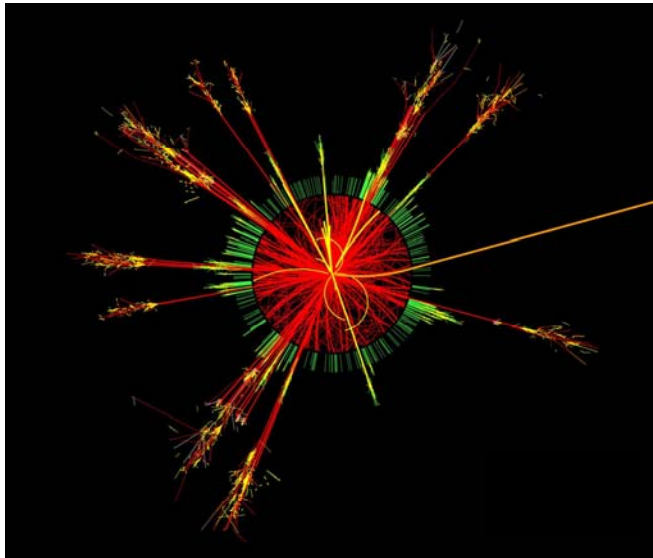


CHEP 2010

How to harness the performance potential of current Multi-Core CPUs and GPUs

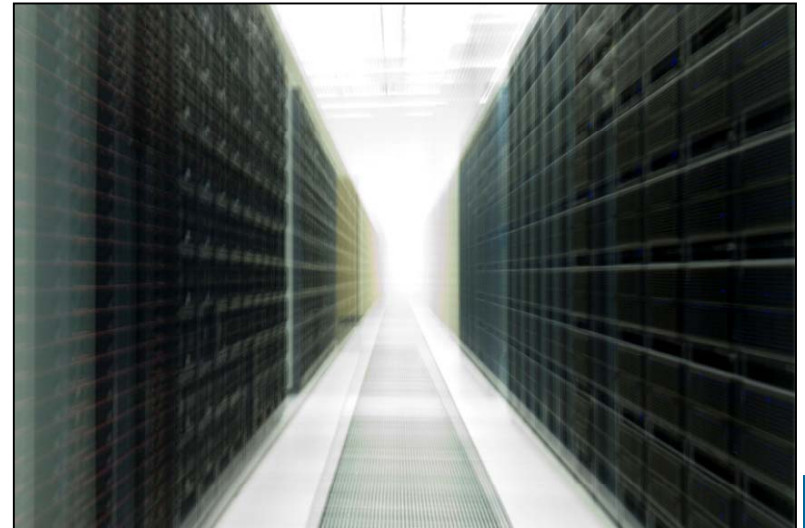


Sverre Jarp

CERN
openlab

IT Dept.

CERN



Taipei, Monday 18 October 2010

Contents

- **The hardware situation**
- **Current software**
- **Software prototypes**
- **Some recommendations**
- **Conclusions**

The hardware situation

In the days of the Pentium

- **Life was really simple:**

- Basically two dimensions
 - The frequency of the pipeline
 - The number of boxes
- The semiconductor industry increased the frequency
- We acquired the right number of (single-socket) boxes

Pipeline

Superscalar

Nodes

Sockets

Today: Seven dimensions of multiplicative performance

■ First three dimensions:

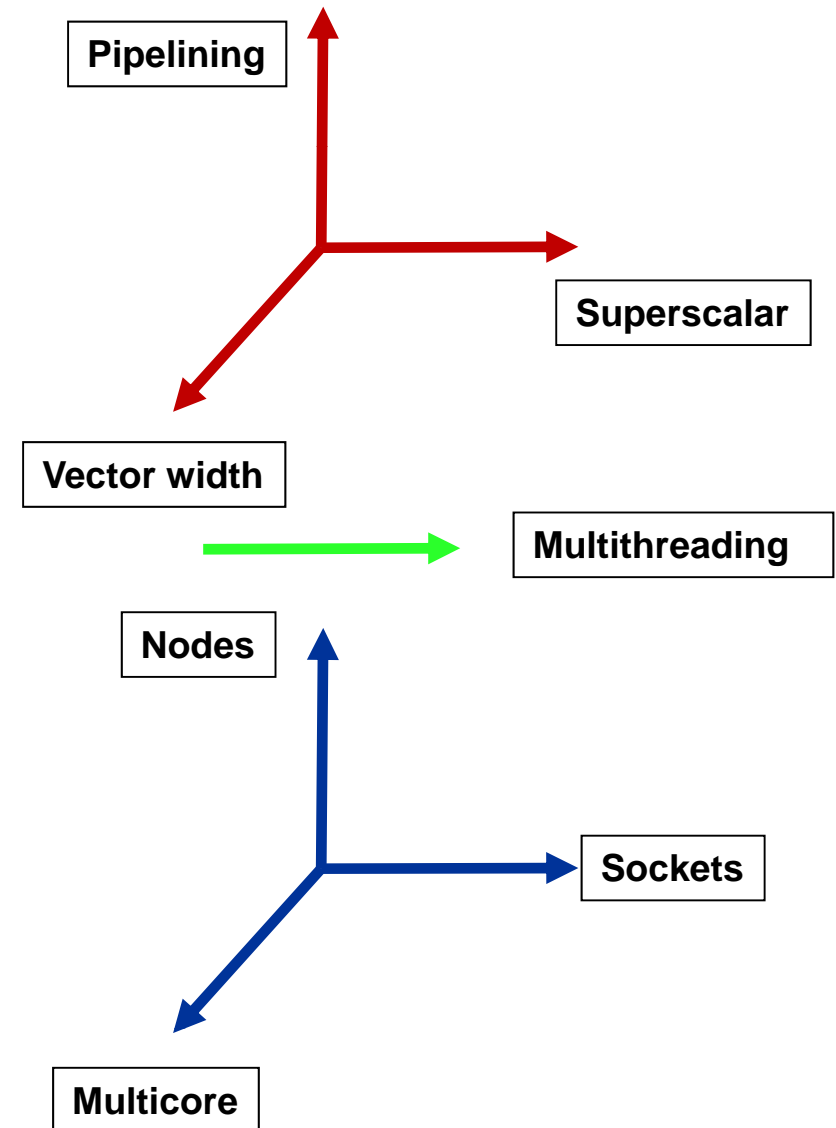
- Pipelined execution units
- Large superscalar design
- Wide vector width (SIMD)

■ Next dimension is a “pseudo” dimension:

- Hardware multithreading

■ Last three dimensions:

- Multiple cores
- Multiple sockets
- Multiple compute nodes

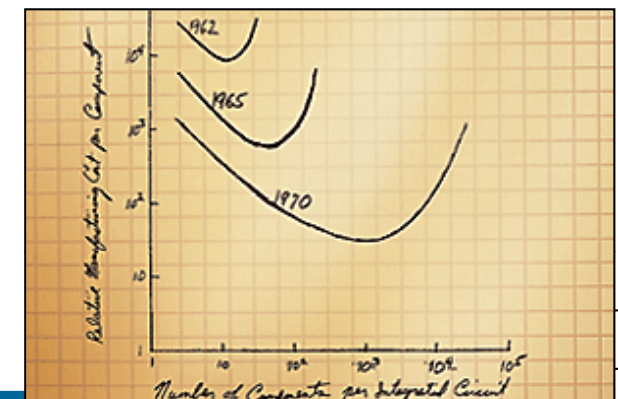
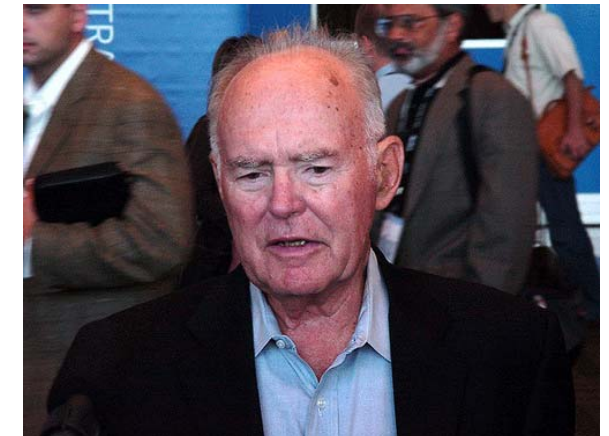
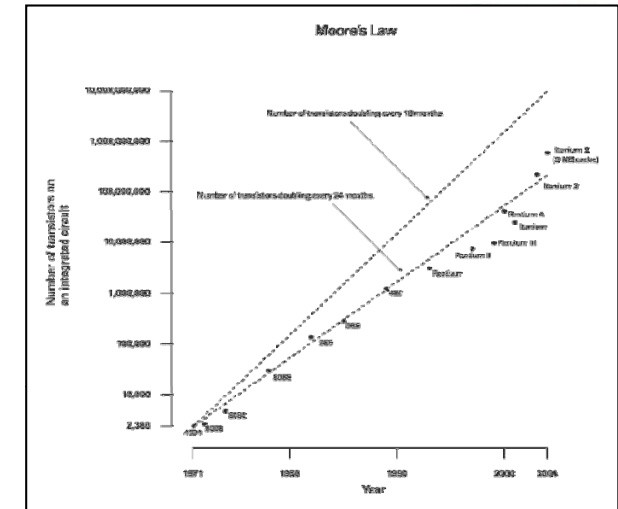




Moore's law

- We continue to double the number of transistors every other year
 - The consequences
 - CPUs
 - Single core → Multicore → Manycore
 - Vectors
 - Hardware threading
 - GPUs
 - Huge number of FMA units

- Today we commonly acquire chips with 1'000'000'000 transistors!



Real consequence of Moore's law

- We are being “**drowned**” in transistors:
 - More (and more complex) execution units
 - Hundreds of new instructions
 - Longer SIMD vectors
 - Large number of cores
 - More hardware threading
- In order to profit we need to “think parallel”
 - Data parallelism
 - Task parallelism

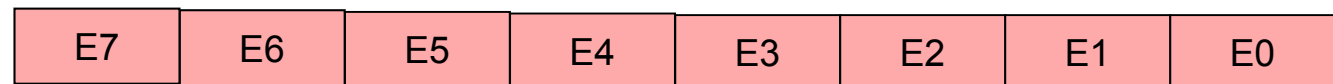
Four floating-point data flavours (256b)

- **Longer vectors:**

- AVX (Advanced Vector eXtension) is coming:
 - As of next year, vectors will be 256 bits in length
 - Intel's "Sandy Bridge" first (others are coming, also from AMD)

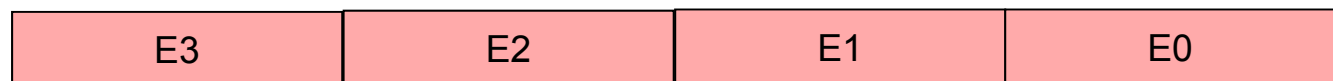
- **Single precision**

- Scalar single (SS)
- Packed single (PS)



- **Double precision**

- Scalar Double (SD)
- Packed Double (PD)



**Without vectors in our software, we will use
1/4 or 1/8 of the available execution width**

The move to many-core systems

- **Examples of “CPU slots”: Sockets * Cores * HW-threads**

- Basically what you observe in “`cat /proc/cpuinfo`”

- **Conservative:**

- Dual-socket AMD six-core (Istanbul): $2 * 6 * 1 = 12$
- Dual-socket Intel six-core (Westmere): $2 * 6 * 2 = 24$

- **Aggressive:**

- Quad-socket AMD Magny-Cours (12-core) $4 * 12 * 1 = 48$
- Quad-socket Nehalem-EX “octo-core”: $4 * 8 * 2 = 64$

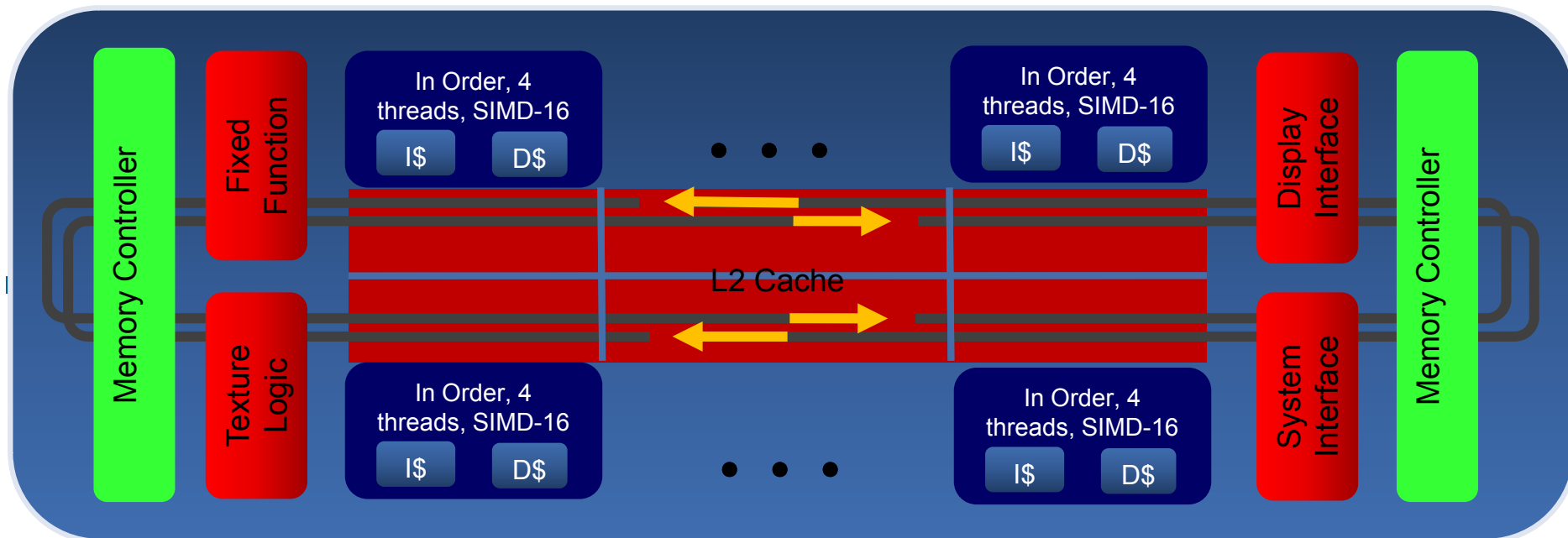
- **In the near future: Hundreds of CPU slots !**

- Quad-socket Sun Niagara (T3) processors w/16 cores and 8 threads (each): $4 * 16 * 8 = 512$

- **And, by the time new software is ready: Thousands !!**

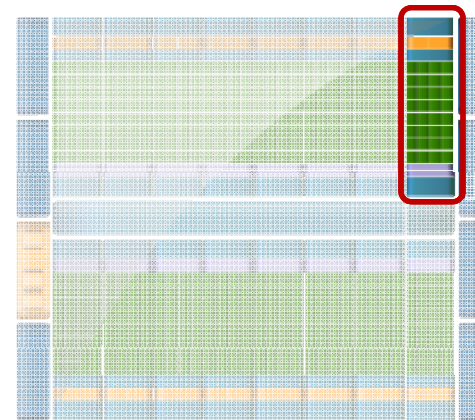
Accelerators (1): Intel MIC

- **Many Integrated Core architecture:**
 - Announced at ISC10 (June 2010)
 - Based on the x86 architecture, 22nm (in 2012?)
 - Many-core (> 50 cores) + 4-way multithreaded + **512-bit vector unit**
 - **Limited memory: Few Gigabytes**

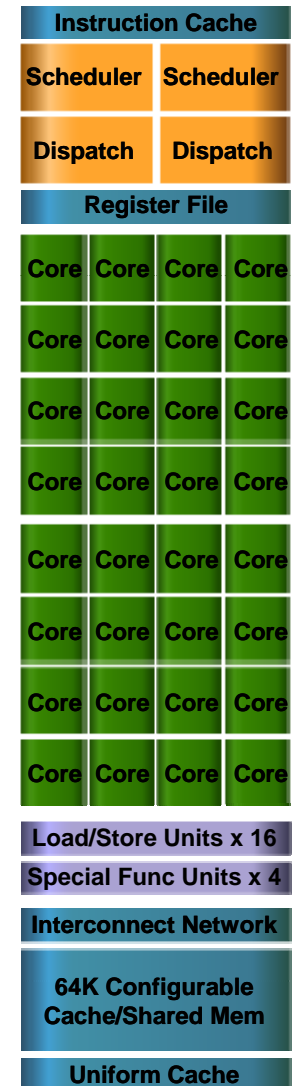


Accelerators (2): Nvidia Fermi GPU

- **Streaming Multiprocessing (SM) Architecture**
- 32 “CUDA cores” per SM (512 total)
- Peak single precision floating point performance (at 1.15 GHz”:
 - Above 1 Tflop
- **Double-precision: 50%**
- **Dual Thread Scheduler**
- 64 KB of RAM for shared memory and L1 cache (configurable)
- A few Gigabytes of main memory



Lots of interest in the HEP on-line community



Adapted from Nvidia

Current software

SW performance: A complicated story!

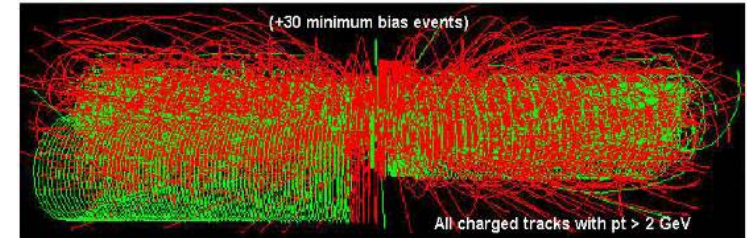
- **We start with a concrete, real-life problem to solve**
 - For instance, simulate the passage of elementary particles through matter
- **We write programs in high level languages**
 - C++, JAVA, Python, etc.
- **A compiler (or an interpreter) transforms the high-level code to machine-level code**
- **We link in external libraries**
- **A sophisticated processor with a complex architecture and even more complex micro-architecture executes the code**
- **In most cases, we have little clue as to the efficiency of this transformation process**

We need forward scalability

- **Not only should a program be written in such a way that it extracts maximum performance from today's hardware**
- **On future processors, performance should scale automatically**
 - In the worst case, one would have to recompile or relink
- **Additional CPU/GPU hardware, be it cores/threads or vectors, would automatically be put to good use**
- **Scaling would be as expected:**
 - If the number of cores (or the vector size) doubled:
 - Scaling would be close to 2x, but certainly not just a few percent
- **We cannot afford to “rewrite” our software for every hardware change!**

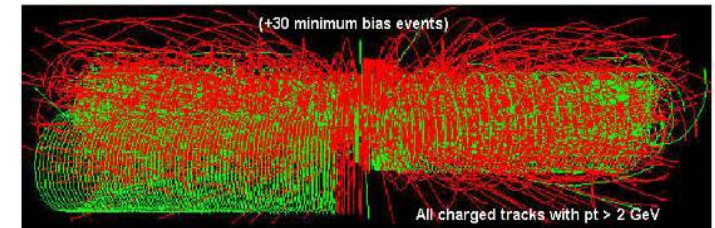
Concurrency in HEP

- We are “blessed” with lots of it:
 - Entire events
 - Particles, hits, tracks and vertices
 - Physics processes
 - I/O streams (ROOT trees, branches)
 - Buffer manipulations (also data compaction, etc.)
 - Fitting variables
 - Partial sums, partial histograms
 - and many others
- Usable for both **data** and **task** parallelism!
- **But, fine-grained parallelism is not well exposed in today's software frameworks**



HEP programming paradigm

- **Event-level parallelism has been used for decades**
- **And, we should not lose this advantage:**
 - Large jobs can be split into N efficient “chunks”, each responsible for processing M events
 - Has been our “forward scalability”
- **Disadvantage with current approach:**
 - Memory must be made available to each process
 - A dual-socket server with six-core processors needs 24 – 36 GB (or more)
 - Today, SMT is often switched off in the BIOS (!)
- **We must not let memory limitations decide our ability to compute!**

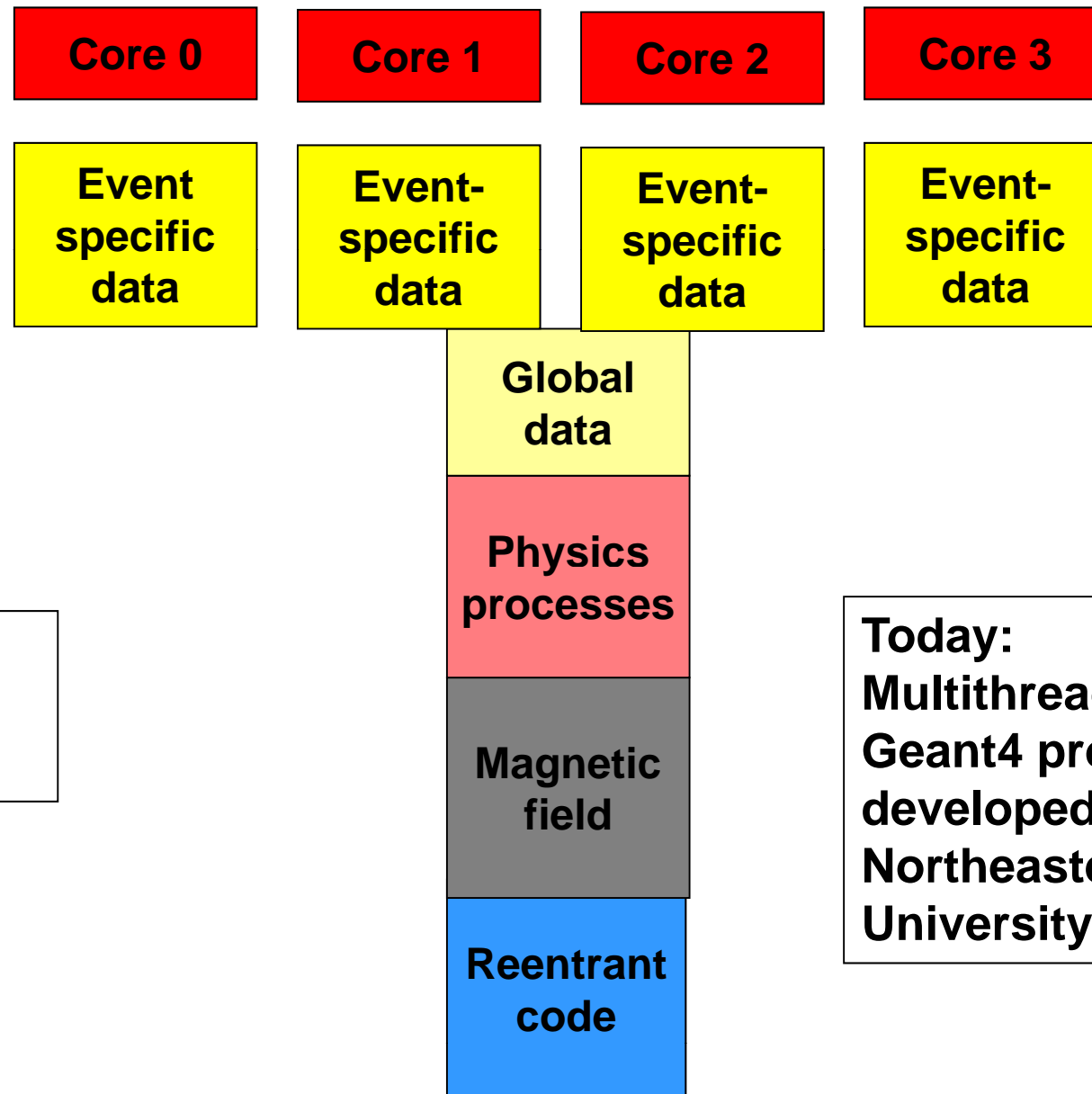


What are the multi-core options?

- **There is a discussion in the community about the best way(s) forward:**
 - 1) Stay with event-level parallelism (and entirely independent processes)
 - Assume that the necessary memory remains affordable
 - Or rely on tools, such as KSM, to help share pages
 - 2) Rely on forking:
 - Start the first process; Run through the first “event”
 - Fork N other processes
 - Rely on the OS to do “copy on write”, in case pages are modified
 - 3) Move to a fully multi-threaded paradigm
 - Still using coarse-grained (event-level) parallelism
 - But, watch out for increased complexity

Achieving an efficient memory footprint

- As follows:



Slide shown
in my talk at
CHEP2007

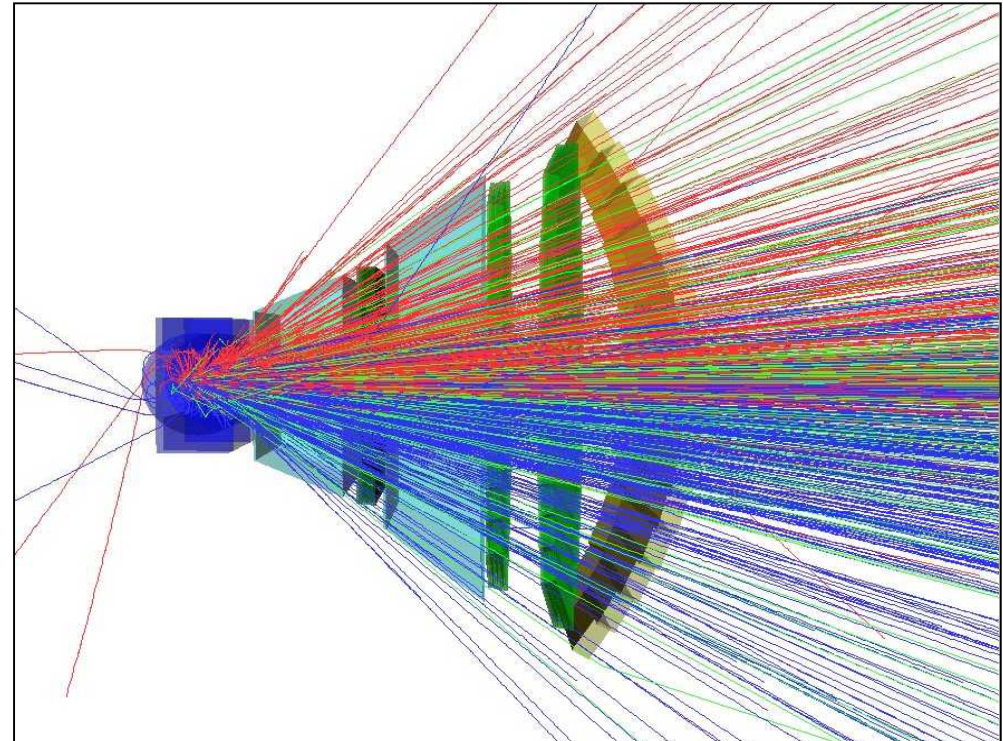
Today:
Multithreaded
Geant4 prototype
developed at
Northeastern
University

Promising software examples

Examples of parallelism: CBM/ALICE track fitting

- **Extracted from the High Level Trigger (HLT) Code**
 - Originally ported to IBM's Cell processor
- **Tracing particles in a magnetic field**
 - Embarrassingly parallel code
- **Re-optimization on x86-64 systems**
 - Using vectors instead of scalars

I.Kisel/GSI: "Fast SIMDized Kalman filter based track fit"
http://www-linux.gsi.de/~ikisel/17_CPC_178_2008.pdf



"Compressed Baryonic Matter"

CBM/ALICE track fitting

- **Details of the re-optimization:**

- **Step 1:** use SSE vectors instead of scalars

- Operator overloading allows seamless change of data types

- **Intrinsics** (from Intel/GNU header file): Map directly to instructions:

- `__mm_add_ps` corresponds directly to **ADDPS**, the instruction that operates on **four** packed, single-precision FP numbers
 - 128 bits in total

- **Classes**

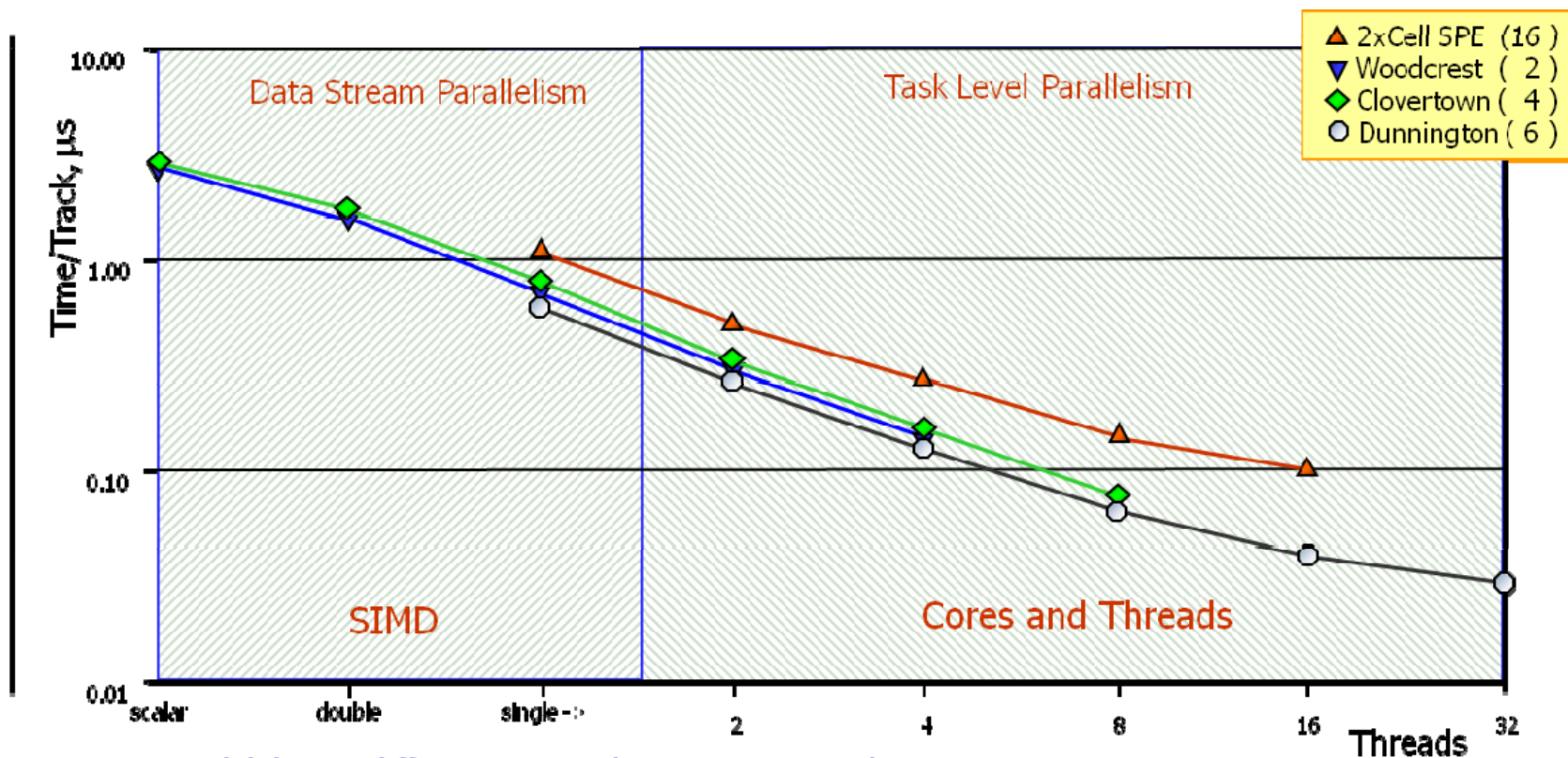
- `P4_F32vec4` – packed single class with overloaded operators

- `F32vec4` operator `+(const F32vec4 &a, const F32vec4 &b) { return __mm_add_ps(a,b); }`

- **Result:** **4x** speed increase from x87 scalar to packed SSE (single precision)

Examples of parallelism: CBM track fitting

- **Re-optimization on x86-64 systems**
 - Step 1: Data parallelism using SIMD instructions
 - **Step 2:** use TBB (or OpenMP) to scale across cores



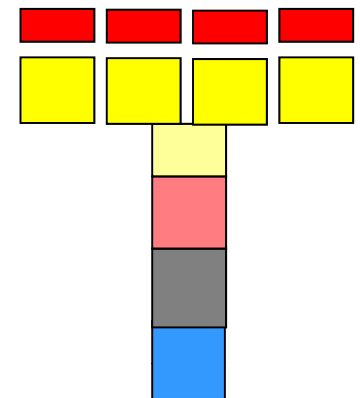
Scalability on different CPU architectures – speed-up 100

Examples of parallelism: GEANT4

- **Initially: ParGeant4 (Gene Cooperman/NEU)**
 - implemented event-level parallelism to simulate separate events across remote nodes.

- **New *prototype* re-implements thread-safe event-level parallelism inside a multi-core node**
 - Done by NEU PhD student Xin Dong: Using **FullCMS** and TestEM examples
 - Required change of lots of existing classes (10% of 1 MLOC):
 - Especially *global*, “*extrn*”, and *static* declarations
 - Preprocessor used for automating the work.
 - Major reimplementation:
 - Physics tables, geometry, stepping, etc.

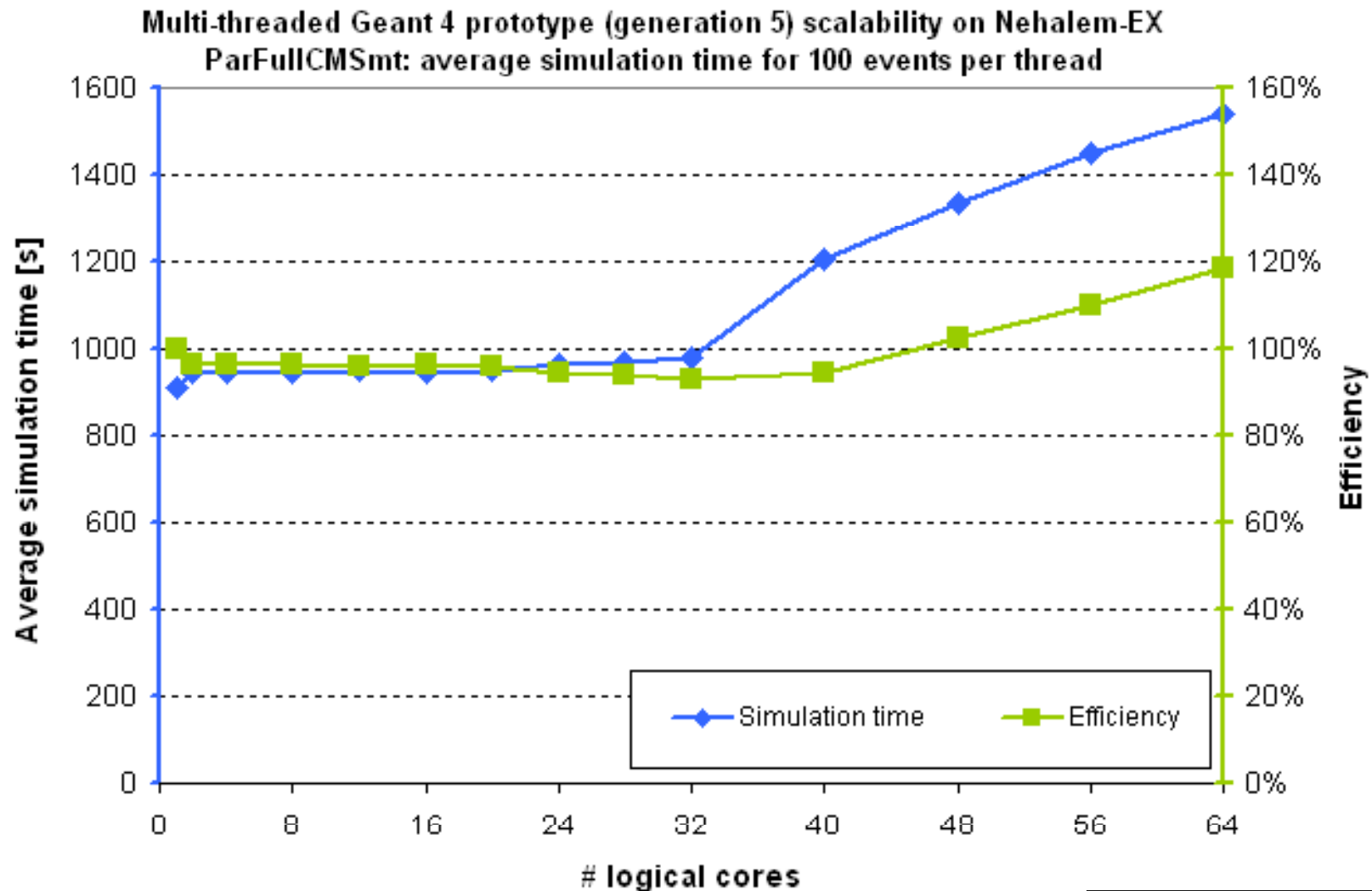
- **Additional memory: **Only 25 MB/thread (!)****



Dong, Cooperman, Apostolakis: “Multithreaded Geant4: Semi-Automatic Transformation into Scalable Thread-Parallel Software”, Europar 2010

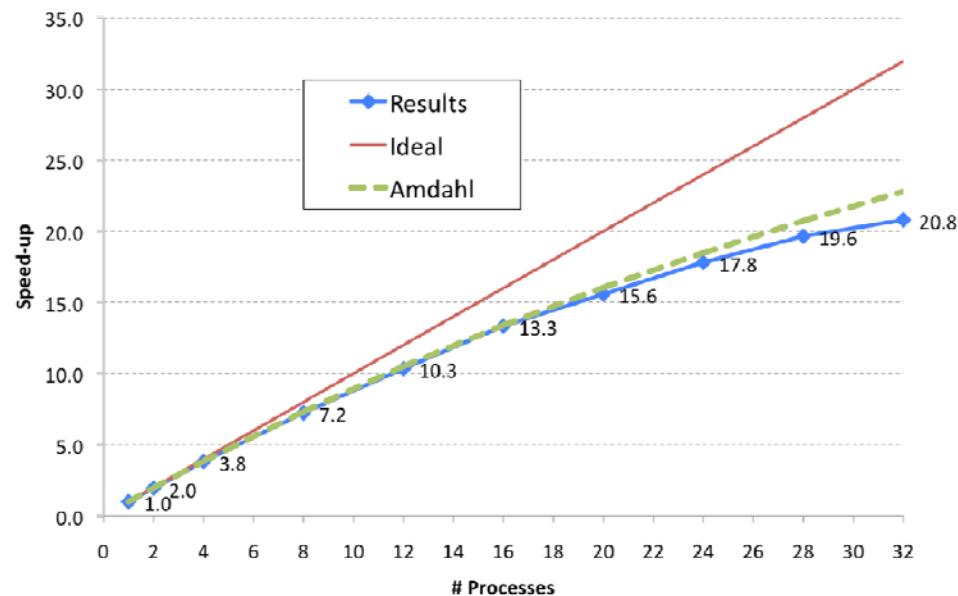
Multithreaded GEANT4 benchmark

- **Excellent scaling on 32 (real) cores**
 - With a 4-socket server



Example: ROOT minimization and fitting

- Minuit parallelization is independent of user code
- Log-likelihood parallelization (splitting the sum) is quite efficient
- Example on a 32-core server:

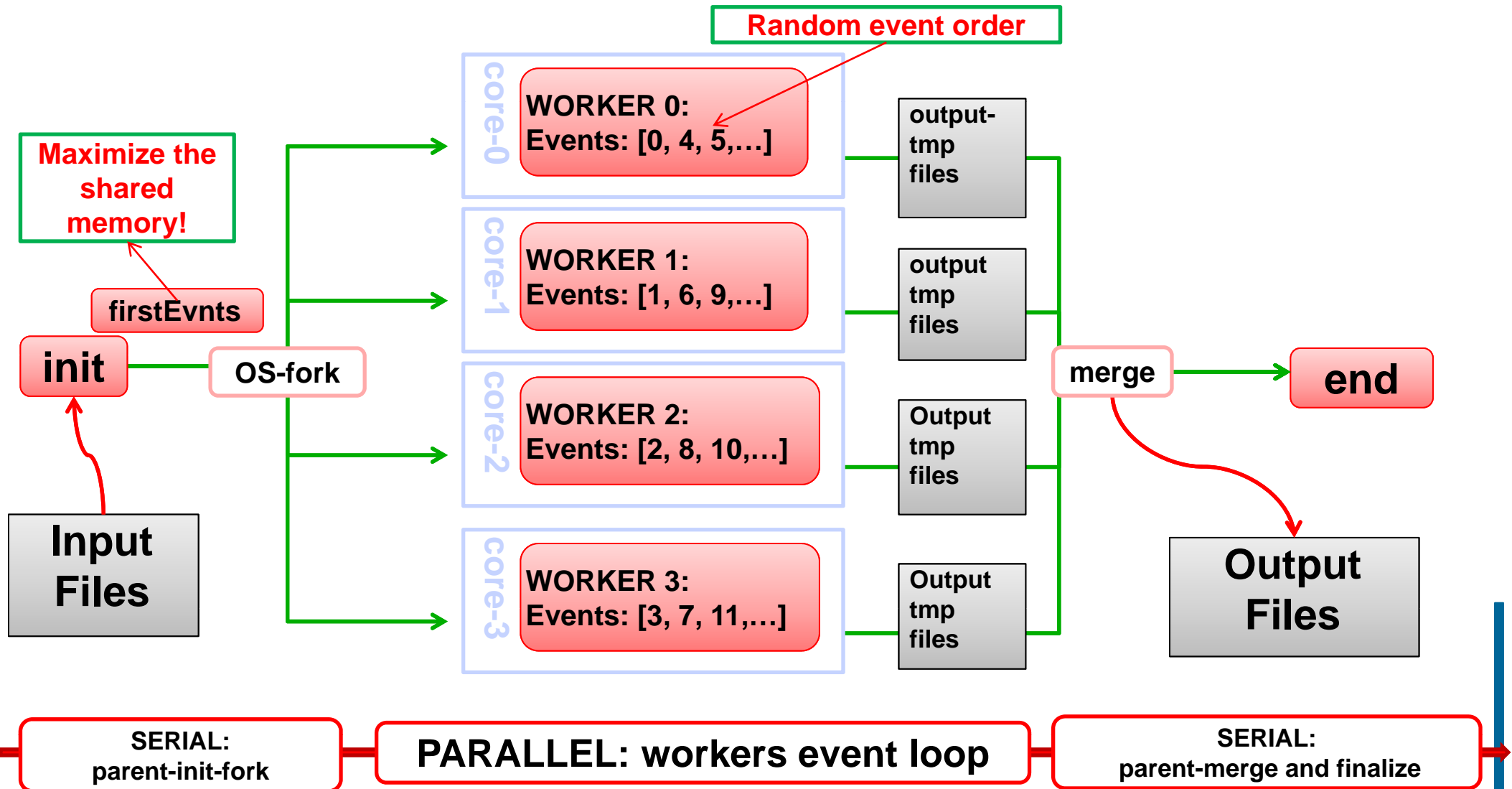


complex
BaBar fitting
provided by
A. Lazzaro
and
parallelized
using MPI

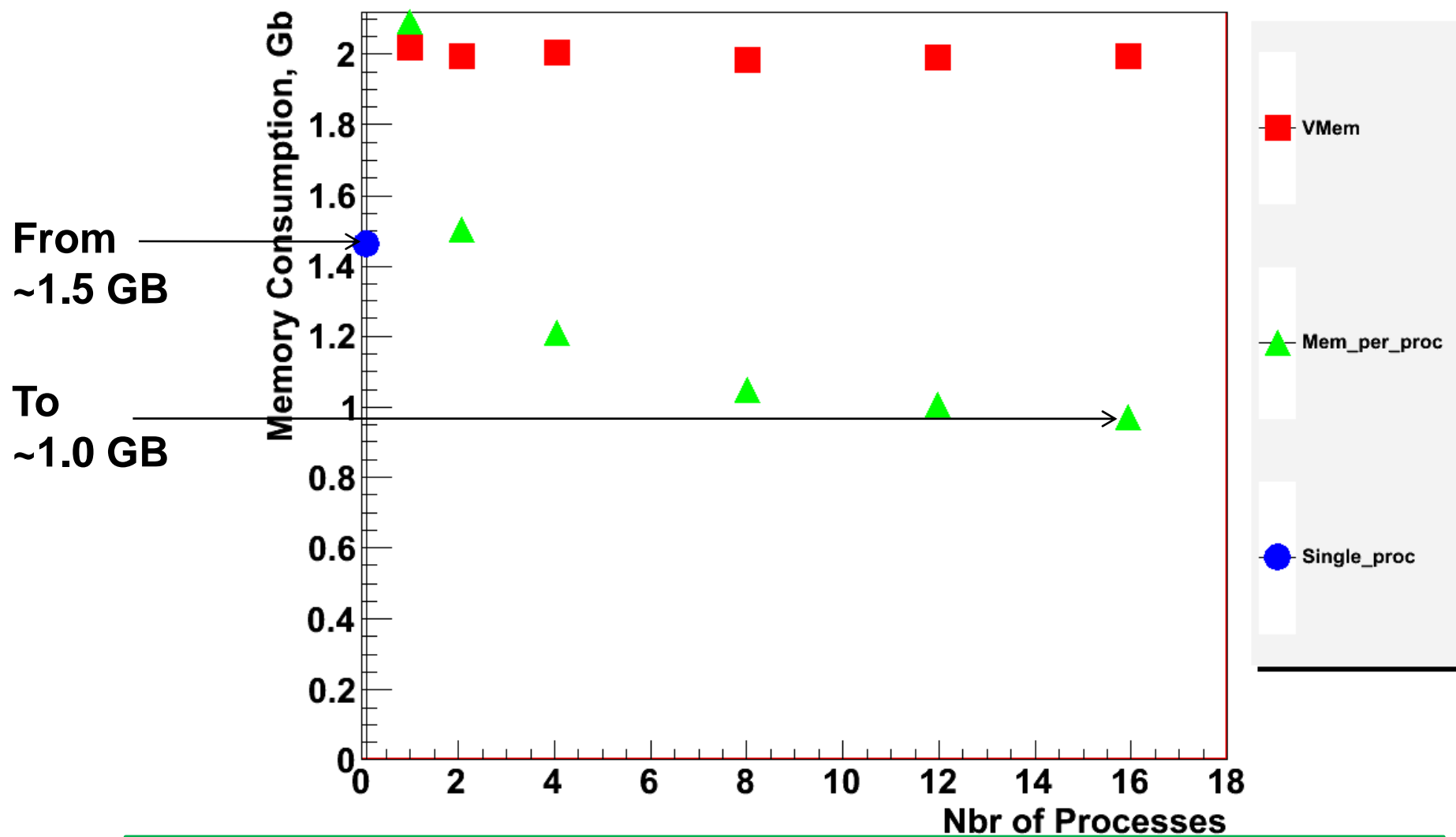
- **In principle, we can have combination of:**
 - parallelization via multi-threading in a multi-core CPU
 - multiple processes in a distributed computing environment

AthenaMP: event level parallelism

```
$> Athena.py --nprocs=4 -c EvtMax=100 Jobo.py
```

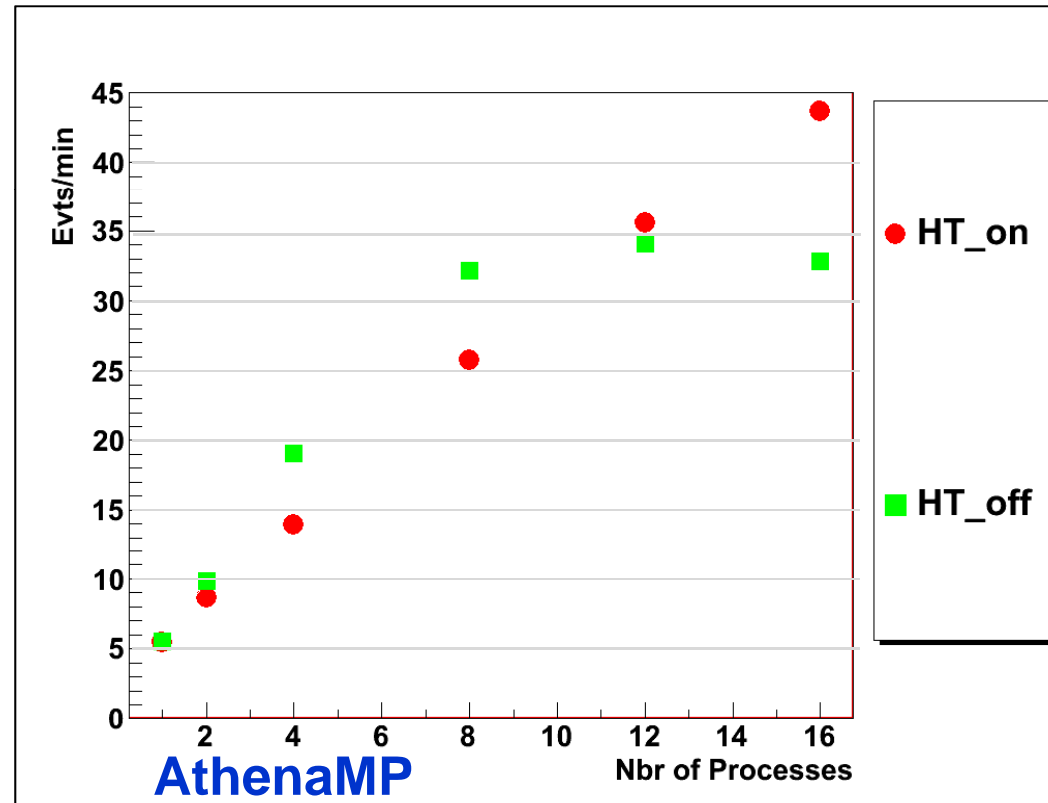


Memory footprint of AthenaMP



AthenaMP ~0.5 GB physical memory saved per process

Scalability plots for Athena MP



- Surprisingly good scaling with SMT on server with 8 physical cores (16 logical)

Recommendations

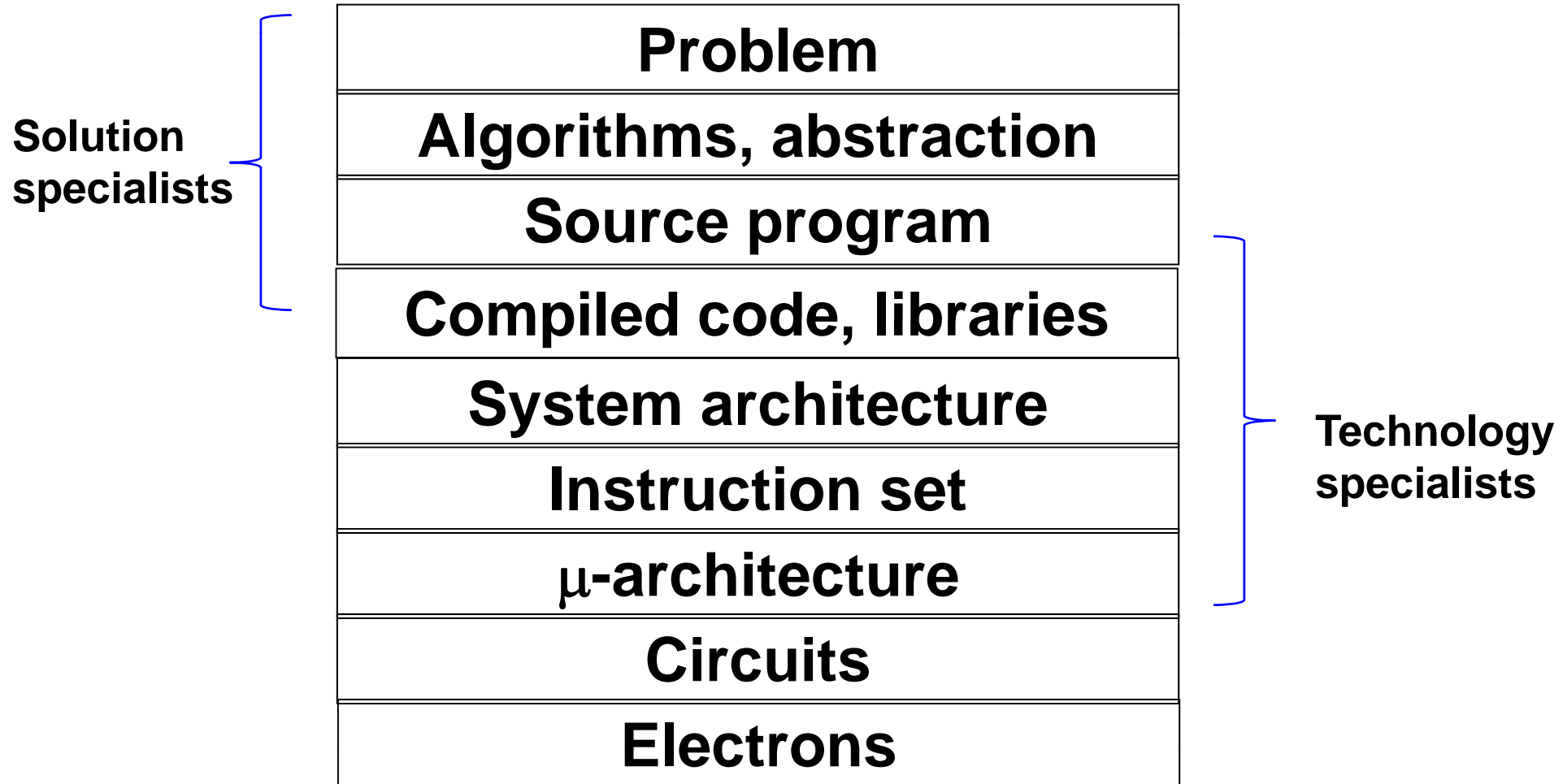
(based on observations in openlab)

Shortlist


- 1) Broad Programming Talent**
- 2) Holistic View with a clear split:
Prepare to compute – Compute**
- 3) Controlled Memory Usage**
- 4) C++ for Performance**
- 5) Best-of-breed Tools**

Broad Programming Talent

- In order to cover as many layers as possible



Performance guidance (cont'd)

- **Take the whole program and its execution behaviour into account**
 - Get yourself a global overview as soon as possible
 - Via early prototypes
 - Influence early the design and definitely the implementation
- **Foster clear split:**
 - Prepare to compute
 - Do the heavy computation 
 - Where you go after the available parallelism
 - Post-processing
- **Consider exploiting the entire server**
 - Using affinity scheduling

Performance guidance (cont'd)

- **Control memory usage (both in a multi-core and an accelerator environment)**
 - Optimize malloc/free
 - Forking is good; it may cut memory consumption in half
 - Don't be afraid of threading; it may perform miracles !
 - Optimize the cache hierarchy
 - NUMA: The “new” blessing (or curse?)

- **C++ for performance**
 - Use light-weight C++ constructs
 - Prefer SoA over AoS
 - Minimize virtual functions
 - Inline whenever important
 - Optimize the use of math functions
 - SQRT, DIV; LOG, EXP, POW; ATAN2, SIN, COS

C++ parallelization support

- **Large selection of tools (inside the compiler or as additions):**
 - Native: pthreads/Windows threads
 - Forthcoming C++ standard: `std::thread`
 - OpenMP
 - Intel Array Building Blocks (beta version from Intel; integrating RapidMind)
 - Intel Threading Building Blocks (TBB)
 - TOP-C (from NE University)
 - MPI (from multiple providers), etc.
 -

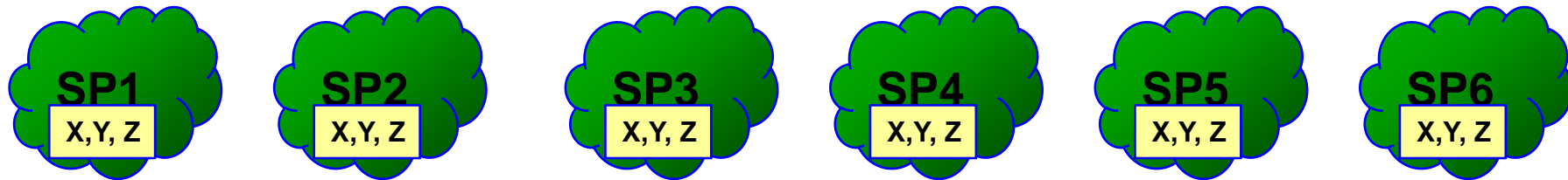
**We must also keep a close eye on
OpenCL (www.khronos.org/opencvl)**

Performance guidance (cont'd)

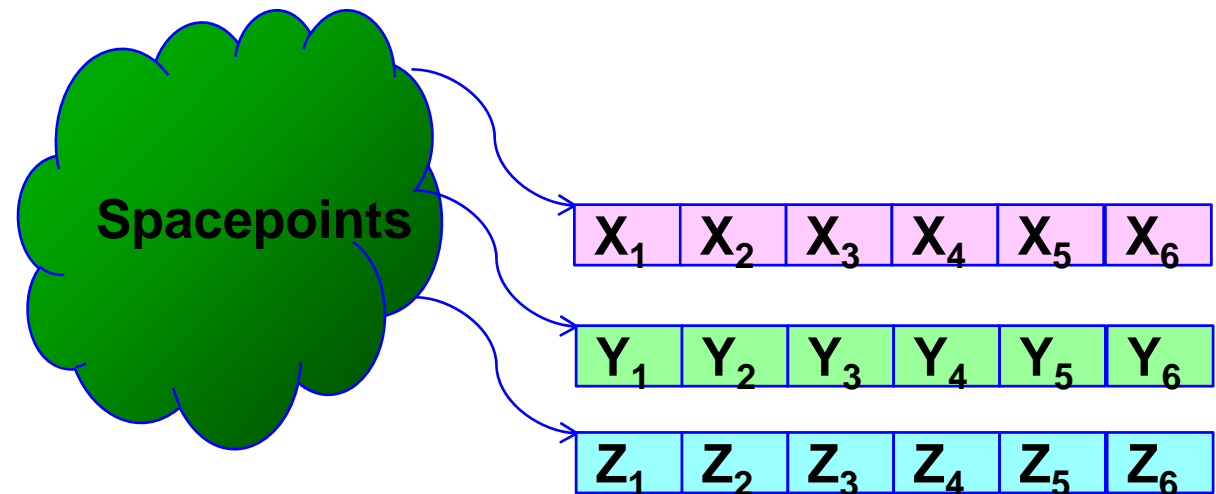
- Control memory usage (both in a multi-core and an accelerator environment)
 - Optimize malloc/free
 - Forking is good; it may cut memory consumption in half
 - Don't be afraid of threading; it may perform miracles !
 - Optimize the cache hierarchy
 - NUMA: The new blessing (or curse?)
- **C++ for performance**
 - Use light-weight C++ constructs
 - Prefer SoA over AoS
 - Minimize virtual functions
 - Inline whenever important
 - Optimize the use of math functions
 - SQRT, DIV; LOG, EXP, POW; ATAN2, SIN, COS

Organization of data: AoS vs SoA

- In general, compilers and hardware prefer the latter!
- Arrays of Structures:

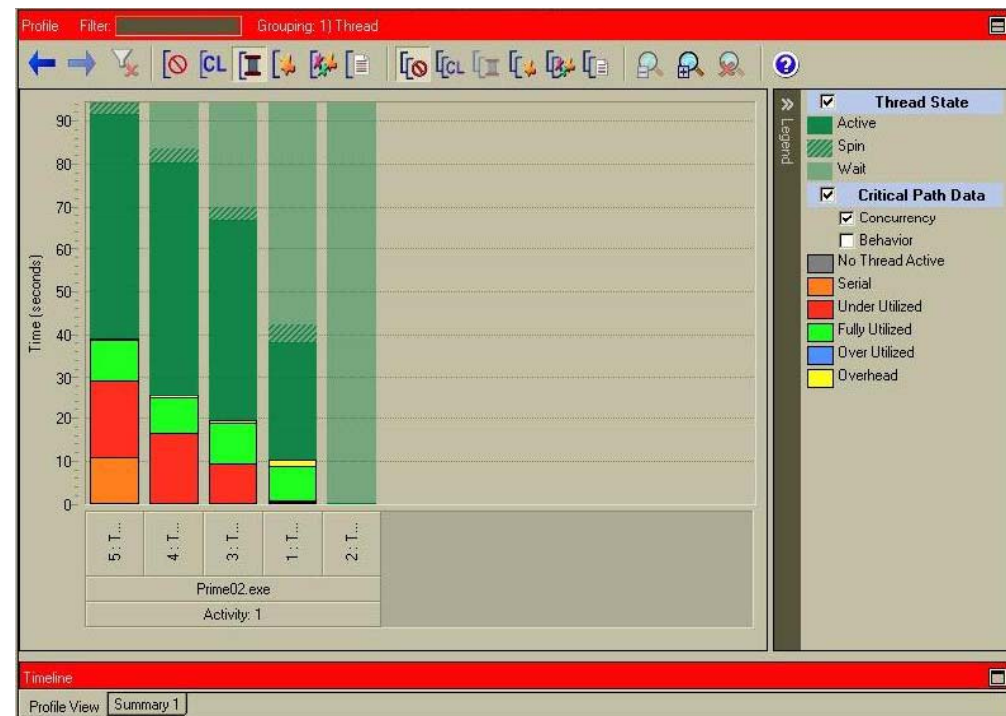


- Structure of Arrays:



Performance guidance (cont'd)

- **Surround yourself with good tools:**
 - Compilers
 - Libraries
 - Profilers
 - Debuggers
 - Thread checkers
 - Thread profilers



Lots of related presentations during this CHEP conference (Sorry if I missed some!)

- Evaluating the **Scalability** of HEP Software and Multi-core Hardware [77]
- ng: What Next-Gen Languages Can Teach Us About HENP Frameworks in the **Manycore** Era [114]
- **Multicore-aware** Applications in CMS [115]
- Parallelizing Atlas Reconstruction and Simulation: Issues and Optimization Solutions for Scaling on Multi- and **Many-CPU** Platforms [116]
- **Multi-threaded** Event Reconstruction with JANA [117]
- Track Finding in a High-Rate Time Projection Chamber Using **GPUs** [163]
- **Fast Parallel** Tracking Algorithm for the Muon System and Transition Radiation Detector of the CBM Experiment at FAIR [164]
- Real Time Pixel Data Reduction with **GPUs** And Other HEP GPU Applications [272]
- Algorithm Acceleration from **GPGPUs** for the ATLAS Upgrade [273]
- Maximum Likelihood Fits on **Graphics Processing Units** [297]
- Partial Wave Analysis **on Graphics Processing Units** [298]
- **Many-Core** Scalability of the Online Event Reconstruction in the CBM Experiment [299]
- Adapting Event Reconstruction Software to **Many-Core** Computer Architectures [300]
- BOF 3 – GPUs: High Performance Co-Processors

Concluding remarks

- **The hardware is getting more and more powerful**
 - But also more and more **complex!**
 - Watch out for the transistor “tsunami”!
- **In most HEP programming domains event-level processing will and should continue to dominate**
- **We can still move the software forward in multiple ways**
- **But it should be able to profit from ALL the available hardware**
 - Accelerators with limited memory, as well as
 - Conventional servers
- **Holy grail: Forward scalability**

Thank you!

“Intel platform 2015” (and beyond)

- **Today’s silicon processes:**

- 45 nm
- 32 nm

- **On the roadmap:**

- 22 nm (2011/12)
- 16 nm (2013/14)

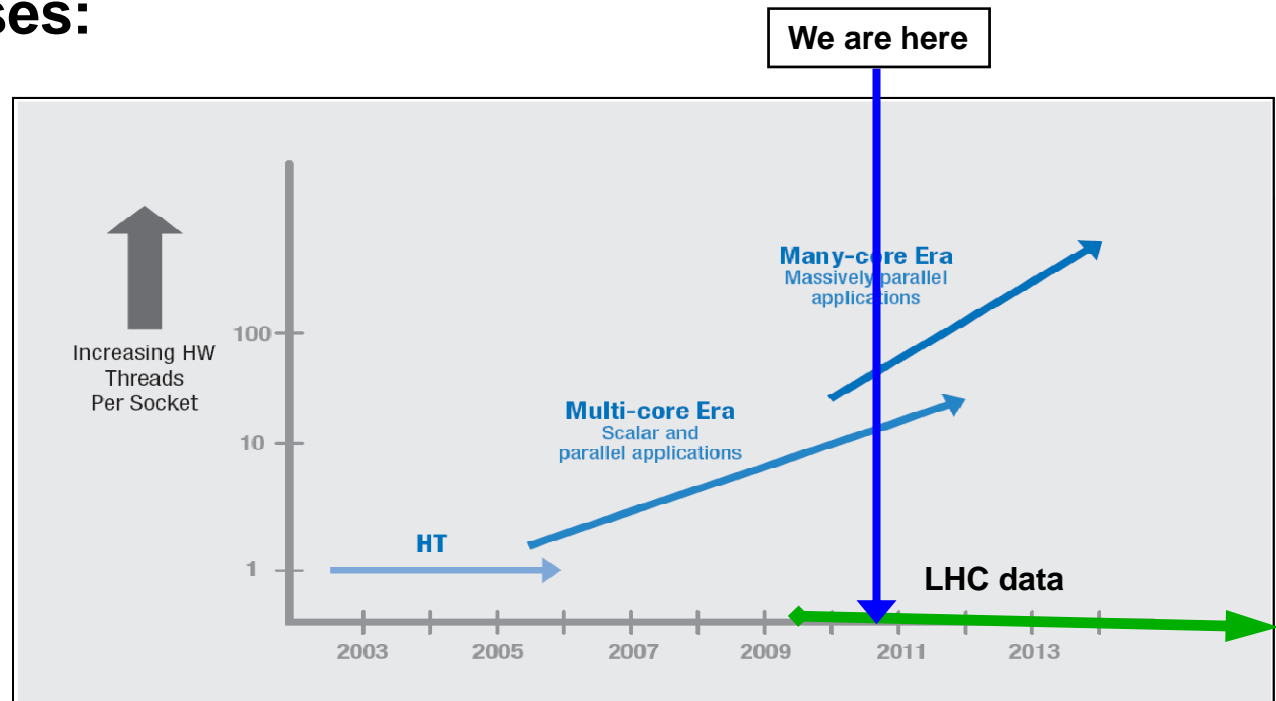
- **In research:**

- 11 nm (2015/16)
- 8 nm (2017/18)

– Source: Bill Camp/Intel HPC

- **Each generation will push the core count:**

- **We are already in the many-core era (whether we like it or not) !**



S. Borkar et al. (Intel), "Platform 2015: Intel Platform Evolution for the Next Decade", 2005.

HEP and vectors

- **Too little common ground**
 - And, practically all attempts in the past failed!
 - w/CRAY, IBM 3090-Vector Facility, etc.
- **From time to time, we see a good vector example**
 - For example: Track Fitting code from ALICE trigger
 - → See later
- **Interesting development from ALICE (Matthias Kretz):**
 - Vc (Vector Classes)
 - <http://www.kip.uni-heidelberg.de/~mkretz/Vc/>
- **Other examples: Use of STL vectors; small matrices**