.
- Available for MIC architectures

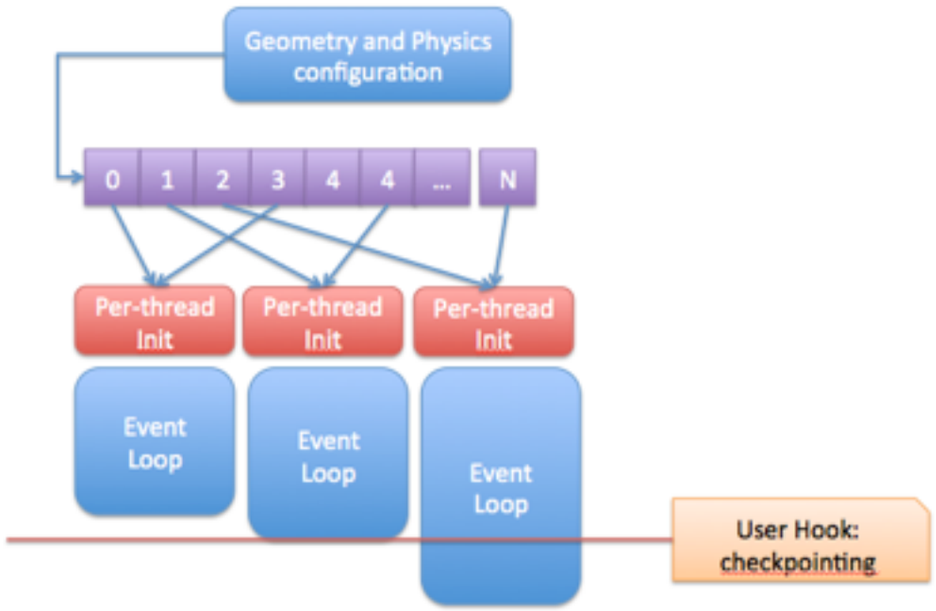
- Two scenarios tested:
  - Improve startup of job running on the MIC producing "images" just before first event (checkpoint at end of sequential phase)
  - Load balancing (stop job when number of threads drops below threshold)

Speeding up initialization



- With CMS-style application
- Checkpoint image preparation:
  - Initialization takes about 5 minutes
  - Checkpointing takes about 1 minute
  - Checkpoint image file size 1.4GB
- **Restart from checkpoint image file: <10 s**
- Checkpoint image can be distributed to other nodes and simulation process "cloned"
  - In real life applications one needs to re-seed RNG after checkpoint restart
- Note: virtual FS on card uses NFS, not yet studied its impact or alternative strategies (e.g. image compression)

Tails



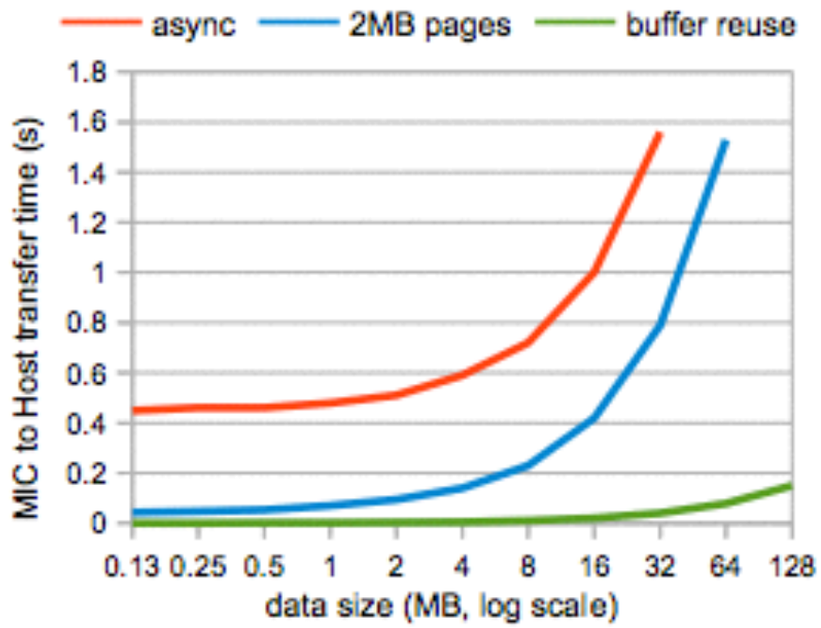
- Efficient use of cores (interesting for production systems)
- Scenario tested with success (reproducibility confirmed), however: final Geant4 Version 10.0 introduces load balancing between threads

- A simulation job will finish only when all events have been simulated
  - Some events may be longer than others
- Some threads may finish earlier than events processing "slow events"
  - For a fraction of execution time at the end of job life cycle: N active threads < N cores
- Checkpoint when N active threads drop below a given threashold and kill process
- Start a new job and reapeat
- When enough "checkpoint" have been collected start "tails" all together

G4-ATLAS on MIC

ATLAS G4 on Xeon Phi?

- Problem: X-Compile LCG, Athena, CINT dicts, etc.
  - Actually, that may resolve itself with next-gen Phi ...
- Problem: I/O from the card
  - Difficult to get performance
  - Want ROOT I/O for streaming
    - Reuse existing code
    - Is desired output format



- "N→1" I/O also for AthenaMP(I), EventService
  - But with different transport layers (e.g. MPI)

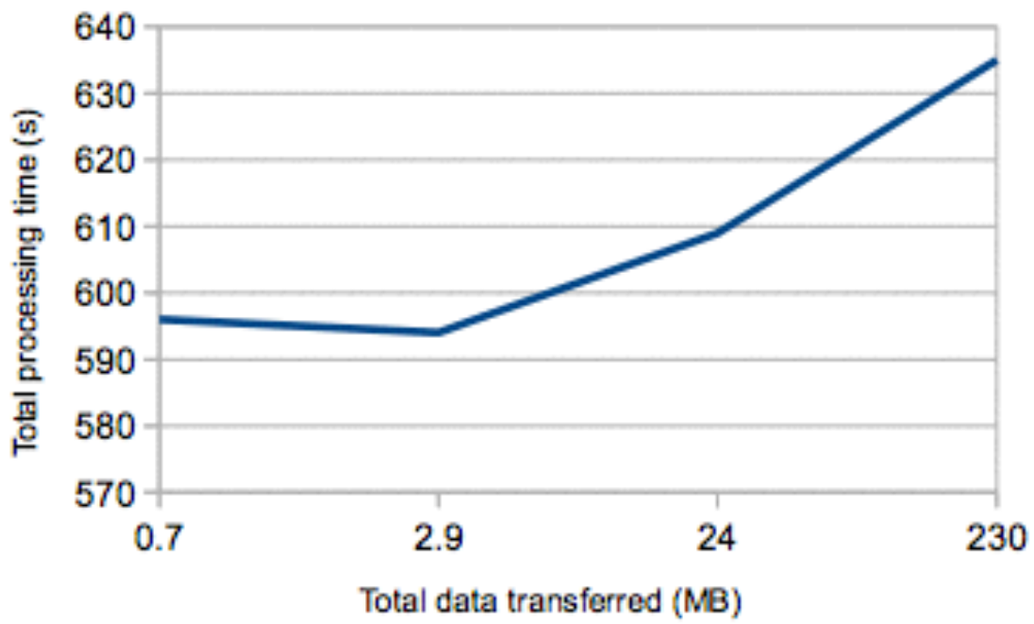
CRD TParallelMergingFile

- ROOT5 works in principle on Xeon Phi
  - Default “linuxx8664k1omicc” build is very minimal
  - I/O needs manual work (e.g. no rootcint, ACLiC)
- Test setup based on TParallelMergingFile
  - Each Phi process/thread has a memory based file
  - Sends buffers (TTree) to server process on host
  - Host merges buffers onto a single file
- Initial impressions: it works!
  - Some race condition in init on the server left
  - TSocket (UDP) connection has large overhead

Overhead

- Fill tree in 120 clients, sent to single server
  - Use a “sink” tree to equalize total CPU use on Phi
  - Max 30 simultaneous connections available
  - Processing on server is tiny (3% CPU, 0.1% Mem)
  - Processing on card is small (~10% CPU, 48% Mem)

- Overhead is clearly large
  - As seen also for offloads
- Next steps:
  - Detailed profiling ...
  - Customize transport



Conclusions

Summary of first round of experiments

SLAC

- First experience with Intel Xeon Phi promising:
  - When I/O is limited Geant4 Version 10.0 scales very well on card
- Absolute performances: 1 co-processor is equivalent to 1 CPU
  - In line with expectation from code not optimized for coprocessors
  - More optimizations are under studies
- Main goal reached: fit complex applications into memory limit with full occupancy of card and still be linear in speedup
- We see two benefits w/ Intel Xeon Phi:
  - Optimizations done for MIC also benefit "standard" jobs
  - Natural support hybrid simulations via MPI (one job on host one on card)
- Checkpointing may be the solution for speeding up initialization times (at least for production systems)

Acknowledgments

We would like to thank Openlab for providing hardware for Intel Xeon Phi and Intel Atom tests

We would like to thank CMS collaboration for providing hardware for ARM ODROID evaluation boards

We would like to thank T. LeCompte (ANL) for running the tests on Mira supercomputer

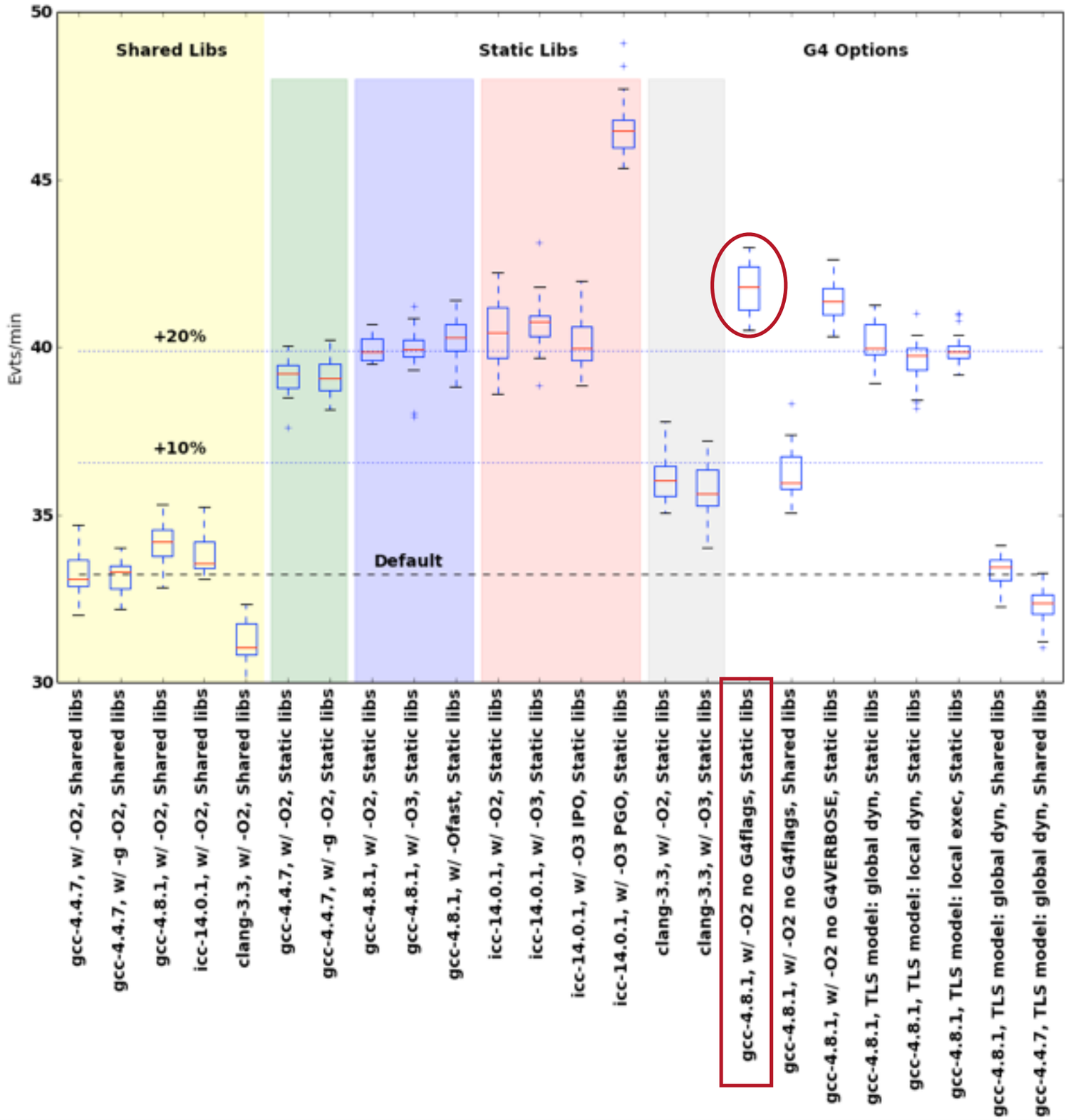
Backup

Comparison of compilation options/libraries

SLAC

Compilers
GCC 4.4.7
GCC 4.8.1
ICC 14.0.1
CLANG 3.3
G4 options

- Using to static libraries shows a 20% gain in performances
- Turning off some G4 options for productions also additional brings 6-7%

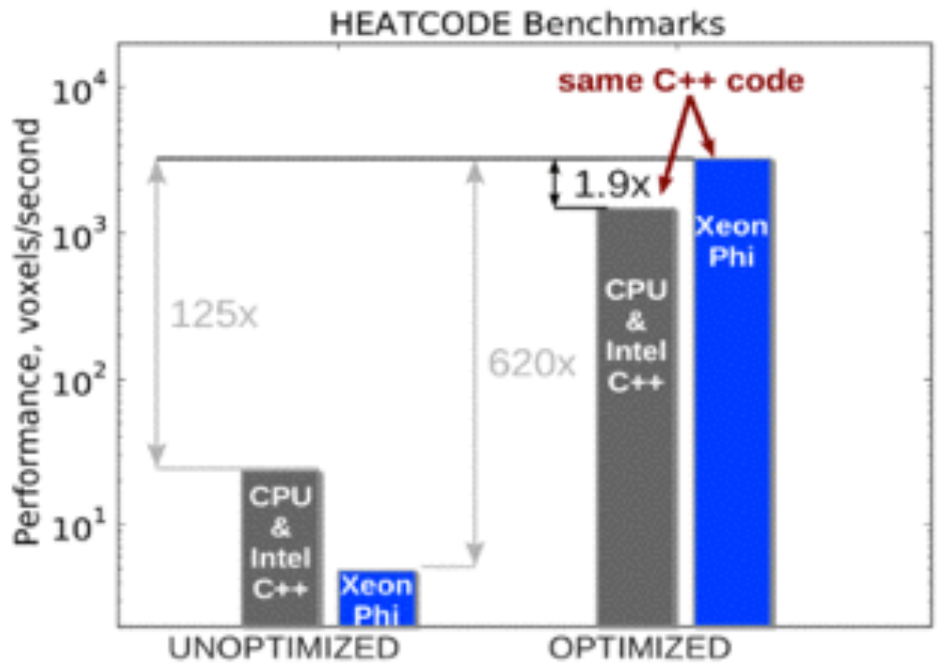


Code optimization for both CPU and coprocessor

SLAC

Performance Expectations: "Two Birds with One Stone"

- Performance will be disappointing if code is not optimized for multi-core CPUs
- Optimized code runs better on the MIC platform *and* on the multi-core CPU
- Single code for two platforms + Ease of porting = Incremental optimization



More information in case study on research.colfaxinternational.com

Dr. Andrey Vladimirov (Colfax International) – seminar at Stanford U.