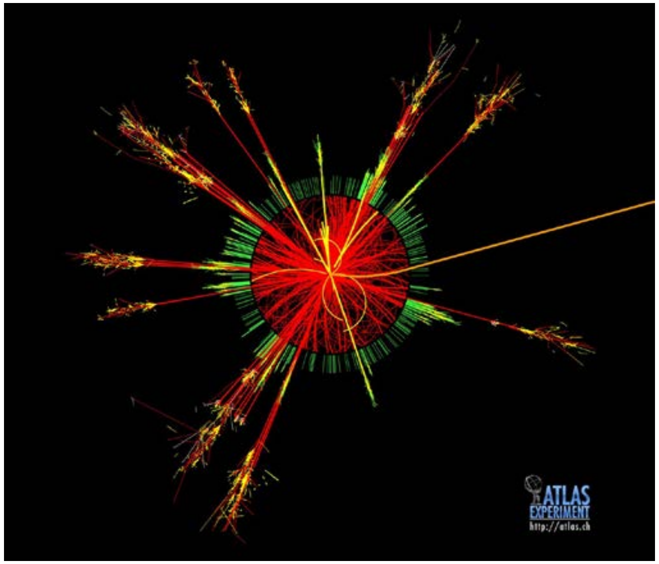
GPUs in High Energy Physics workshop
15 - 16 April 2013

Moving to Good Software Designs

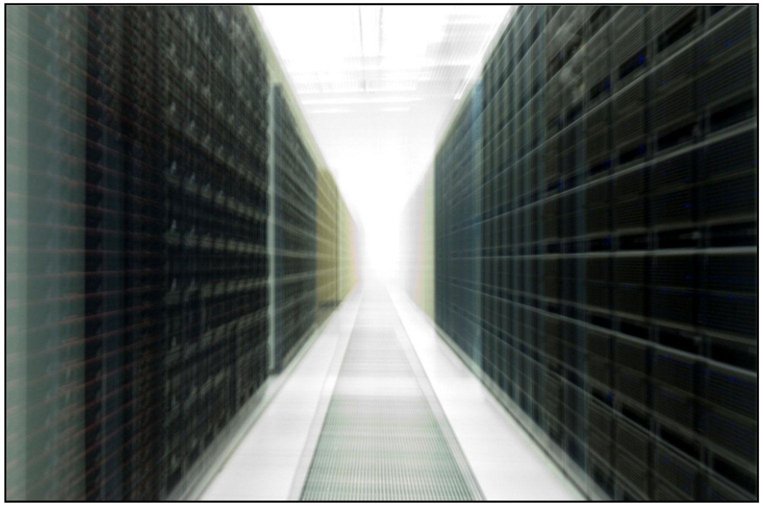
(given the complexity of modern computing devices)

“The 7 dimensions of performance”



Sverre Jarp
CERN
openlab
CTO

IT Dept., CERN



What is the CERN openlab?

- A science-industry partnership to drive R&D and innovation with over a decade of success
- Evaluate state-of-the-art technologies in a challenging environment and improve them
- Test in a research environment today what will be used in many business sectors tomorrow
- Train next generation engineers/employees
- Disseminate results and outreach to new audiences

PARTNERS

hp

HUAWEI

intel

ORACLE

SIEMENS

ASSOCIATE

Yandex

Contents

- **Why worry about performance?**
- **Complexity in Computing**
- **Guidelines for SW design**
- **Some HEP examples**
- **Conclusions**

Why worry about performance?

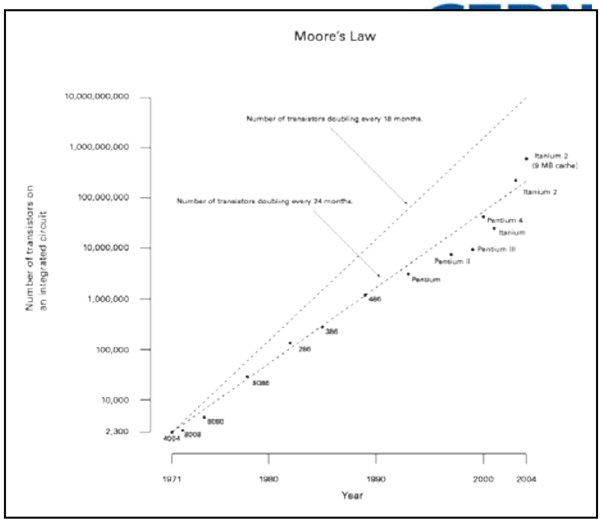
- **My arguments:**
  - The “easy ride” disappeared: The frequency scaling we enjoyed in the past does not exist any longer.
    It stopped a decade ago!
    - ..and, as a “by-product”, the CPU/GPU architectures are becoming (much) more complicated
  - Performance per watt: There are important thermal issues associated with large scale computing
    - Even when 1W processors exist!
  - Performance per €: There are important cost issues associated with large scale computing
    - Even when using “commodity equipment”

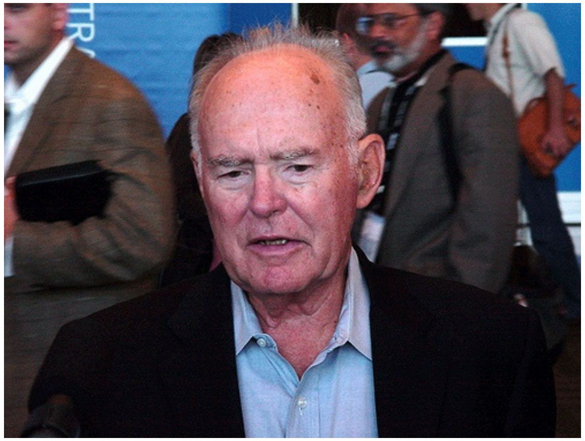
Moore’s law

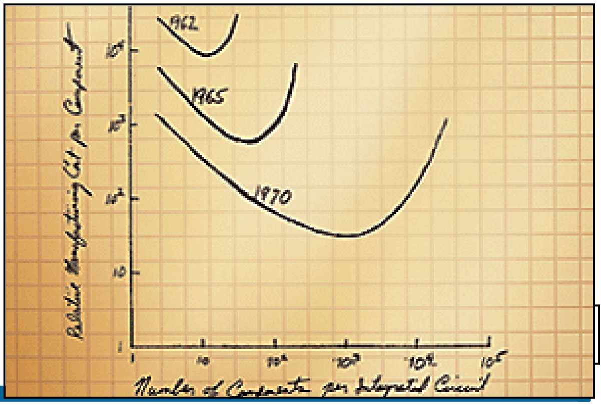
- **We continue to double the number of transistors every other year**

- **The consequences:**
  - CPUs
    - Single core → Multicore → Manycore
    - Hardware vector support
    - Hardware threading
  - GPUs
    - Huge number of floating-point units

- **Today, we commonly acquire chips with 1’000’000’000 transistors!**
  - Intel/AMD server chips and high-end GPU devices are much more
    - Kepler GK110: 7.1 billion transistors

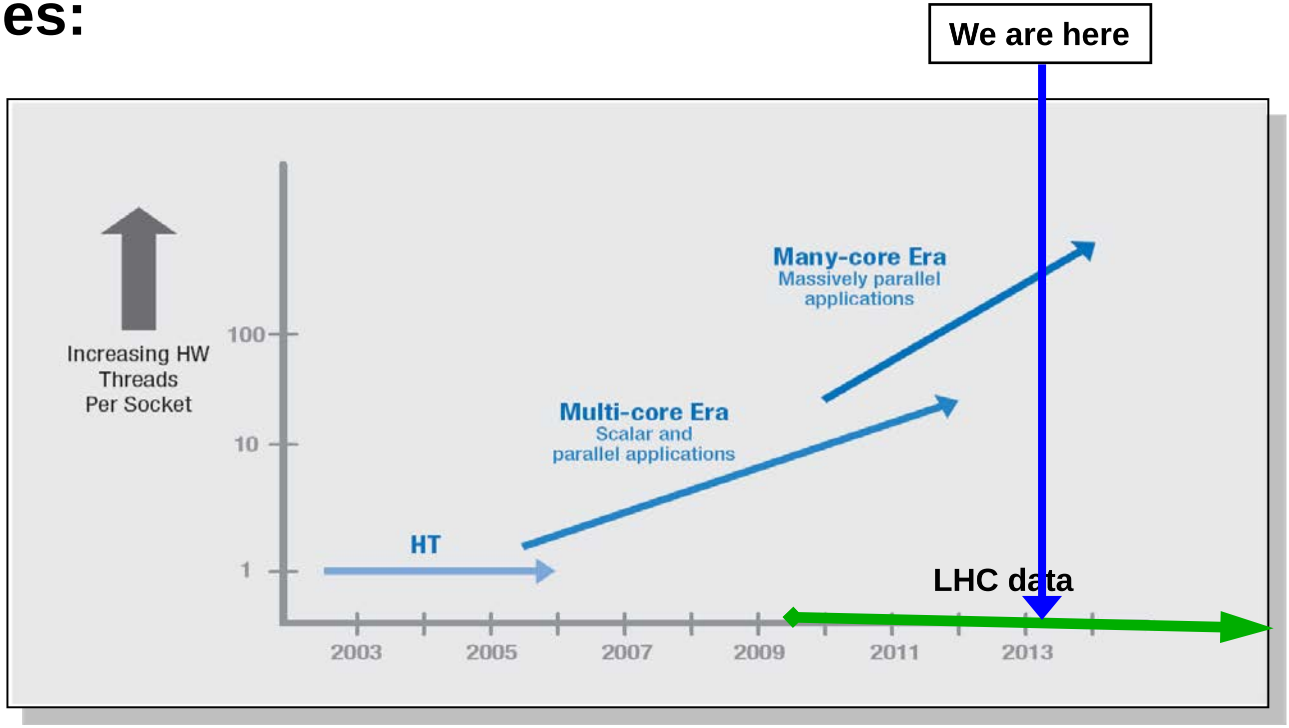





1962
1965
1970
Relative Manufacturing Cost per Component
10^4
10^3
10^2
10
1
1
10
10^2
10^3
10^4
10^5
Number of Components per Integrated Circuit

“Intel platform 2015” (and beyond)

- **Today’s silicon processes:**
  - 32, 28, 22 nm
- **Being introduced:**
  - 14 nm (2013/14)
- **In research:**
  - 10 nm (2015/16)
  - 7 nm (2017/18)
  - 5 nm (2019/20)
    - Source: Intel

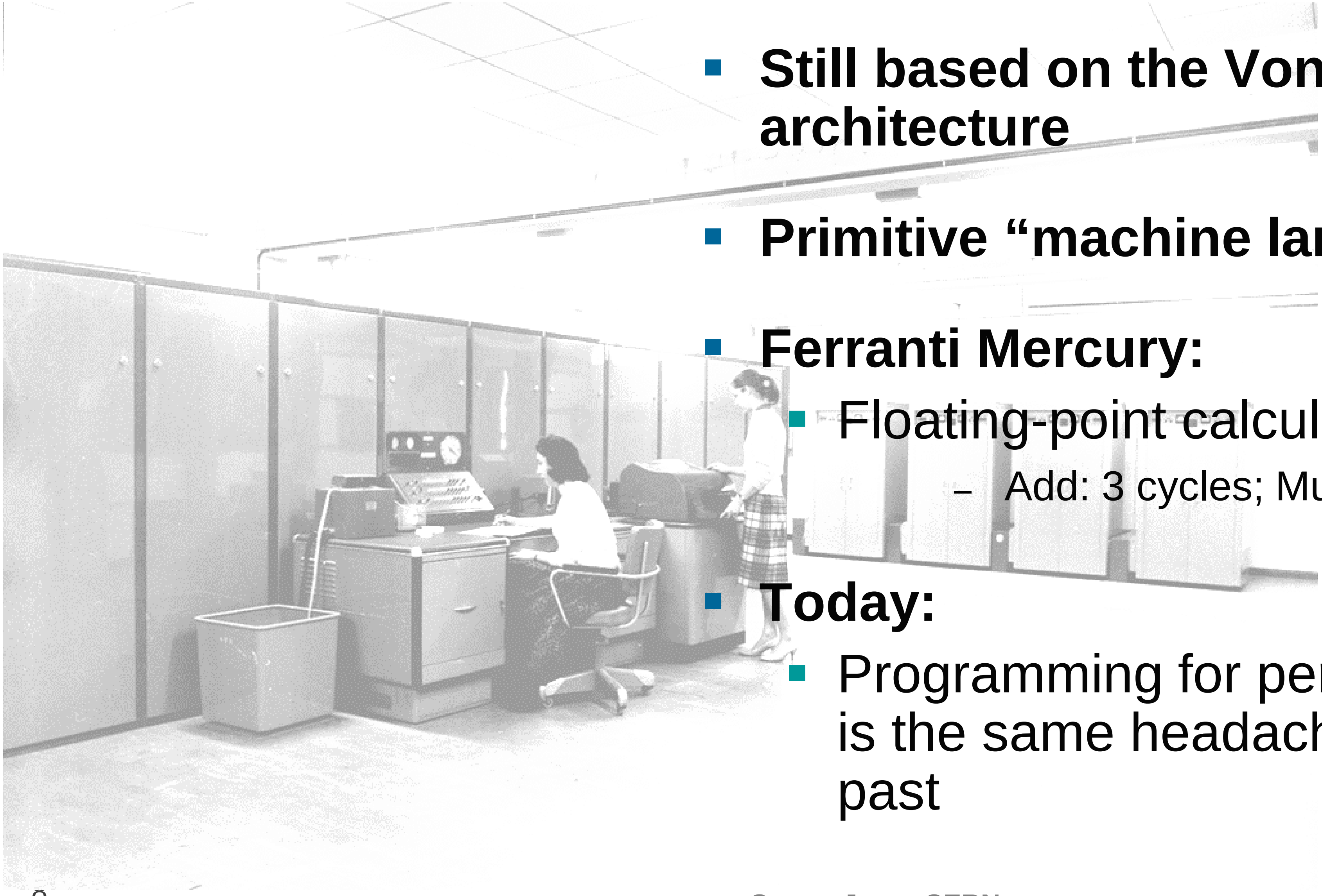


S. Borkar et al. (Intel), "Platform 2015: Intel Platform Evolution for the Next Decade", 2005.

- **Each generation will push the core count:**
  - We are inside the many-core era (whether we like it or not) !

Complexity in Computing

Archaic Computing Units



- **As “stupid” as 50 years ago**
- **Still based on the Von Neumann architecture**
- **Primitive “machine language”**
- **Ferranti Mercury:**
  - Floating-point calculations
    - Add: 3 cycles; Multiply: 5 cycles
- **Today:**
  - Programming for performance is the same headache as in the past

And the language is ancient, too!

- **Assembly/machine code!**

```
..B1.31:                      # Preds ..B1.31 ..B1.30         # Infreq
        movsd     (%rsp), %xmm3                                 #94.17
        lea       (%rbx,%rbx,2), %rcx                           #94.36
        movsd     (%rsi,%rcx,8), %xmm2                          #94.40
        incl      %eax                                          #93.42
        movsd     8(%rsi,%rcx,8), %xmm0                         #94.40
        cmpl      %edx, %eax                                    #93.39
        mulsd     %xmm2, %xmm2                                  #94.40
        mulsd     %xmm0, %xmm0                                  #94.40
        movsd     16(%rsi,%rcx,8), %xmm1                        #94.40
        addsd     %xmm0, %xmm2                                  #94.40
        mulsd     %xmm1, %xmm1                                  #94.40
        movl      %eax, %ebx                                    #93.42
        addsd     %xmm1, %xmm2                                  #94.40
        sqrtsd    %xmm2, %xmm2                                  #94.40
        addsd     %xmm2, %xmm3                                  #94.17
        movsd     %xmm3, (%rsp)                                 #94.17
        jb        ..B1.31       # Prob 82%                      #93.39
```

And, even assembly is “too high level”

- **Intel translates x86 assembly instructions**
  - into micro-operations

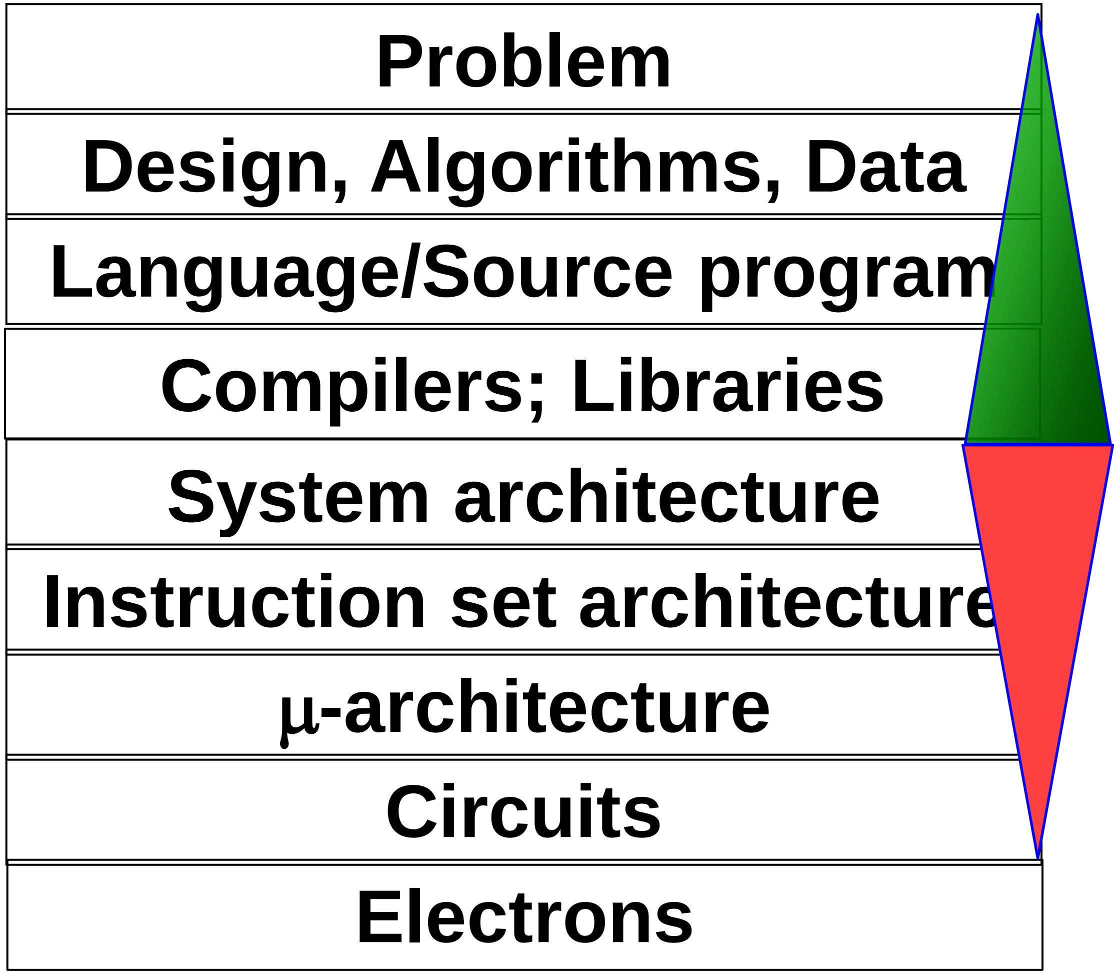
- **NVIDIA translates PTX (virtual assembly)**
  - into machine instructions

- **So, what does it mean (?) when the hardware tells you:**
  - “XX^N instructions executed”

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, C++, CUDA, 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**

A Complicated Story (in 9 layers!)



Problem

Design, Algorithms, Data

Language/Source program

Compilers; Libraries

System architecture

Instruction set architecture

μ-architecture

Circuits

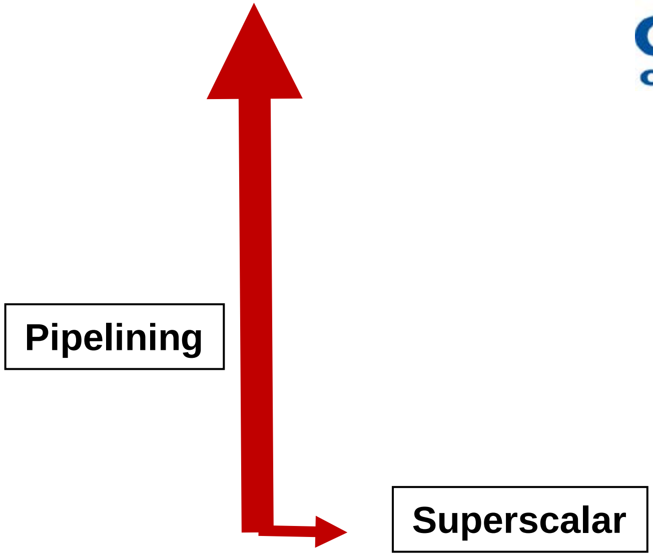
Electrons

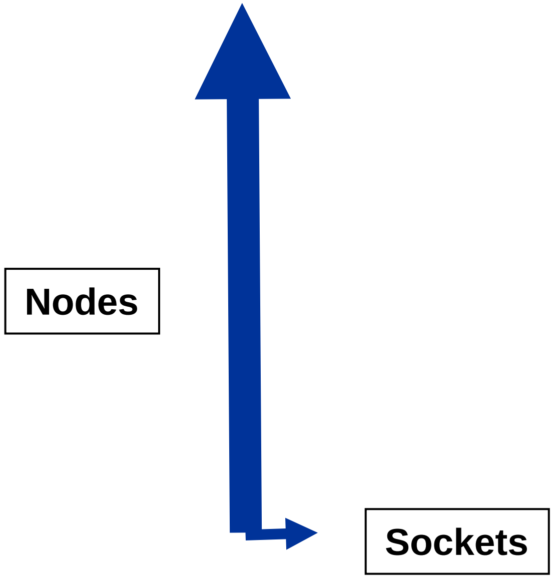
- **We must avoid being fenced into a single layer!**

Adapted from Y.Patt, U-Austin

In the days of the Pentium

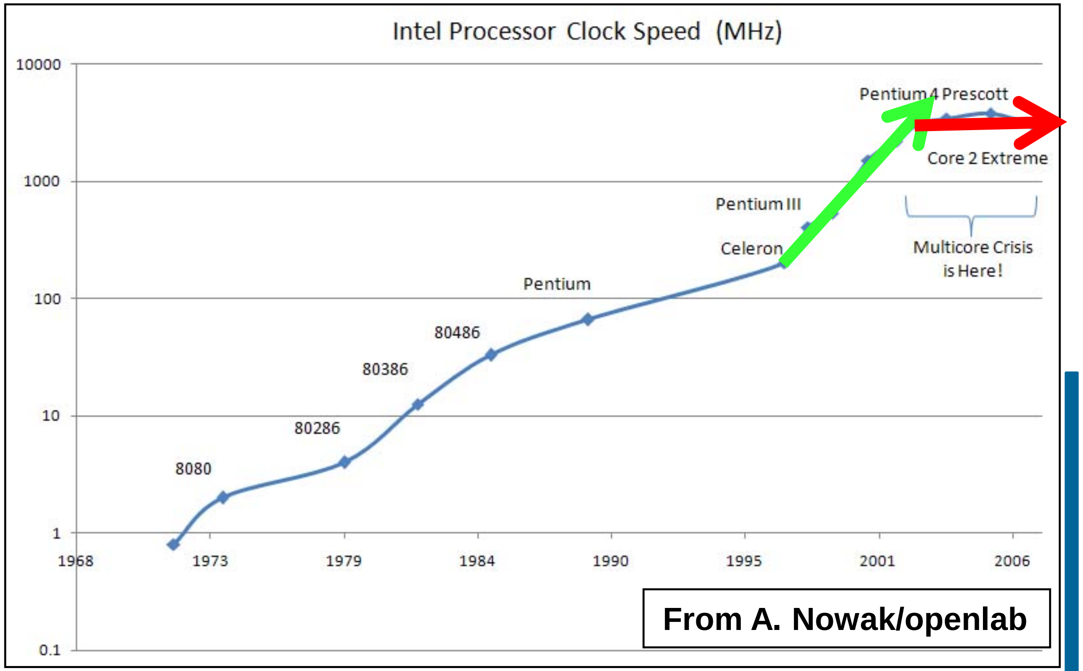
- **Life was really simple:**
  - Basically two dimensions
    - The frequency of the pipeline
    - The number of boxes
  - The semiconductor industry increased the frequency
  - We acquired the right number of (single-socket) boxes





Frequency scaling

- **The 7 “fat” years of frequency scaling in HEP**
  - The Pentium Pro in 1996: 150 MHz
  - The Pentium 4 in 2003: 3.8 GHz (~25x)
- **But, this was 10 years ago!**
- **Since then**
  - Core 2 systems:
    - ~3 GHz
    - Multi-core
- **Recent CERN purchase:**
  - Intel Xeon E5-2630L
    - “only” 2.00 GHz



From A. Nowak/openlab

Accelerators (1): Nvidia Kepler GPU

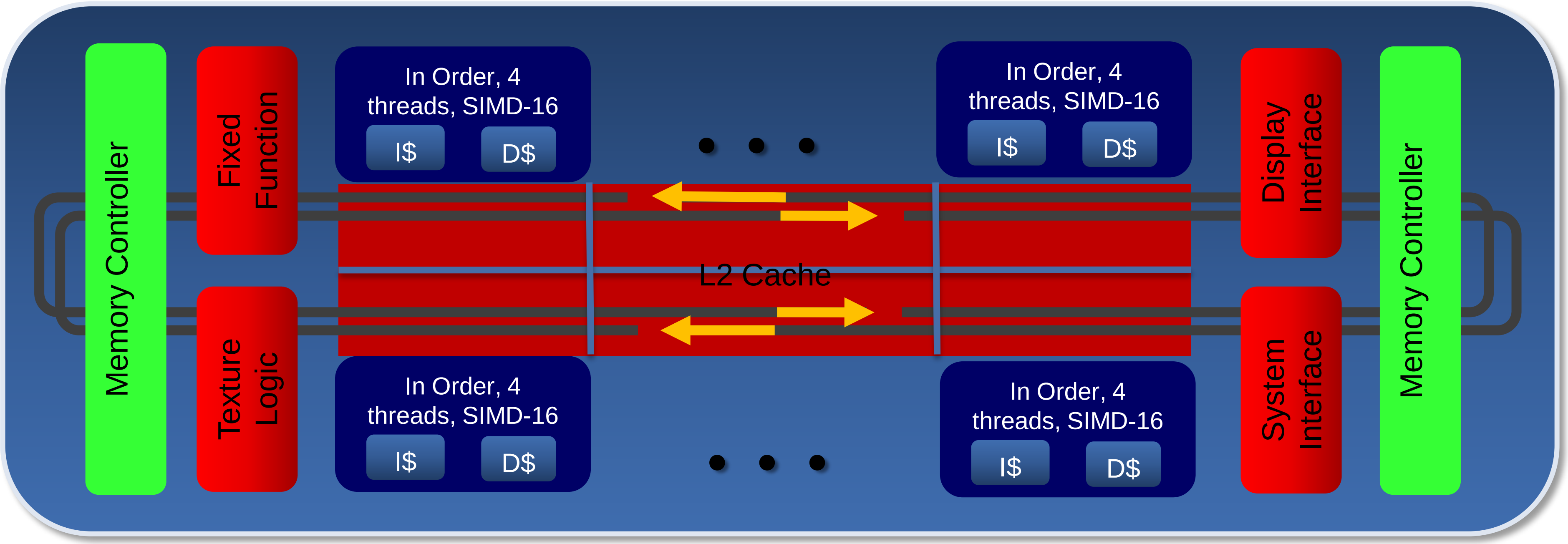
- **Made available in 4Q2012**
  - GK110 GPU
  - 3x DP performance:
    - >1 Teraflops
  - Innovative design:
    - **SMX** (streaming multiprocessors)
    - **Dynamic parallelism** for spawning new threads
    - **Hyper-Q** enables multiple CPU cores to utilise CUDA cores

Considerable interest in the HEP community

Adapted from Nvidia

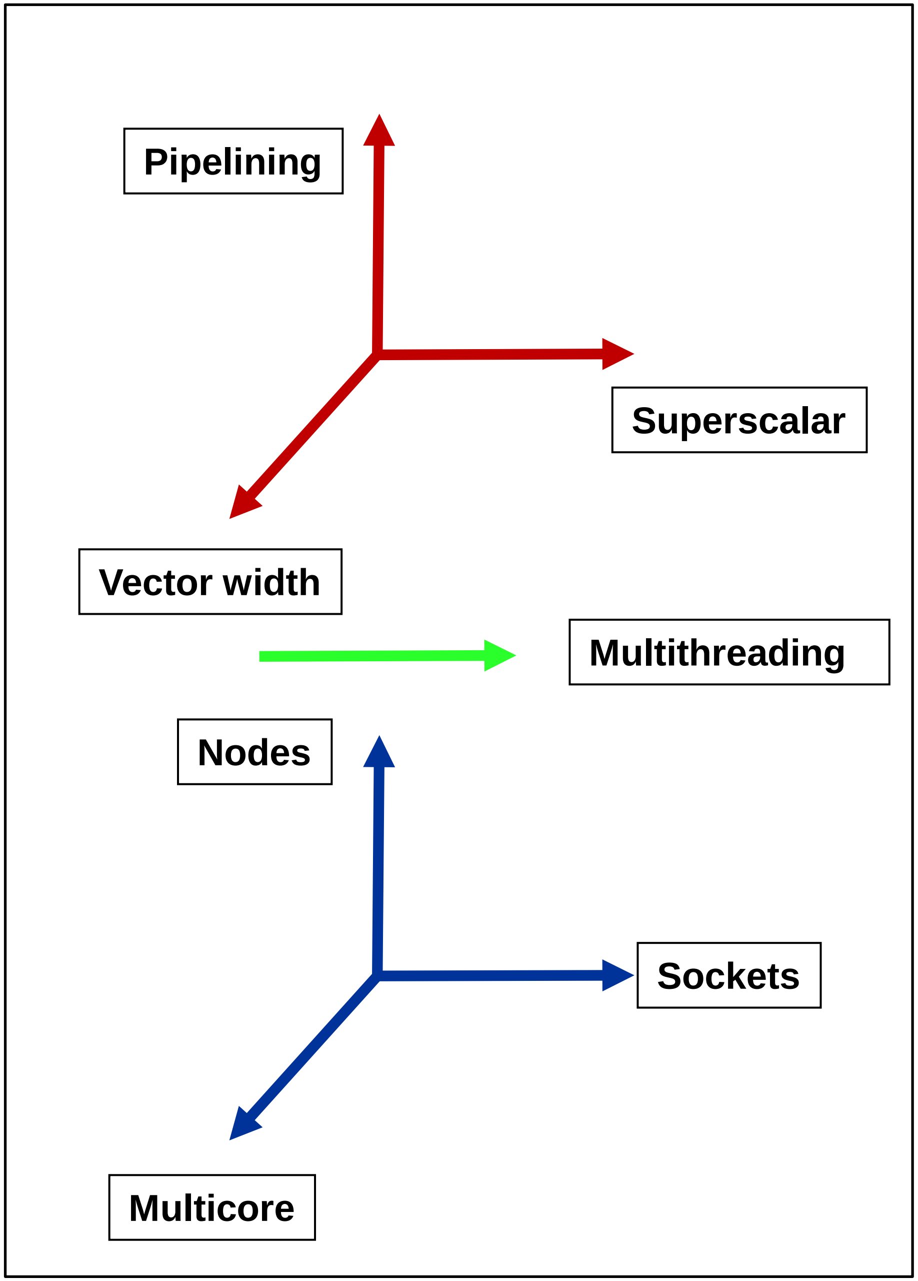
Accelerators (2): Intel Xeon Phi

- **Intel Many Integrated Cores (MIC):**
  - Announced at ISC10, available 2 ½ years later
  - Based on the x86 architecture, 22nm, ~1.0 GHz
  - Many-core (up to 62 cores) + 4-way multithreaded + 512-bit vector unit
  - Limited memory: 8 Gigabytes



CPU servers: 7 dimensions of performance

- **First three dimensions:**
  - Pipelining
  - Superscalar
  - Hardware vectors/SIMD
- **Next dimension is a “pseudo” dimension:**
  - Hardware multithreading
- **Last three dimensions:**
  - Multiple cores
  - Multiple sockets
  - Multiple compute nodes



SIMD = Single Instruction Multiple Data

Seven multiplicative dimensions:

- **First three dimensions:**
  - Pipelining
  - Superscalar
  - Hardware vectors/SIMD

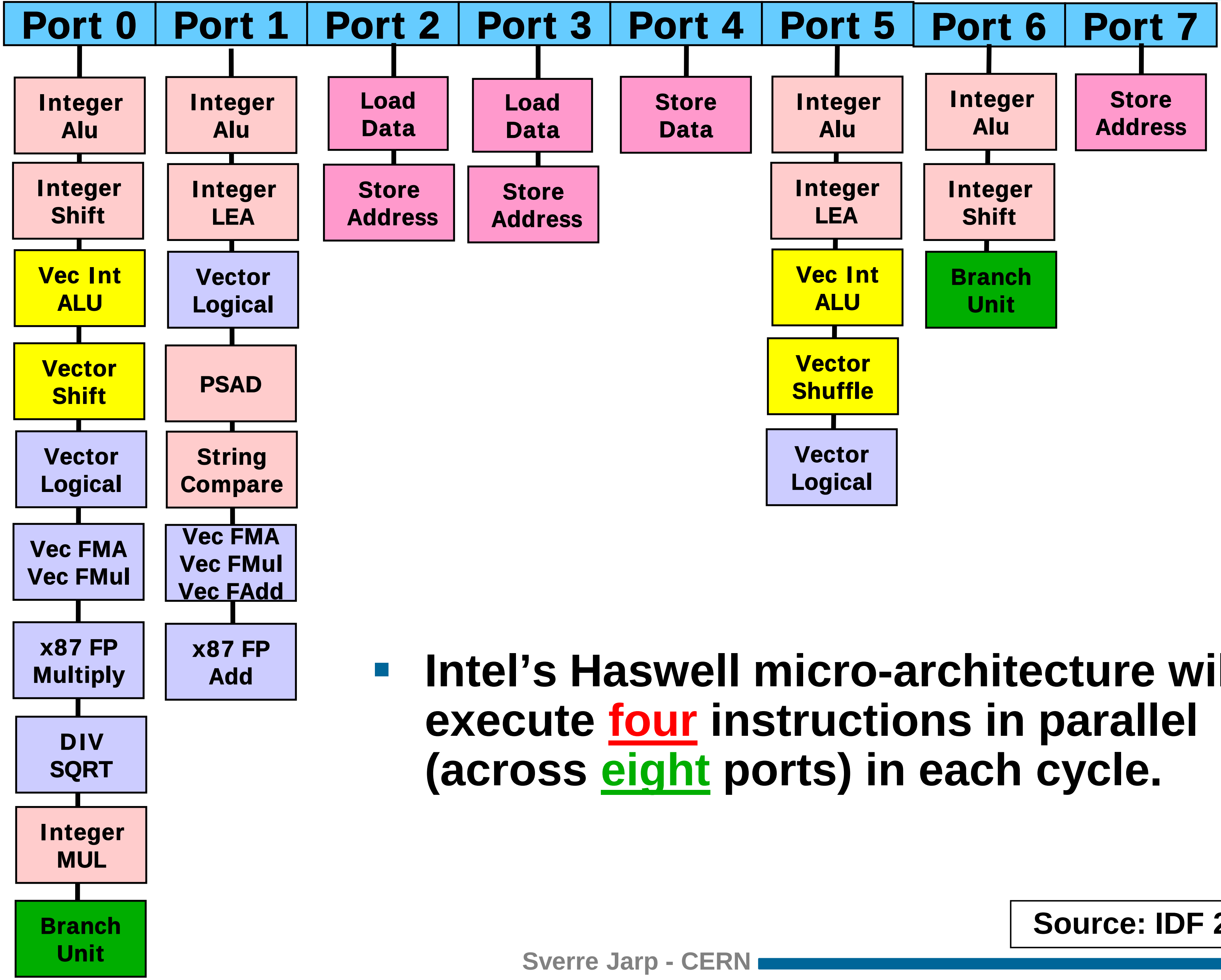
Data parallelism (Vectors/Matrices)

- **Next dimension is a "pseudo" dimension:**
  - Hardware multithreading
- **Last three dimensions:**
  - Multiple cores
  - Multiple sockets
  - Multiple compute nodes

Task parallelism (Events/Tracks)

Task/process parallelism

Intel Haswell superscalar architecture

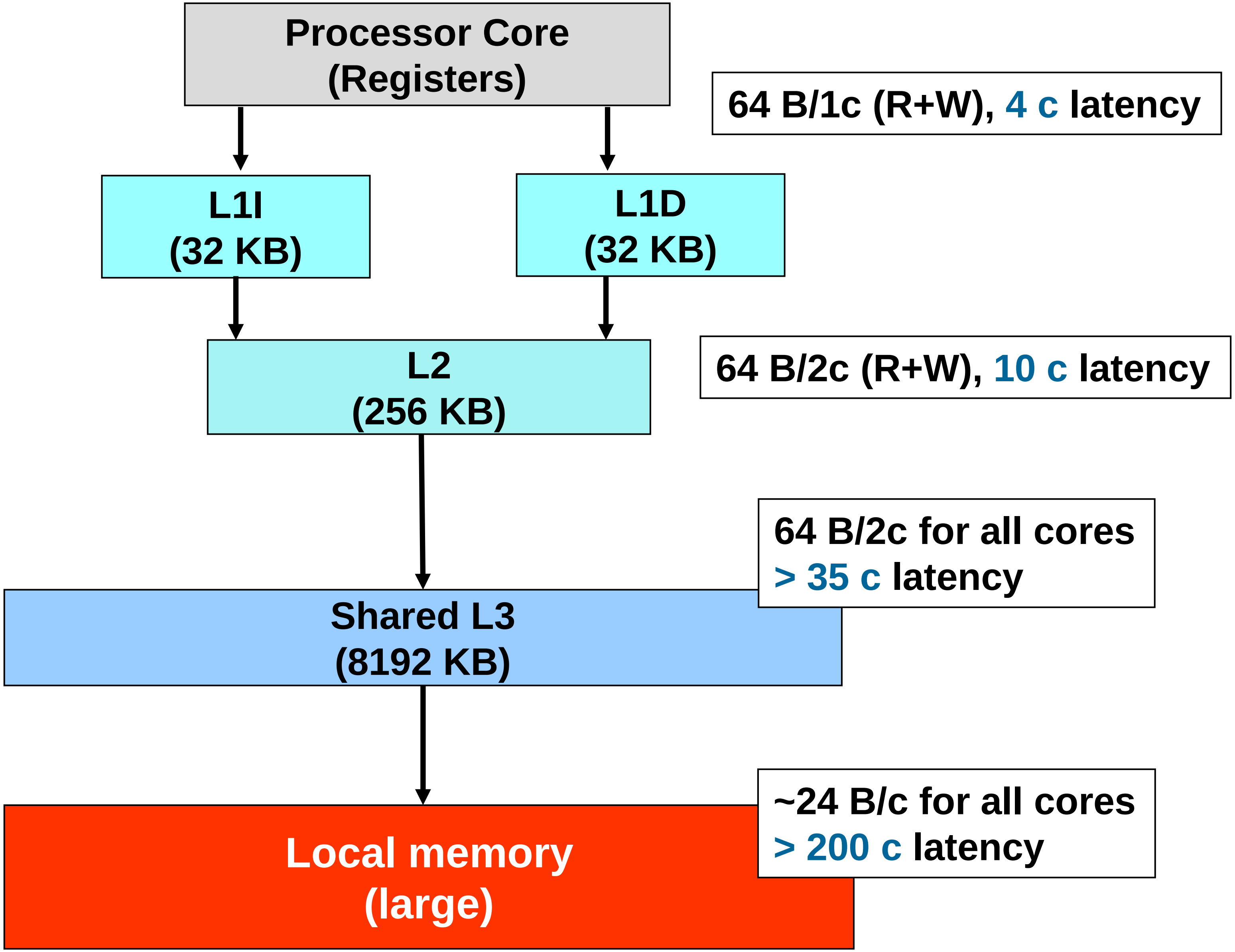


- **Intel's Haswell micro-architecture will execute <u>four</u> instructions in parallel (across <u>eight</u> ports) in each cycle.**

Source: IDF 2012

Memory Hierarchy

- **From CPU to main memory on a Nehalem processor**
    - With multicore, memory bandwidth is shared between cores in the same processor (socket)



64 B/1c (R+W), 4 c latency

64 B/2c (R+W), 10 c latency

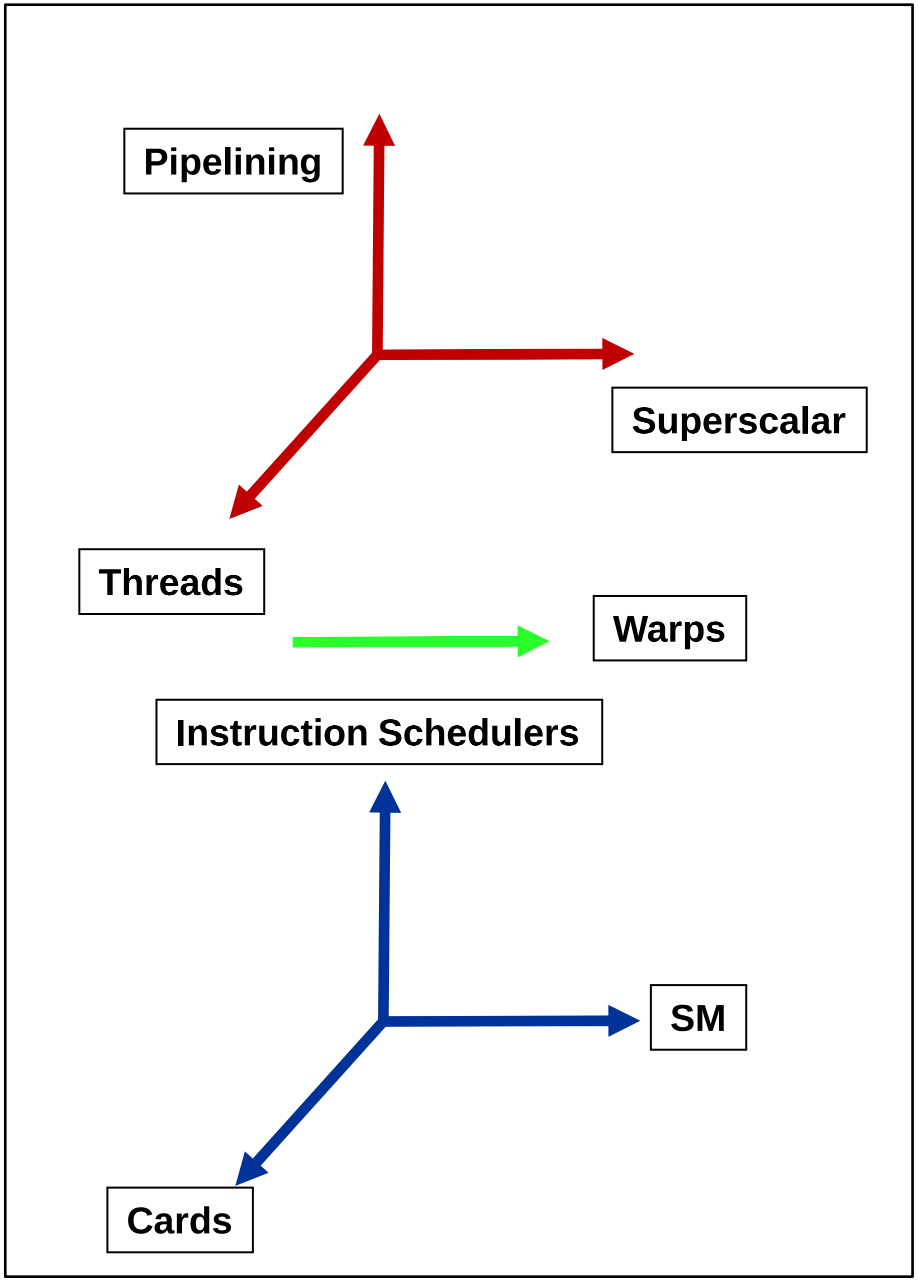
64 B/2c for all cores
> 35 c latency

~24 B/c for all cores
> 200 c latency

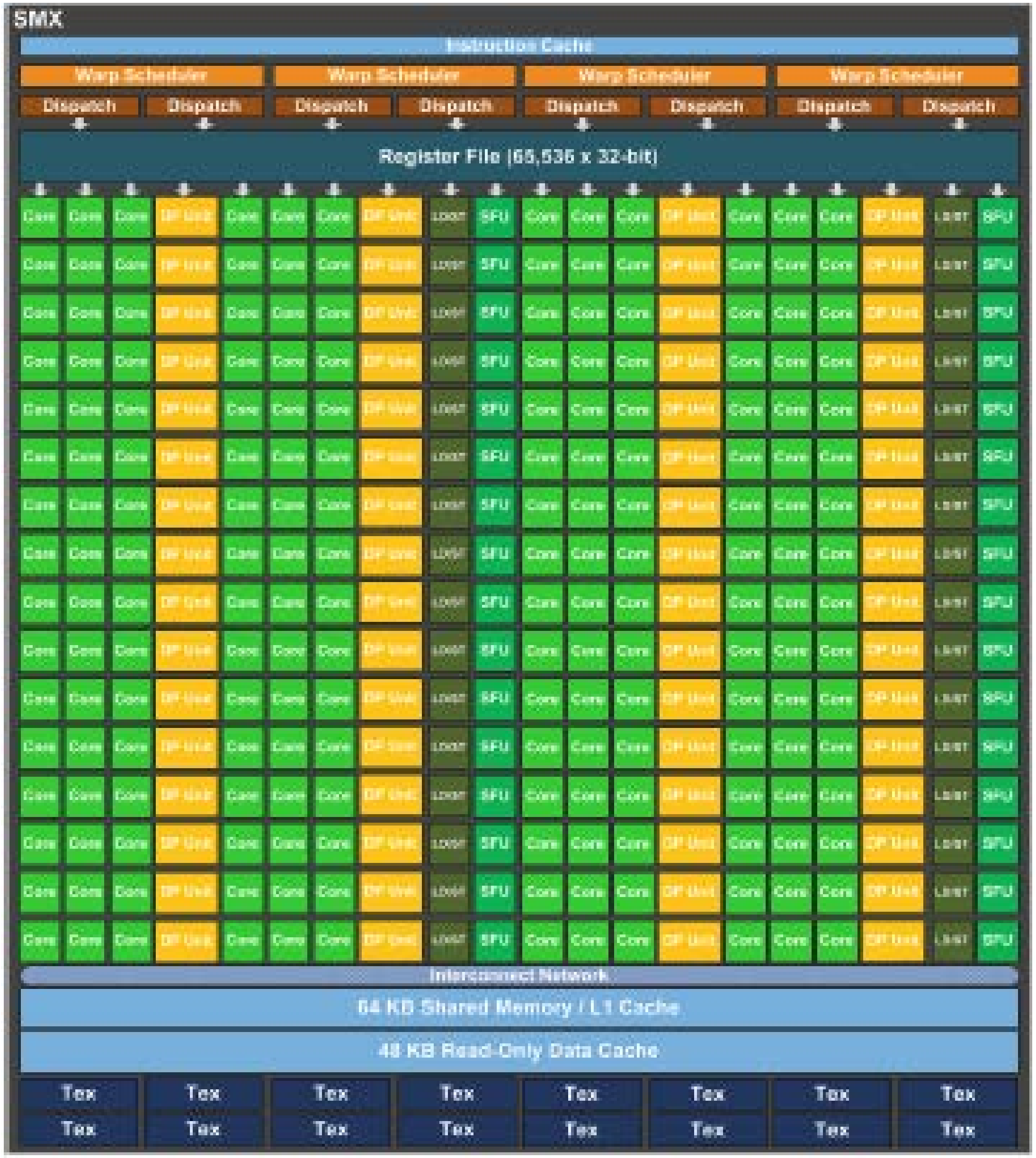
c = cycle

GPUs: 7 dimensions of performance

- **First four dimensions:**
  - Pipelining
  - Superscalar (dual issue)
  - Threads (32)
  - Instruction Scheduler (4)
- **Then, there are:**
  - Warps
- **Last dimensions:**
  - Multiple SMs
  - Multiple accelerators



Streaming Multiprocessor Architecture

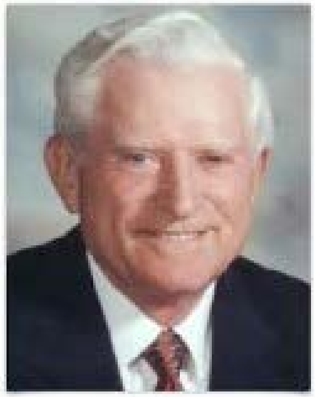


SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

Source: NVIDIA white paper

Amdahl's law

- **Maximum speedup defined by Amdahl's law**



Gene Amdahl (born: 1922)

$$\boxed{S_p^{\max}(n) = \frac{1}{1-p+\frac{p}{n}}}$$

$\boxed{\textbf{\textit{n = \#threads, p = parallel fraction}}}$

- **Three possibilities**
  - Speedup <u>less than </u>thread-count: sub-linear
  - Speedup <u>equal to</u> thread-count: linear
  - Speed-up <u>greater than</u> thread-count: super-linear

Scaled Speedup (Gustafson-Barsis's law)

- **Amdahl's law does not take into account**
  - Overhead costs
  - Natural desire to increase the problem size when computing with more cores

John L. Gustafson CalTech in 1977 (Moved from Intel to AMD in 2012)

- **Increasing the core count enables**
  - An increase of the problem size → A decrease of the sequential fraction of computation → Increased speed-up

Edwin Barsis: Director at Sandia Labs (at the time)

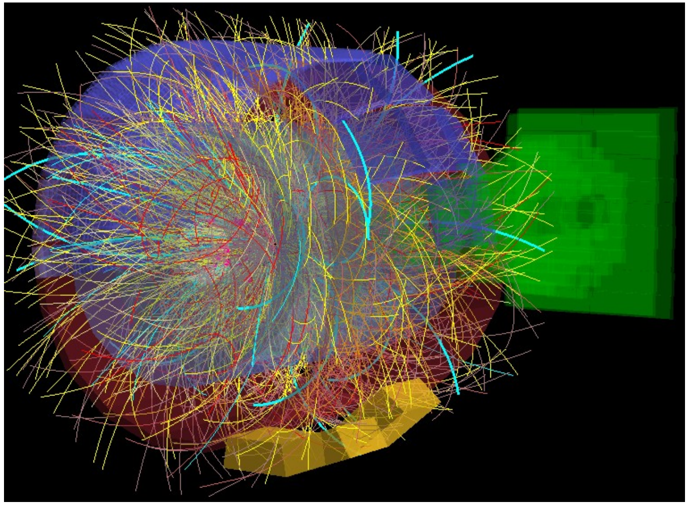
Recommendations

(based on observations in openlab)

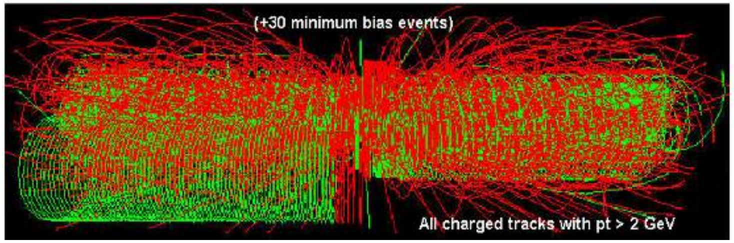
A proposal for “agile” software:

1) **Seek out parallelism at all levels**
   a. Events, tracks, vertices, etc.
   b. Perform “chunk” processing (removing event separation)
2) **Build forward scalability**
3) **Create compute-intensive kernels**
4) **Optimise data layout for locality of reference**
5) **Performance-oriented Code**
6) **Combine broad programming talents**
7) **Use best-of-breed tools**

Concurrency in High Energy Physics



- **We are "blessed" with lots of it:**
  - Entire events
  - Particles, hits, tracks and vertices
  - Physics processes
  - I/O streams (ROOT trees, branches)
  - Buffer handling (also data compaction, etc.)
  - Fitting variables
  - Partial sums, partial histograms
  - and many others …..



- **Usable for both data and task parallelism!**

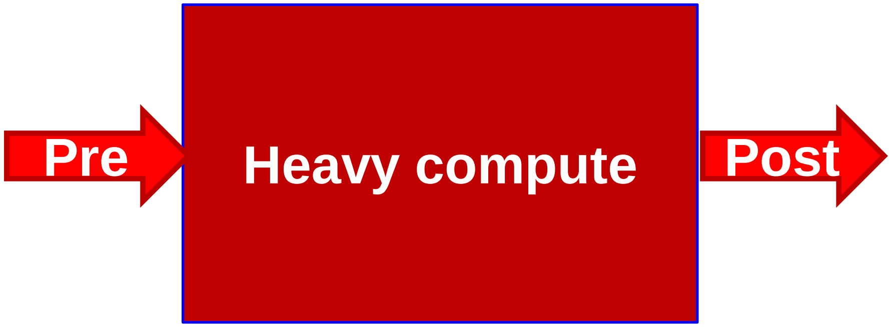
- **But, fine-grained parallelism is not well exposed in today's C++ frameworks**

The holy grail: 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!**

Kernel-oriented Programming

- **Take the whole program and its execution behaviour into account**
  - Get yourself a global overview as soon as possible
    - Via early prototyping with realistic algorithms/data
    - Influence early the design and definitely the implementation



- **Foster clear split:**
  - Prepare to compute
  - Do the heavy computation
    - In kernels, where you go after all the available parallelism
  - Post-processing

- **Often, a single kernel is not sufficient**
  - A sequence of kernels may be needed

CPU / GPU co-existence

- **What I would like to see happen to a (possibly dusty, sequential) x86 application:**
- **A strong porting effort to move it to the GPU**
  - A good “kernel-oriented design” that aims for a triple-digit speed-up
- **Then, a solid port back to the CPU servers**
  - Exploiting vectors and cores
- **Outcome:**
  - Applications that can profit from new breakthroughs on either side of the fence

CPU / GPU comparison (A case study)

- **A study presented by Robert J. Harrison, ORNL**
  - 3 years old (but approach still highly interesting)
  - Metropolis Monte Carlo (Chemistry benchmark)
- **Hardware:**
  - NVIDIA Tesla C1060 @ 1.3 GHz
    - 240 cores, 1/8 DP MADD/cycle
  - Intel Core I7 920 @ 2.67 GHz
    - Quad core, single socket, 4 DP FLOPS/cycle

1.8 : 1 ratio

- **Performance of CUDA kernel (initial port)**
  - 520x faster than Intel (CPU & compiler)

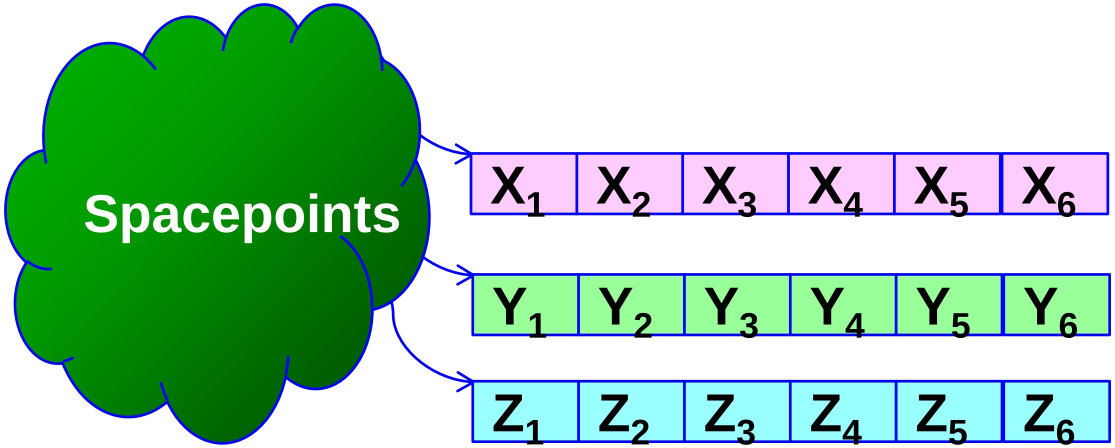
Accelerating past the petascale. A case study of GPGPUs in chemistry (R.J.Harrison, UT/ORNL, 2010)

CPU / GPU comparison (Case study – cont'd)

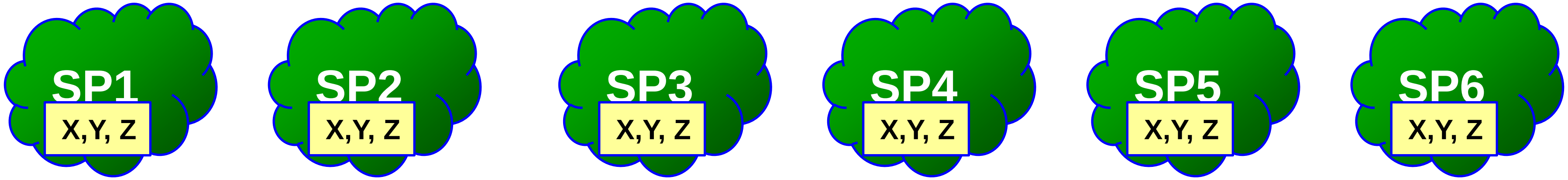
- **Second step:**
  - Go back and understand all performance dimensions of the CPU
    - In particular, get vectorisation to work
- **Bottom line:**
  - Improvement: 30x; new NVIDIA : Intel ratio (17.6x)
  - 'The optimal x86 and CUDA kernels become "identical" '
- **R.J. Harrison's conclusion:**
  - "Any credible architecture benchmark must back port the CUDA kernel to x86 and vectorise it"
    - In the name of "architectural freedom"

Data layout: SoA versus AoS

- **In general, both GPUs and CPUs prefer the former!**
- **Structure of Arrays (SoA):**


Spacepoints
X_1 X_2 X_3 X_4 X_5 X_6
Y_1 Y_2 Y_3 Y_4 Y_5 Y_6
Z_1 Z_2 Z_3 Z_4 Z_5 Z_6

- **Array of Structures (AoS):**



We need Data-Oriented Designs!

Performance-oriented code

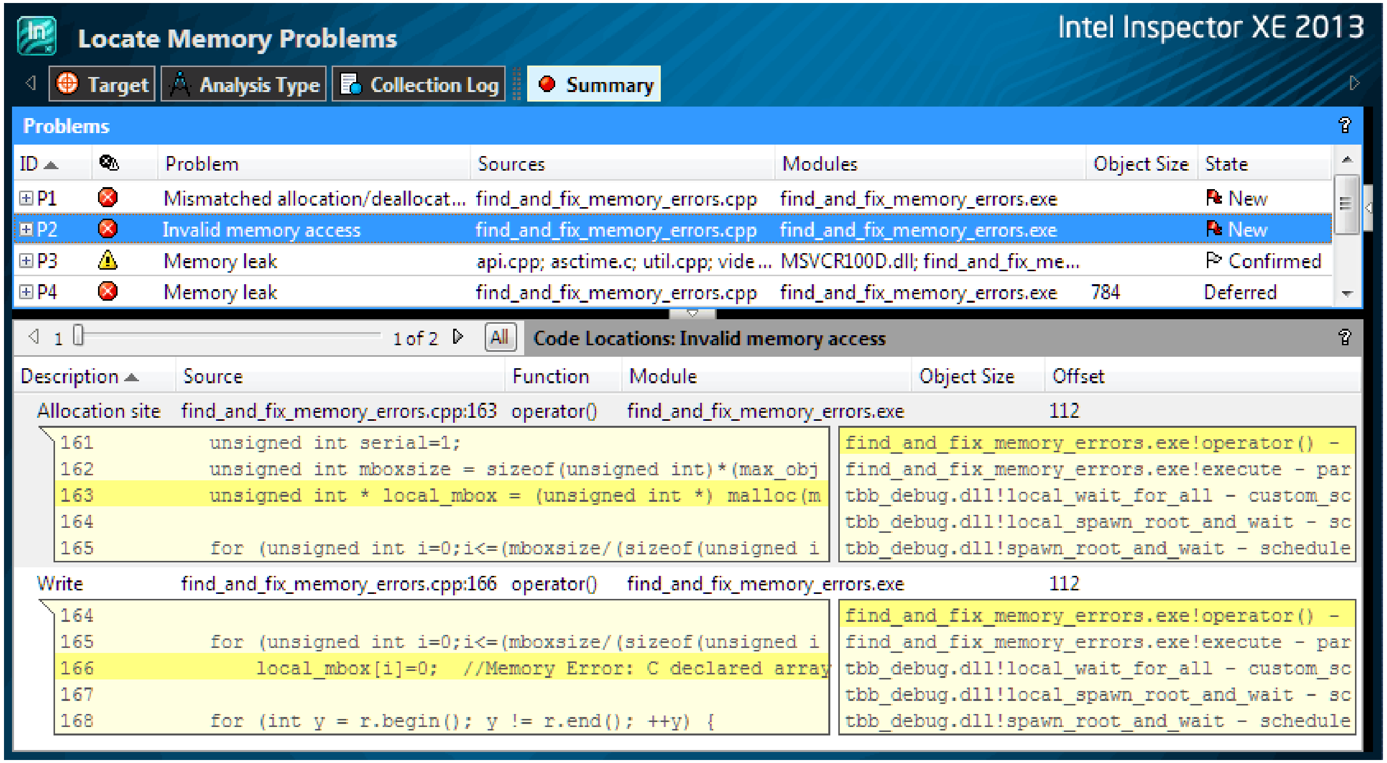
- **C++ for performance**
  - Use light-weight C++ constructs
  - Minimize virtual functions
  - Inline whenever important
  - Optimize the use of math functions
    - SQRT, DIV
    - LOG, EXP, POW
    - SIN, COS, ATAN2

Use vector libraries whenever possible

Learn to inspect the compiler-generated assembly, especially of kernels

Performance tools

- **Surround yourself with good tools:**
  - Compilers (not just one!)
  - Libraries
  - Profilers
  - Debuggers
  - Thread checkers
  - Thread profilers



Problems

ID ▲		Problem	Sources	Modules	Object Size	State
⊞ P1	⊗	Mismatched allocation/deallocat...	find_and_fix_memory_errors.cpp	find_and_fix_memory_errors.exe		New
⊞ P2	⊗	Invalid memory access	find_and_fix_memory_errors.cpp	find_and_fix_memory_errors.exe		New
⊞ P3	⚠	Memory leak	api.cpp; asctime.c; util.cpp; vide ...	MSVCR100D.dll; find_and_fix_me...		Confirmed
⊞ P4	⊗	Memory leak	find_and_fix_memory_errors.cpp	find_and_fix_memory_errors.exe	784	Deferred

1 1 of 2 All Code Locations: Invalid memory access

Description ▲	Source	Function	Module	Object Size	Offset
Allocation site	find_and_fix_memory_errors.cpp:163	operator()	find_and_fix_memory_errors.exe		112

```
161        unsigned int serial=1;
162        unsigned int mboxsize = sizeof(unsigned int)*(max_obj
163        unsigned int * local_mbox = (unsigned int *) malloc(m
164
165        for (unsigned int i=0;i<=(mboxsize/(sizeof(unsigned i
```

```
find_and_fix_memory_errors.exe!operator() -
find_and_fix_memory_errors.exe!execute - par
tbb_debug.dll!local_wait_for_all - custom_sc
tbb_debug.dll!local_spawn_root_and_wait - sc
tbb_debug.dll!spawn_root_and_wait - schedule
```

Description ▲	Source	Function	Module	Object Size	Offset
Write	find_and_fix_memory_errors.cpp:166	operator()	find_and_fix_memory_errors.exe		112

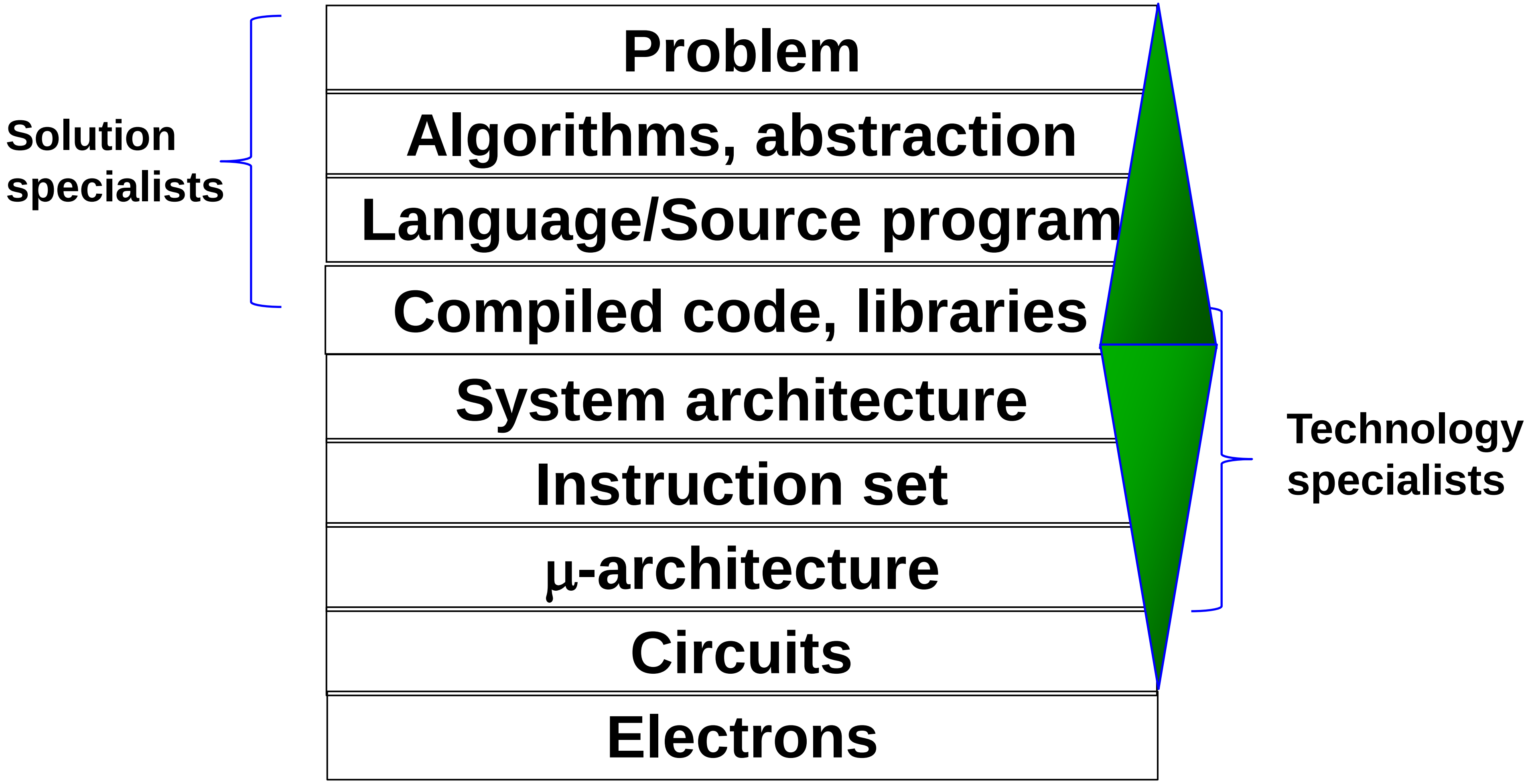
```
164
165        for (unsigned int i=0;i<=(mboxsize/(sizeof(unsigned i
166            local_mbox[i]=0;   //Memory Error: C declared array
167
168        for (int y = r.begin(); y != r.end(); ++y) {
```

```
find_and_fix_memory_errors.exe!operator() -
find_and_fix_memory_errors.exe!execute - par
tbb_debug.dll!local_wait_for_all - custom_sc
tbb_debug.dll!local_spawn_root_and_wait - sc
tbb_debug.dll!spawn_root_and_wait - schedule
```

Image: software.intel.com

Broad Programming Talent

- In order to cover as many layers as possible



Solution specialists

Problem

Algorithms, abstraction

Language/Source program

Compiled code, libraries

System architecture

Instruction set

μ-architecture

Circuits

Electrons

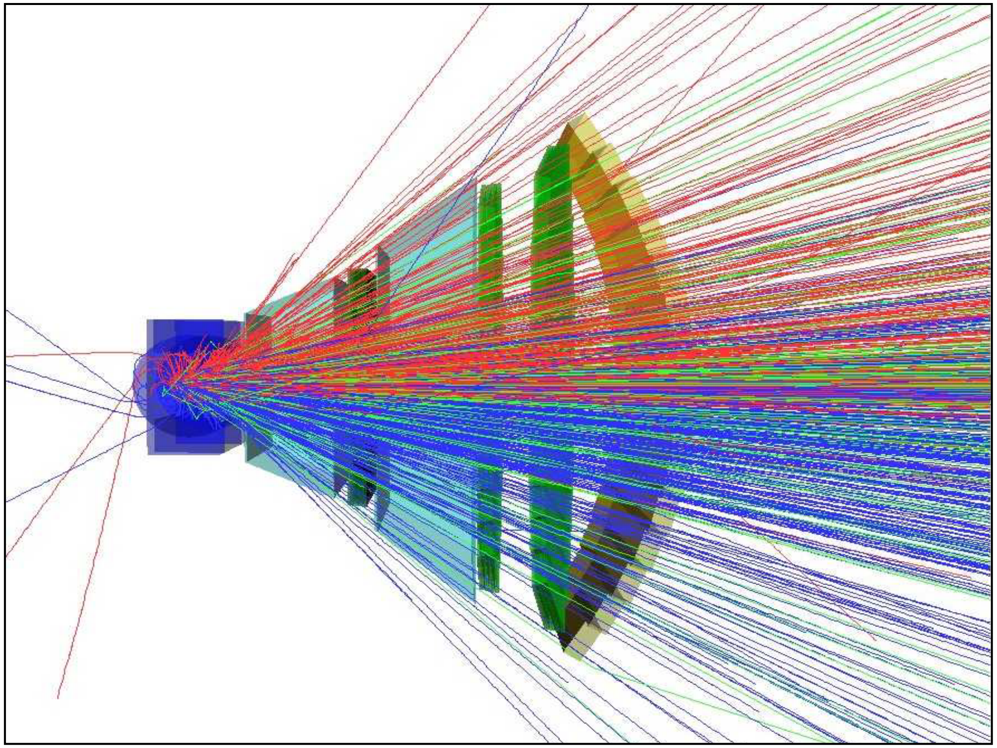
Technology specialists

HEP 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"

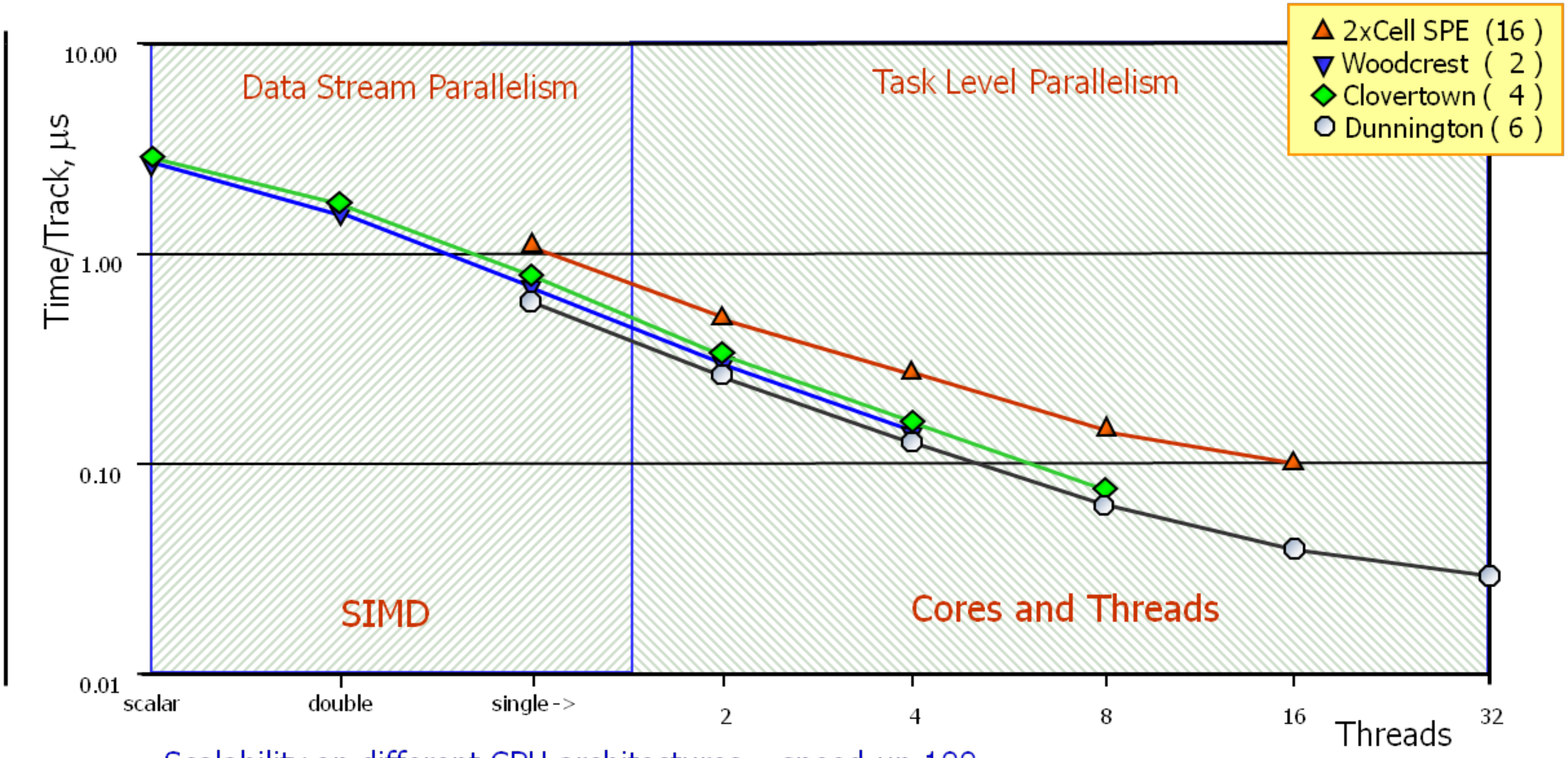
CBM/ALICE track fitting

- **Details of the re-optimization on x86-64:**
  - Part 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

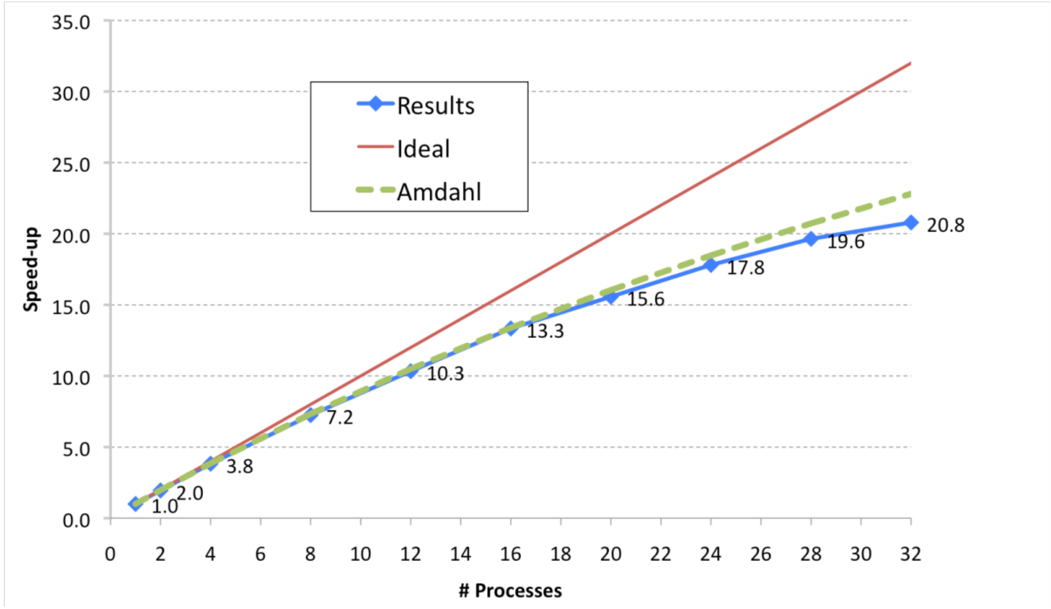


Scalability on different CPU architectures – speed-up 100

From H.Bjerke/CERN openlab, I.Kisel/GSI

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:**



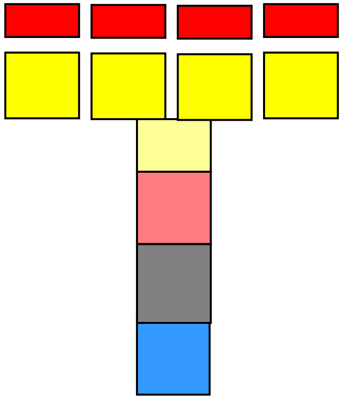
**Recent paper:
Comparison of
Software Technologies
for Vectorization and
Parallelization
(CERN openlab, 2012)**

- **In principle, we can have combinations of:**
  - vectorization (using SSE or AVX)
  - parallelization via multi-threading in a multi-core CPU
  - multiple process in a distributed computing environment

Examples of parallelism: GEANT4

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

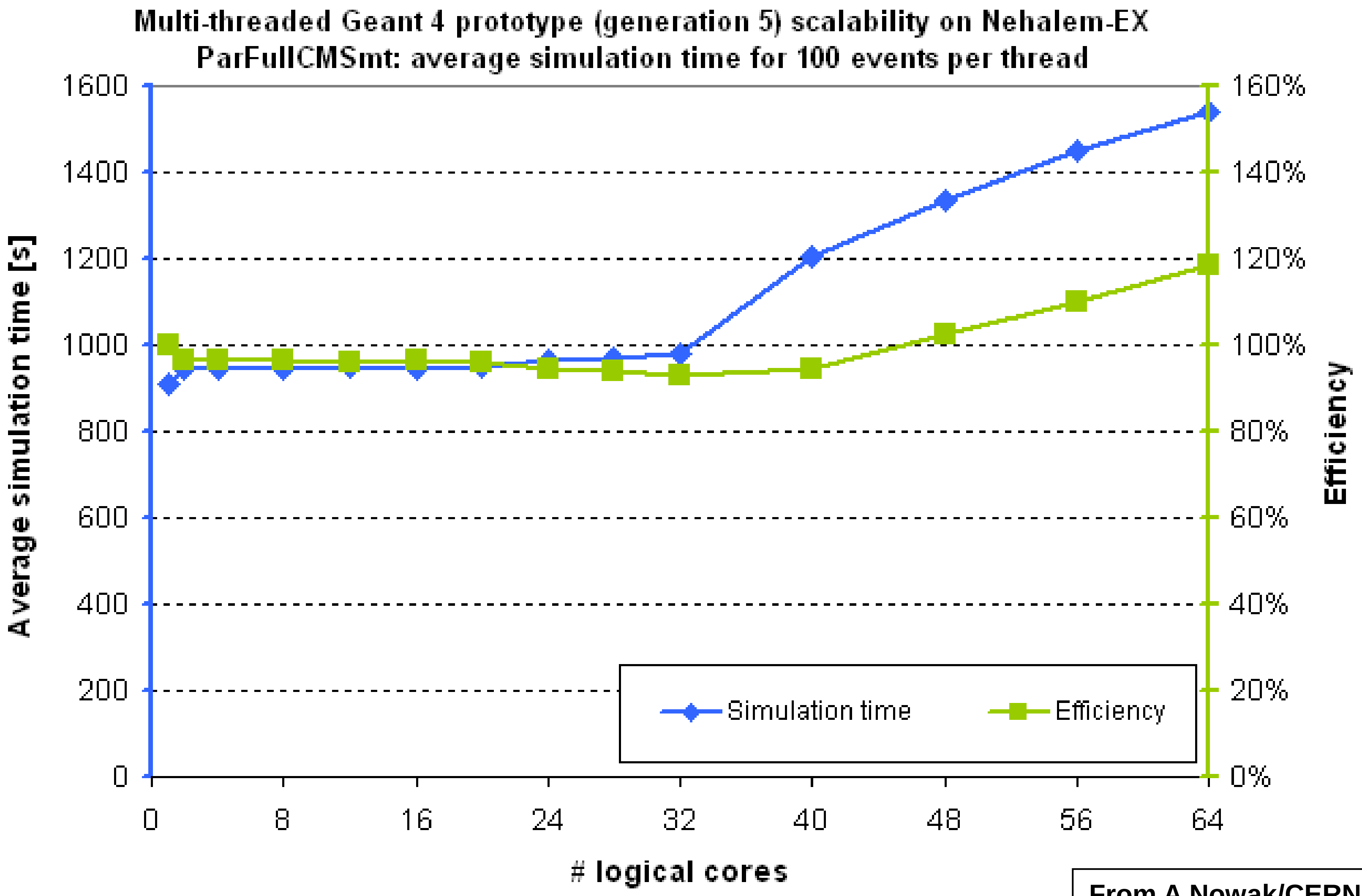
- **New prototype re-implements thread-safe event-level parallelism inside a multi-core node**
  - Done by NEU PhD student Xin Dong:
    - Using FullCMS and TestEM examples
  - Required change of lots of existing classes (10% of 1 MLOC):
    - Especially *global*, "*extrn*", and *static* declarations
    - Preprocessor used for automating the work.
  - Major reimplementation:
    - Now in separate branch in the G4 source tree



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

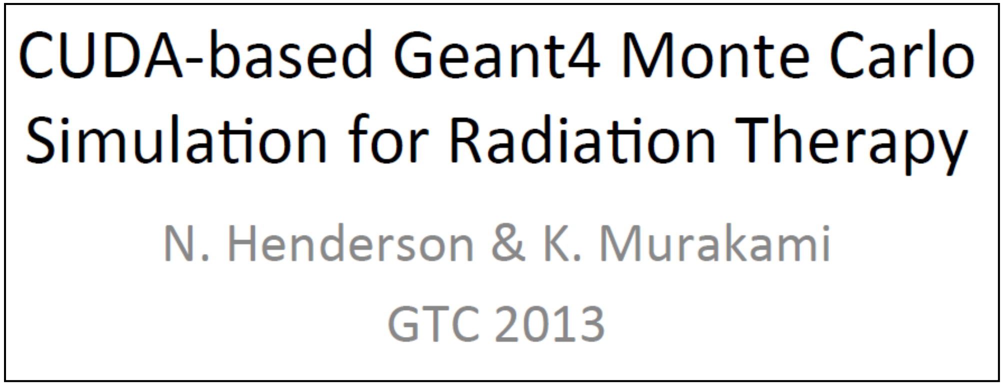
Multithreaded GEANT4 benchmark

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



From A.Nowak/CERN openlab

Geant4 in medicine (Another case study)



CUDA-based Geant4 Monte Carlo Simulation for Radiation Therapy

N. Henderson & K. Murakami

GTC 2013

SOA:

Common pattern in CUDA to allow for coalesced memory access

Experiments with transport showed this to be 3-4x faster than AOS

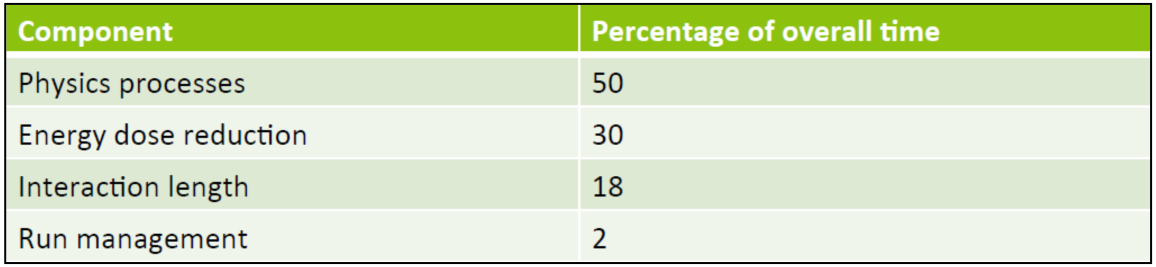
Benchmark on Tesla C2070:

100 million primary particles

Time: 72 minutes

~ 23.1 primary particles per ms

~ 50-60x speedup over Geant4 on 1 CPU



Component	Percentage of overall time
Physics processes	50
Energy dose reduction	30
Interaction length	18
Run management	2

Concluding remarks

- **Massively parallel hardware is here to stay!**
- **Our current software frameworks were not developed for such parallelism**
- **Nevertheless, in physics, we have the parallelism needed**
- **Porting to GPUs is beneficial for code redesign**
- **If you ensure that the CPU version also profits, you can have the best of both worlds!**

Thank you!