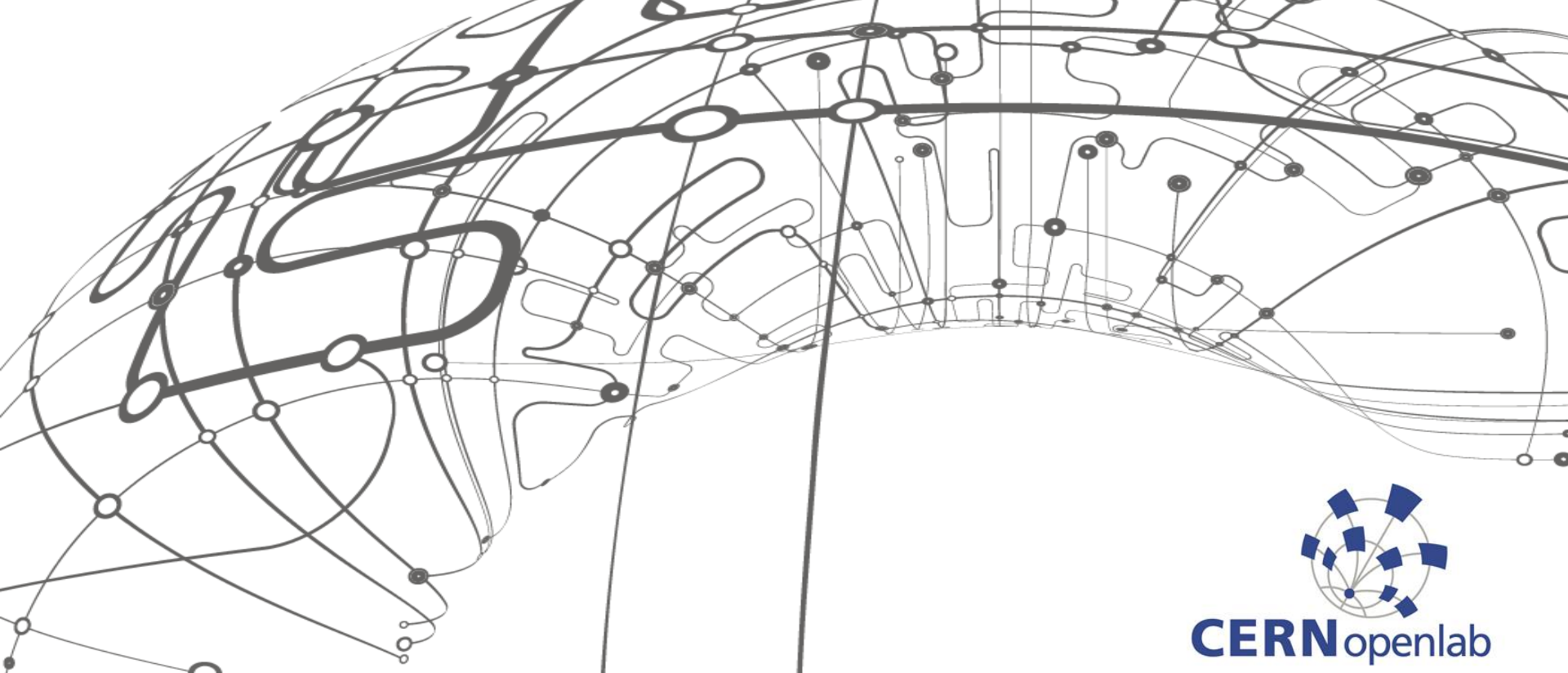


CERN openlab

Does the Intel Xeon Phi processor fit HEP workloads?

October 17th, CHEP 2013, Amsterdam

Andrzej Nowak, CERN openlab CTO office

On behalf of Georgios Bitzes, Havard Bjerke, Andrea Dotti, Alfio Lazzaro, Sverre Jarp, Pawel Szostek, Liviu Valsan, Mirela-Madalina Botezatu, Julien Leduc

CERN openlab

- CERN openlab is a framework for evaluating and integrating cutting-edge IT technologies or services in partnership with industry
- The Platform Competence Center (PCC) has worked closely with Intel for the past decade and focuses on:
  - many-core scalability
  - performance tuning and optimization
  - benchmarking and thermal optimization
  - teaching

Partners

Contributors

Associates

Outline

- A brief history of openlab involvement with the Intel MIC project
- Architecture refresher
- HEP benchmarks
- Results
- Conclusions and projections

CERN openlab

Larrabee: A Many-Core x86 Architecture for Visual Computing

Larry Seiler[1], Doug Carmean[1], Eric Sprangle[1], Tom Forsyth[1], Michael Abrash[2],
Pradeep Dubey[1], Stephen Junkins[1], Adam Lake[1], Jeremy Sugerman[3],
Robert Cavin[1], Roger Espasa[1], Ed Grochowski[1], Toni Juan[1], and Pat Hanrahan[3]

Abstract

This paper presents a many-core visual computing architecture code named Larrabee, a new software rendering pipeline, a many-core programming model, and performance analysis for several applications. Larrabee uses multiple in-order x86 CPU cores that are augmented by a wide vector processor unit, as well as some fixed function logic blocks. This provides dramatically higher performance per watt and per unit of area than out-of-order CPUs on highly parallel workloads. It also greatly increases the flexibility and programmability of the architecture as compared to standard GPUs. A coherent on-die 2nd level cache allows efficient inter-processor communication and high-bandwidth local data access by CPU cores. Task scheduling is performed entirely with software in Larrabee, rather than in fixed function logic. The customizable software graphics rendering pipeline for this architecture uses binning in order to reduce required memory bandwidth, minimize lock contention, and increase opportunities for parallelism relative to standard GPUs. The Larrabee native programming model supports a variety of highly parallel applications that use irregular data structures. Performance analysis on those applications demonstrates Larrabee's potential for a broad range of parallel computation.

CCS: I.3.1 [Computer Graphics]: Hardware Architecture--Graphics Processors, Parallel Processing; I.3.3 [Computer Graphics]: Picture/Image Generation--Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism--Color, shading, shadowing, and texture

Keywords: graphics architecture, many-core computing, real-time graphics, software rendering, throughput computing, visual computing, parallel processing, SIMD, GPGPU.

1. Introduction

Modern GPUs are increasingly programmable in order to support advanced graphics algorithms and other parallel applications. However, general purpose programmability of the graphics pipeline is restricted by limitations on the memory model and by fixed function blocks that schedule the parallel threads of execution. For example, pixel processing order is controlled by the rasterization logic and other dedicated scheduling logic.

This paper describes a highly parallel architecture that makes the rendering pipeline completely programmable. The Larrabee architecture is based on in-order CPU cores that run an extended version of the x86 instruction set, including wide vector processing operations and some specialized scalar instructions. Figure 1 shows a schematic illustration of the architecture. The cores each access their own subset of a coherent L2 cache to provide high-bandwidth L2 cache access from each core and to simplify data sharing and synchronization.

Larrabee is more flexible than current GPUs. Its CPU-like x86-based architecture supports subroutines and page faulting. Some operations that GPUs traditionally perform with fixed function logic, such as rasterization and post-shader blending, are performed entirely in software in Larrabee. Like GPUs, Larrabee uses fixed function logic for texture filtering, but the cores assist the fixed function logic, e.g. by supporting page faults.

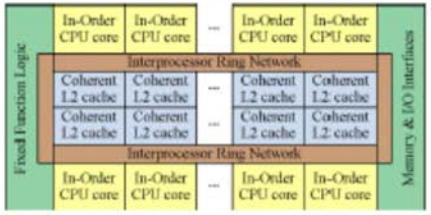


Figure 1: Schematic of the Larrabee many-core architecture: The number of CPU cores and the number and type of co-processors and I/O blocks are implementation-dependent, as are the positions of the CPU and non-CPU blocks on the chip.

This paper also describes a software rendering pipeline that runs efficiently on this architecture. It uses binning to increase parallelism and reduce memory bandwidth, while avoiding the problems of some previous tile-based architectures. Implementing the renderer in software allows existing features to be optimized based on workload and allows new features to be added. For example, programmable blending and order-independent transparency fit easily into the Larrabee software pipeline.

Finally, this paper describes a programming model that supports more general parallel applications, such as image processing, physical simulation, and medical & financial analytics. Larrabee's support for irregular data structures and its scatter-gather capability make it suitable for these throughput applications as demonstrated by our scalability and performance analysis.

[1] Intel® Corporation: larry.seiler, doug.carmean, eric.sprangle, tom.forsyth, pradeep.dubey, stephen.junkins, adam.t.lake, robert.d.cavin, roger.espasa, edward.grochowski & toni.juan @intel.com

[2] RAD Game Tools: mikea@radgametools.com

[3] Stanford University: yoel & hanrahan @cs.stanford.edu

ACM Reference Format
Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P. 2008. Larrabee: A Many-Core x86 Architecture for Visual Computing. ACM Trans. Graph. 27, 3, Article 18 (August 2008), 15 pages. DOI = 10.1145/1360612.1360617 http://doi.acm.org/10.1145/1360612.1360617.

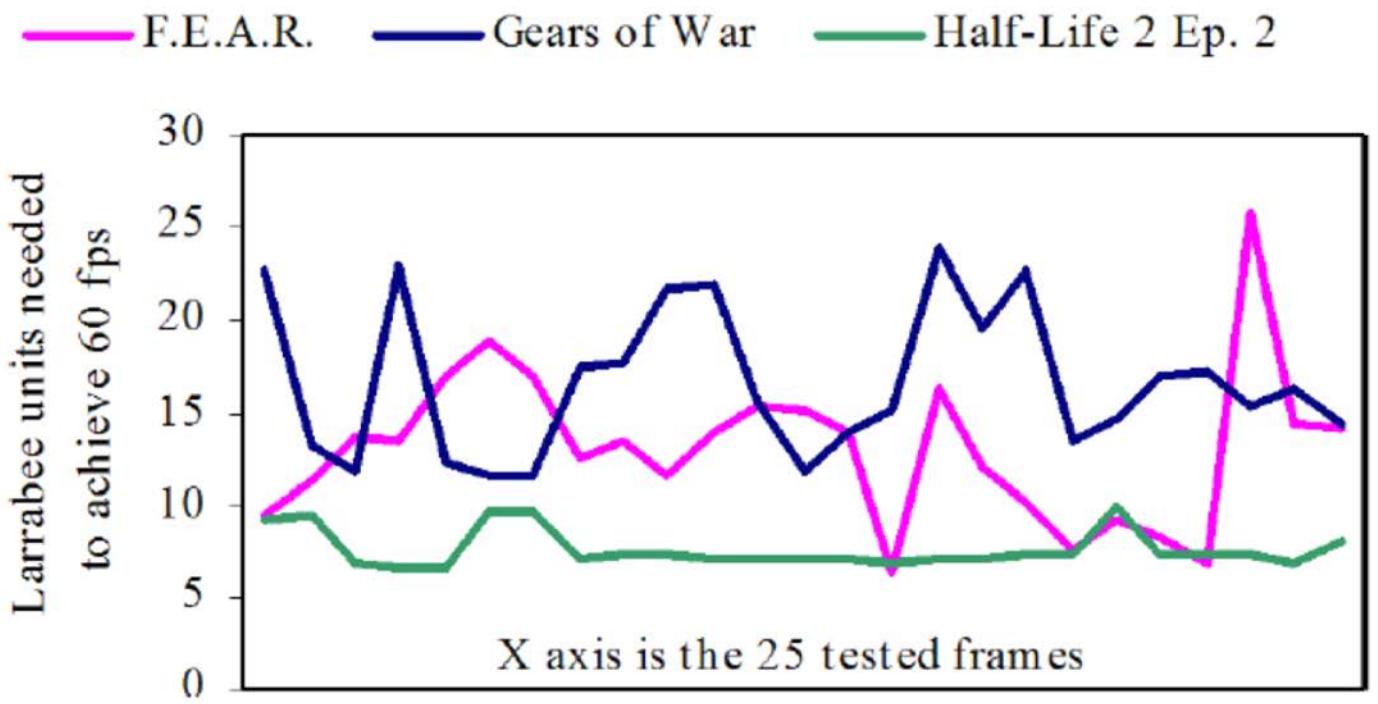


***Figure 10**: Overall performance: shows the number of Larrabee Units (cores running at 1 GHz) needed to achieve 60fps for of the series of sample frames in each game.*

Brief history of Intel MIC at openlab

Early access

- Work since MIC alpha (under RS-NDA)
- ISA reviews in 2008

Results

- 3 benchmarks ported from Xeon and delivering results: ROOT, Geant4, ALICE HLT trackfitter

Expertise

- Understood and compared with Xeon
- Post-launch dissemination

Architecture refresher (1)

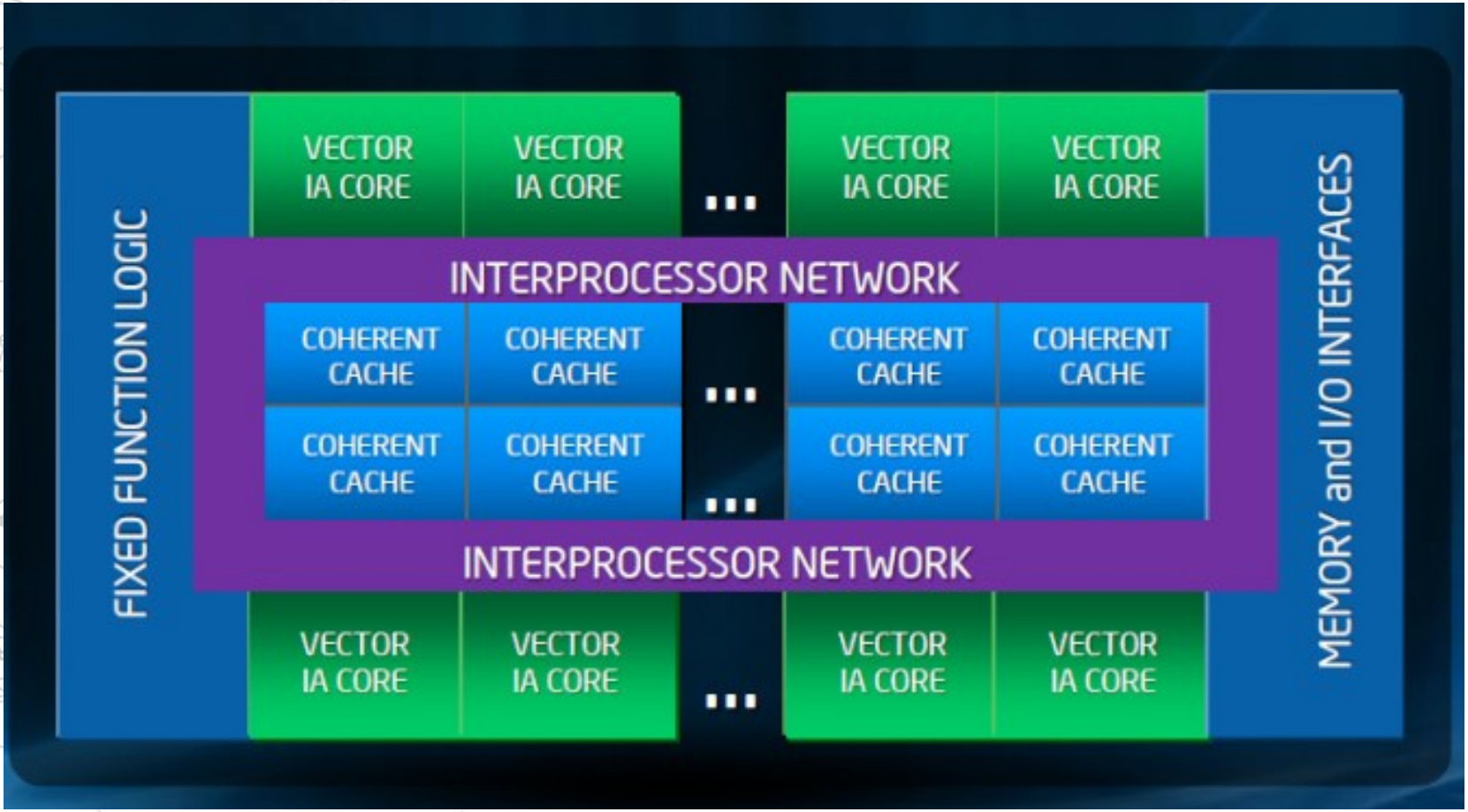


Image: Intel

Architecture refresher (2)

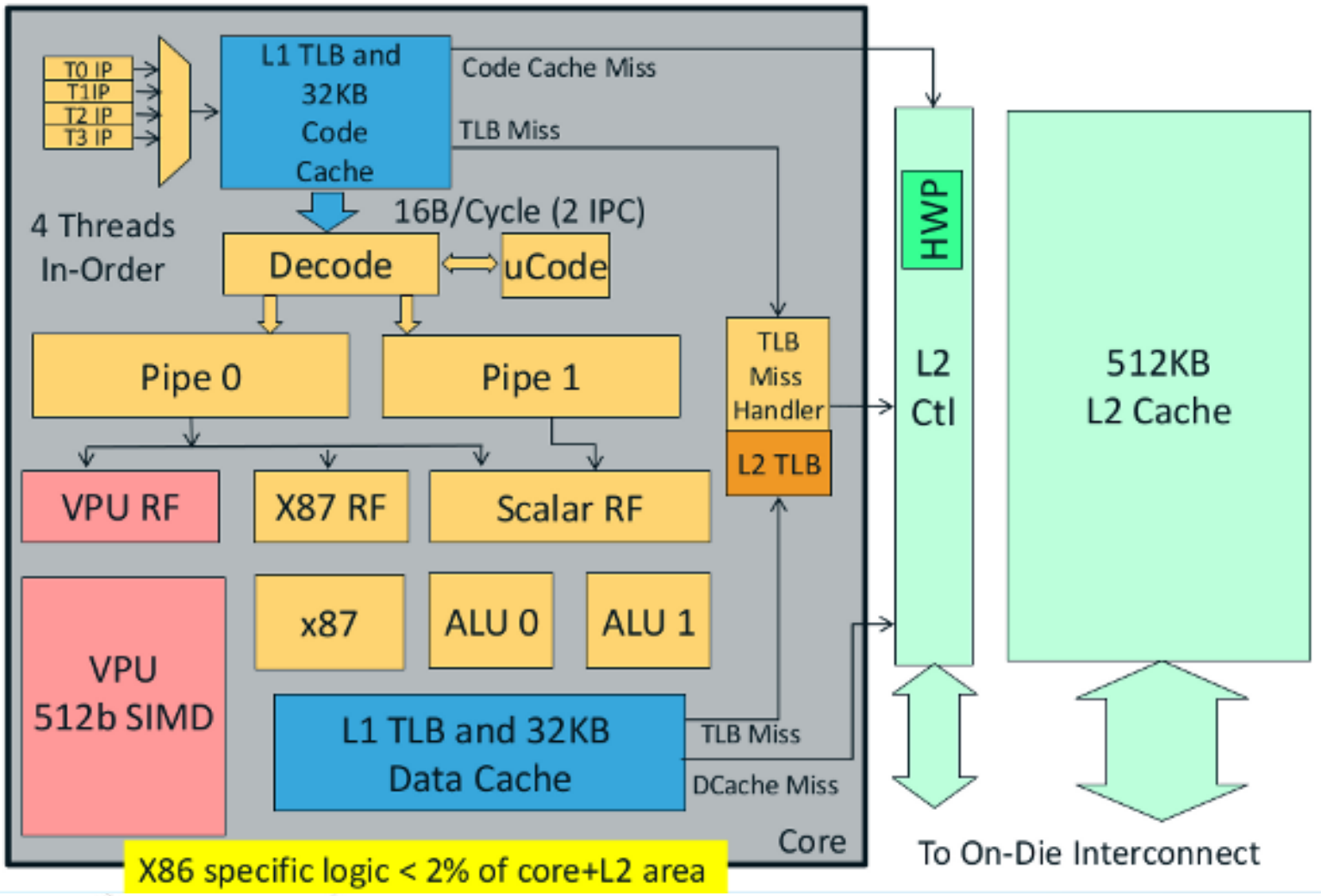


Image: semiaccurate.com / Intel

Architecture refresher (3)

- PCIe card form factor, “SMP machine on a chip”
  - Linux OS on board
  - PCIe power envelope - ~200-300W
  - Limited on-board memory (16GB total limited by GDDR)
  - ~60 cores @ ~1 GHz
- x86 architecture
  - 64-bit
  - P54C core: In-order, Superscalar
  - Shared coherent cache (~256-512k L2/core – KNF/KNC)
  - New ISA with new vector instructions
- Floating point support through vector units
  - 512-bit wide, FMA
- 4-wide hardware threading
  - >200 threads in total
- 1 TFLOP DP
  - Still with the programmability of a Xeon?

HEP software today

- Very limited or no vectorization
  - Online has somewhat better conditions to vectorize
- Sub-optimal instruction level parallelism (CPI at >1)
- Hardware threading often beneficial
- Cores used well through multiprocessing – bar the stiff memory requirements
  - However, systems put in production with delays
- Sockets used well
- Multiple systems used very well
- Relying on in-core improvements and # cores for scaling

Benchmarks

- Standard: HEPSPEC06 (partial test)
- Need to look forward to prototype workloads
- Analysis: MLFit
  - Threaded (pthreads, MPI, OpenMP, TBB)
  - Vectorized (Cilk+)
- Simulation: Next-gen multi-threaded Geant4 prototype
  - Threaded “FullCMS” example
  - No explicit vectorization
- Online: New ALICE/CBM track fitter prototype
  - Threaded (here with OpenMP)
  - Vectorized with Vc
- ICC 14.0 and pinning used unless specified otherwise

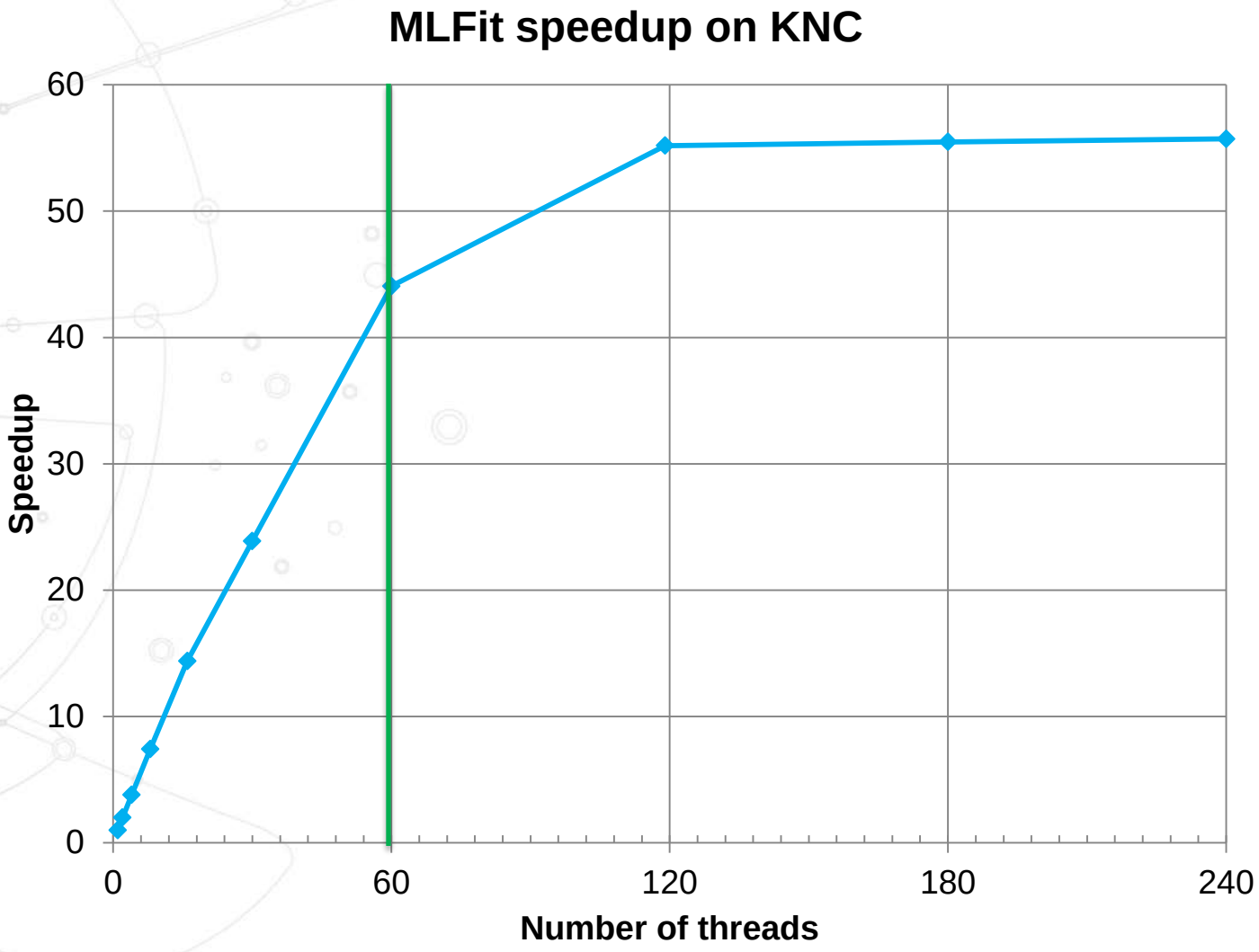
Porting effort

	LOC	1st port time	New ports	Tuning
TF	< 1’000	days	N/A	2 weeks
MLFit	3’000	< 1 day	< 1 day	weeks
MTG	2’000’000	1 month	< 1 day	< 1 week

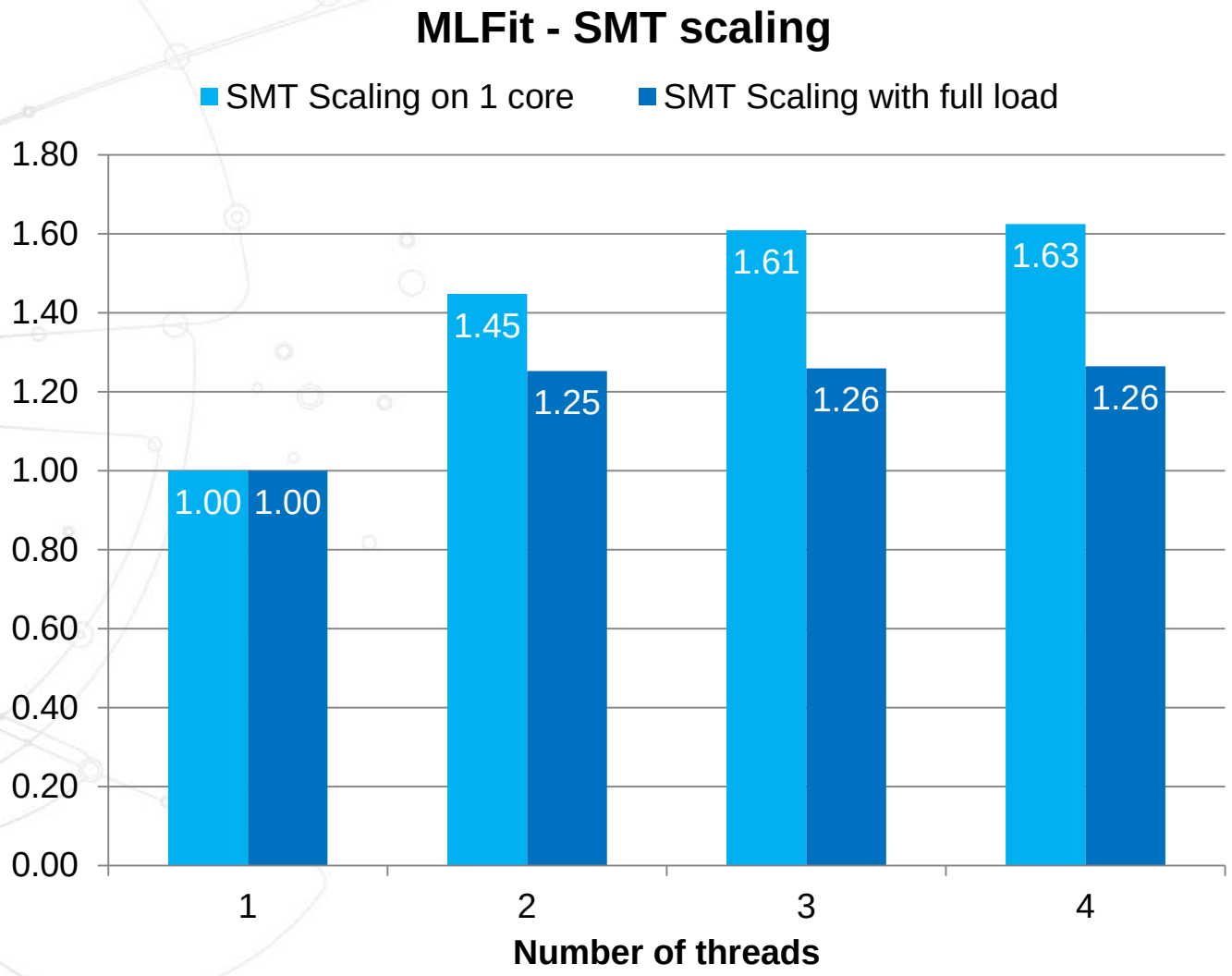
HEPSPEC06 results and extrapolation

- HEPSPEC06 represents our current family of workloads, not optimized for next-gen hardware
- Soplex would not finish properly
  - Ran out of time to investigate
- 32 cores: 57.7 HS06
- Extrapolating to 61 cores: 110 HS06
- SMT scalability:
  - 1.8 / thread @ 1 core
  - 3.48 / 4 threads @ core
- SMT under full load scales differently
  - Using factor from MTG4 - ~70%
- Expected throughput for 244 threads: ~190
- Reference IVB-EP w/ 48 threads: > 450

MLFit

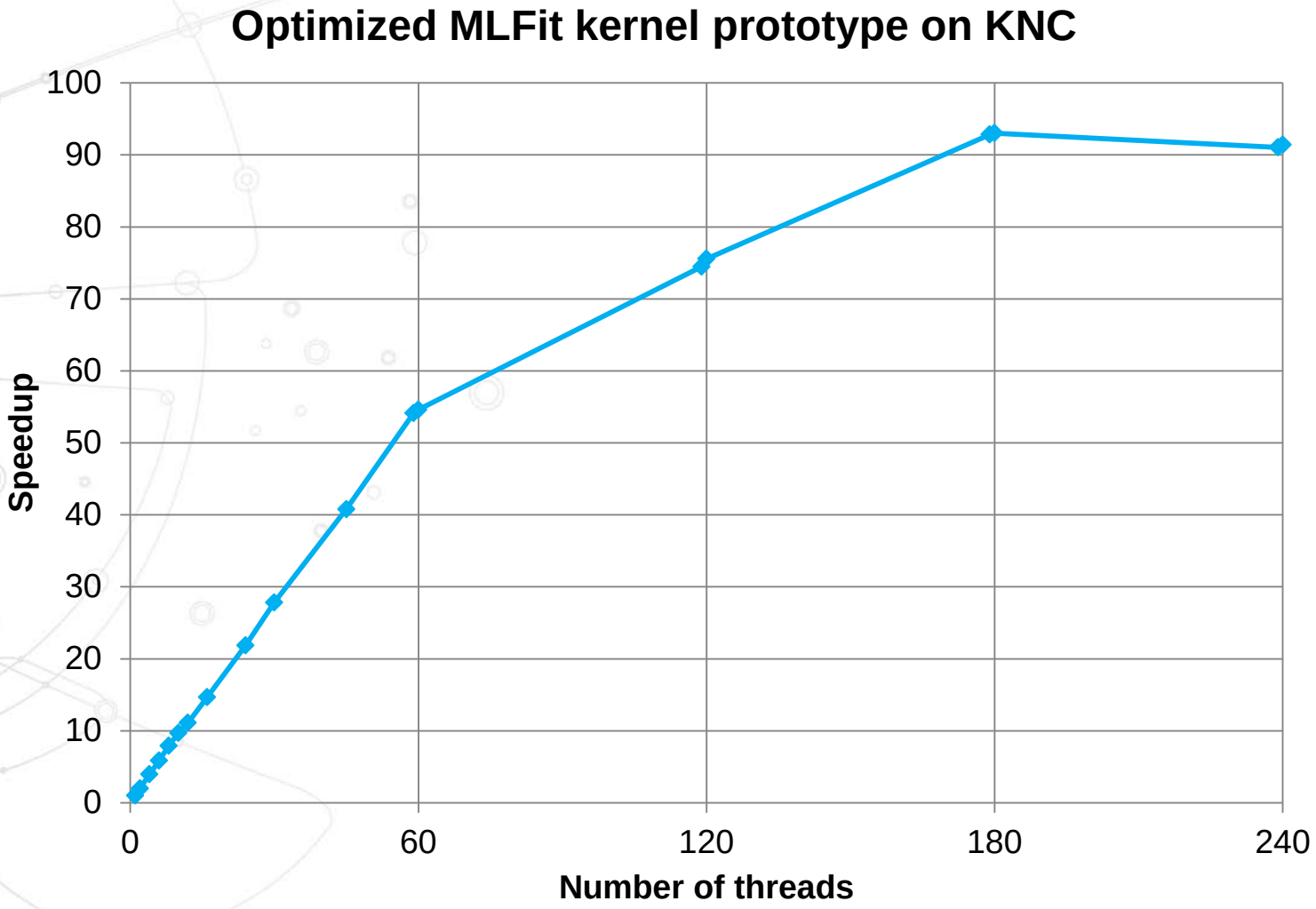

MLFit speedup on KNC
Speedup
0
10
20
30
40
50
60
0
60
120
180
240
Number of threads

MLFit – SMT scaling / BW saturation



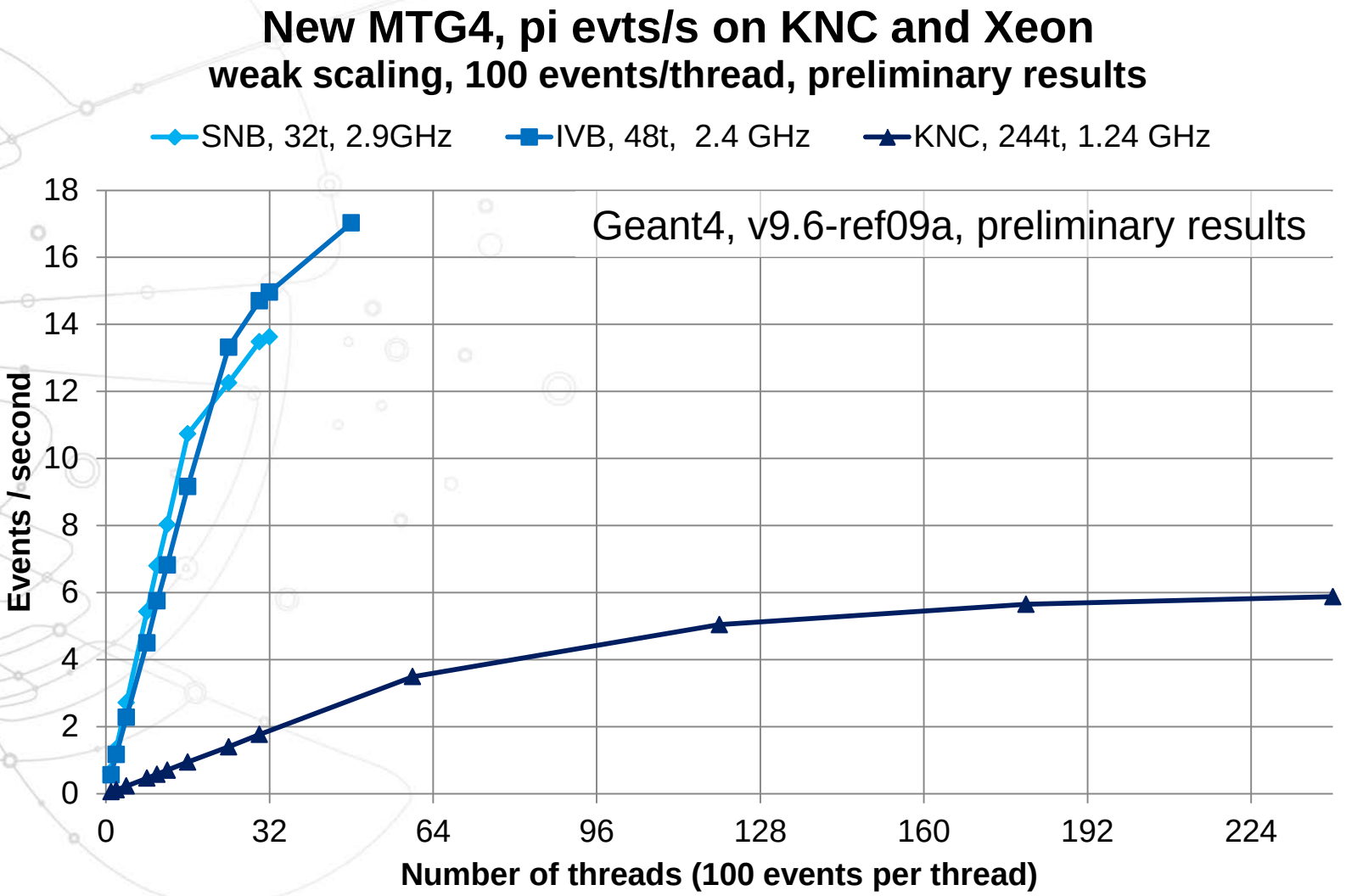
MLFit kernel prototype

Code courtesy of Vincenzo Innocente, CERN



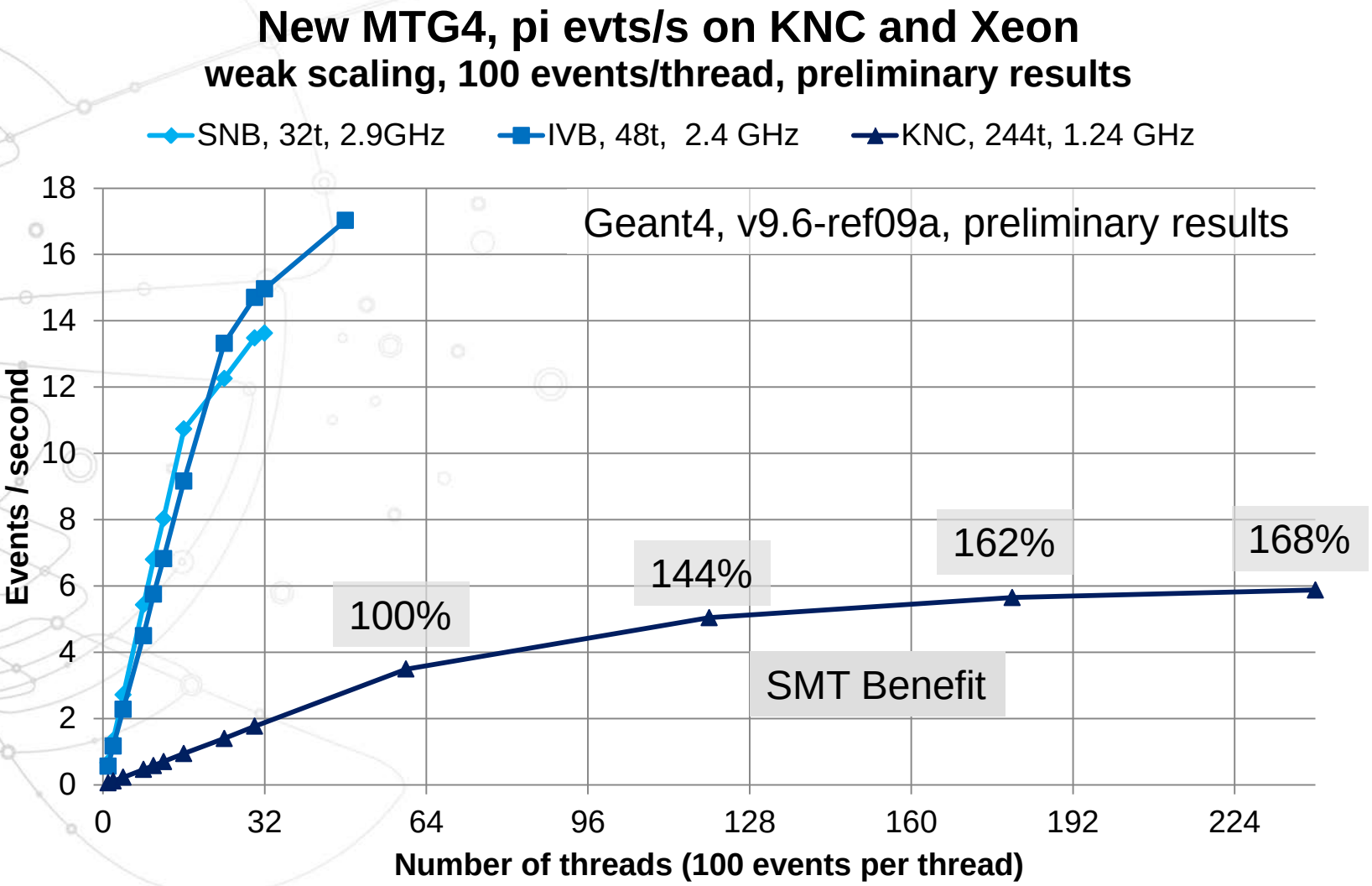
New multi-threaded Geant4 prototype

preliminary results; code courtesy of Andrea Dotti + G4 team



New multi-threaded Geant4 prototype

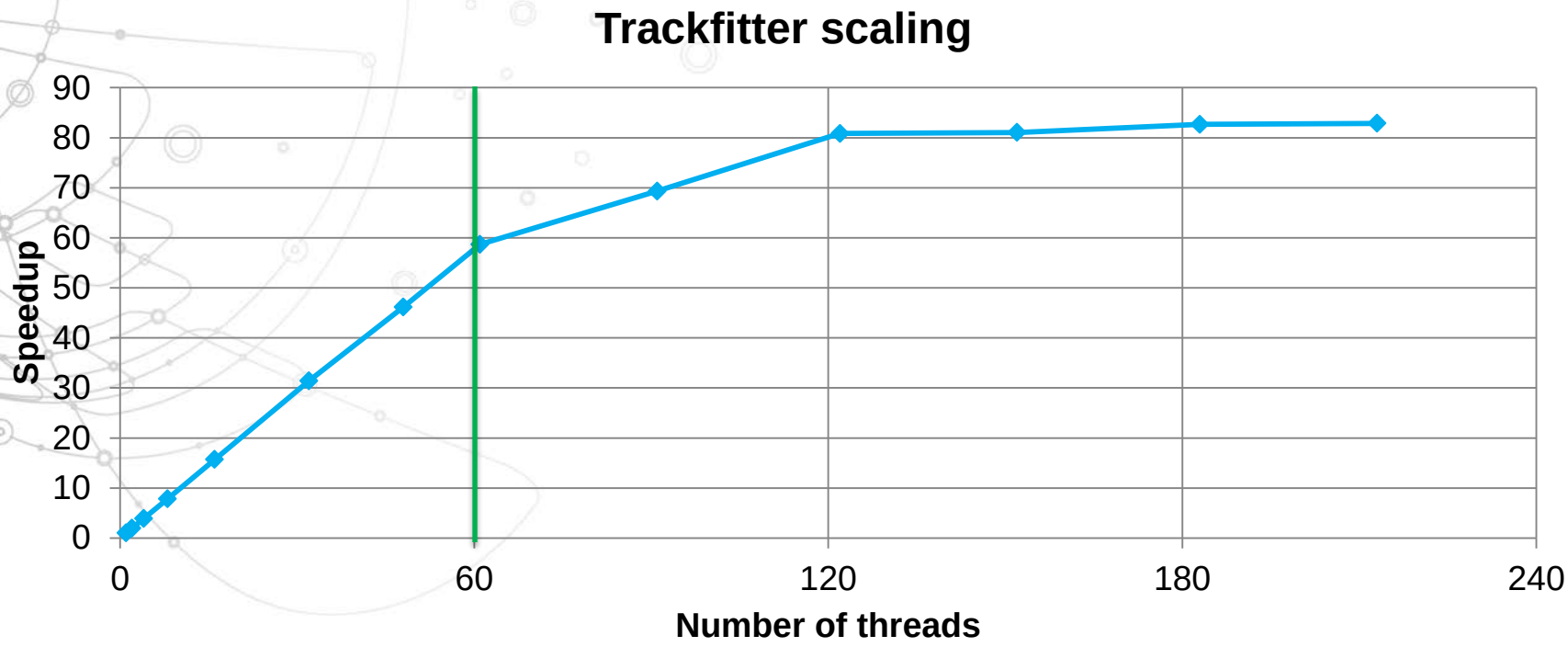
preliminary results; code courtesy of Andrea Dotti + G4 team



The new ALICE/CBM Trackfitter

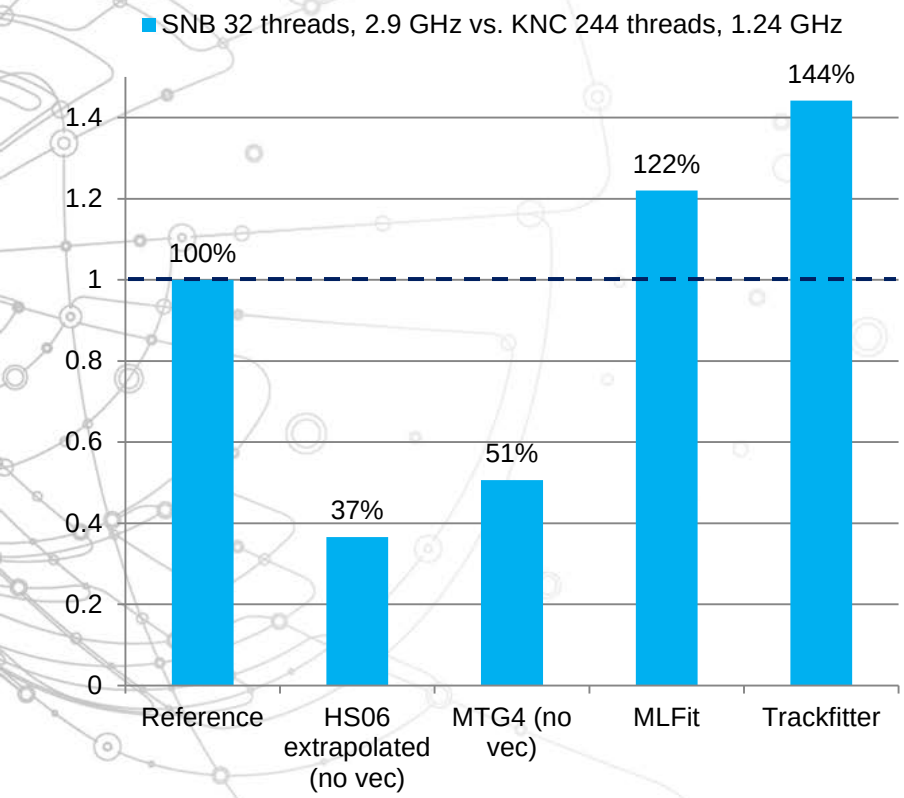
preliminary results

x87	singleVc (vec)	OpenMP 213 threads (max perf)	DP scalar to SP vector speedup	OpenMP to SP vector speedup	OpenMP vec to x87 speedup
11.587	0.811	0.0098	14.2x	82.7x	1182x

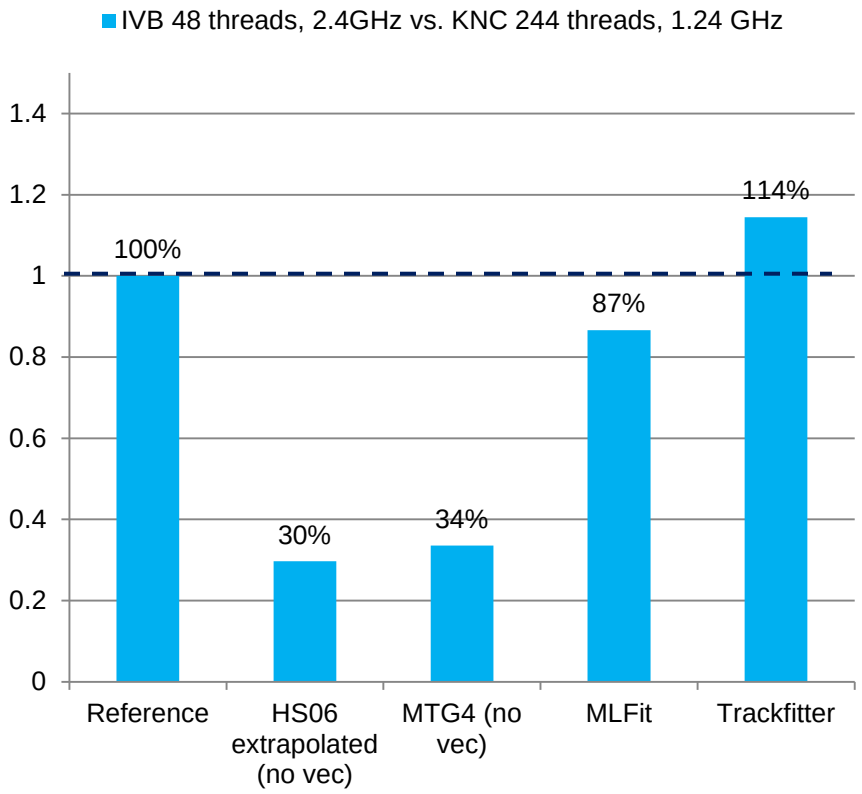


Xeon comparison

SNB-EP vs. KNC throughput



IVB-EP vs. KNC throughput



Higher is better

No manual MIC-specific optimizations applied

Porting and software - conclusions

- Think in multiple dimensions of performance
  - Vectorization, threading, porting to ICC – can and should be done independently, on Xeon
- Optimization
  - Compiler
  - Tuning – threading and performance tools exist, so do many runtimes
  - Math usage (different implementation!)
  - Memory – 16 GB limit today
- Build systems – the on-card OS is “simpler” Linux, with some, not all, OSS addons

VECTORIZATION

(hard, but not always impossible)

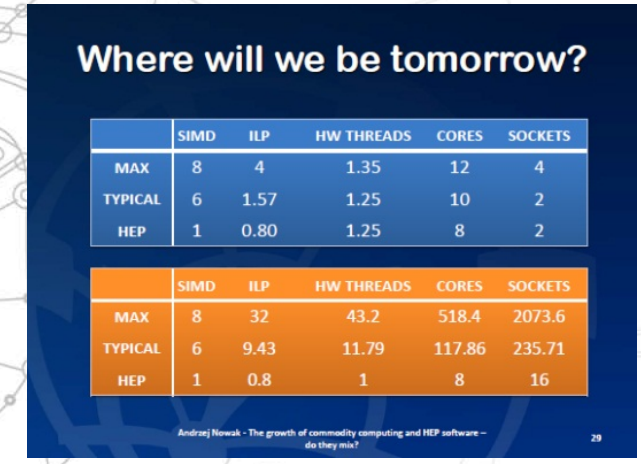
Performance - conclusions

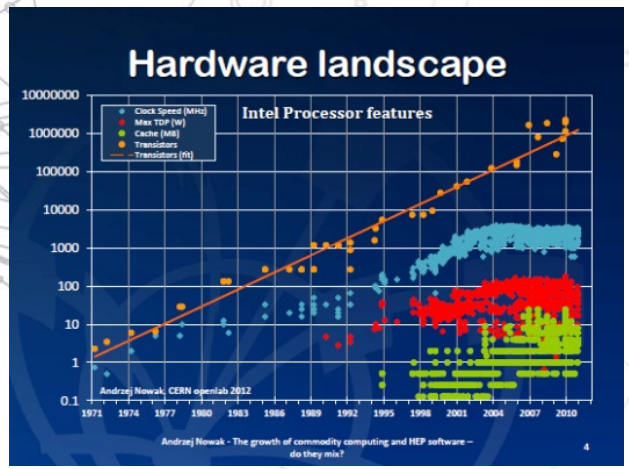
- Optimized applications surpass dual-socket Xeon performance
- Non-optimized performance reaches approximately a single server socket
- We don’t make use of optimized bandwidth
- Control over math function usage and performance is key vis a vis Xeon
- Stack (including compiler) maturity is important and improving
  - SMT benefit for 1, 2, 3, 4 HW threads changed over time

Were we of any help?

- Pre-silicon feedback (Geant4) -> arch. behavior
- System connectivity -> full system
- System integration -> ongoing (KNL)
- Comments on general OS -> Linux
- Math function usage -> better compilers and guidelines
- Documentation -> improved
- Benchmarks -> delivered
- Testimonials -> delivered
- Comments on stack -> ongoing (OSS)
- Many more…

Looking ahead

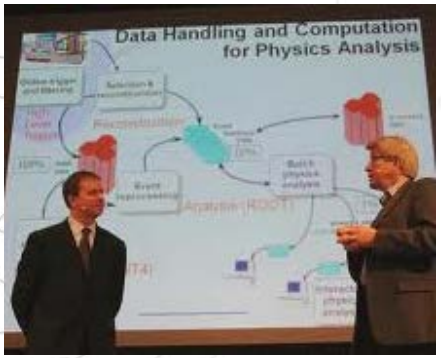




- KNL: 14nm stand-alone or PCI, integrated memory
- Let's do some math
  - Stampede: 8 out of 10 PF from MIC
  - Upgrade to KNL: 15 PF
  - = 50-80% improvement over KNC
- ISA convergence between KNL and Xeon?
- New platform connectivity options
- Heterogeneity
- The future of accessible performance

Thank you

(Q & A)



Andrzej.Nowak@cern.ch

Credits go to Georgios Bitzes, Havard Bjerke, Andrea Dotti, Alfio Lazzaro, Sverre Jarp, Pawel Szostek, Liviu Valsan, Mirela-Madalina Botezatu, Julien Leduc and many others. Special thanks to Intel.