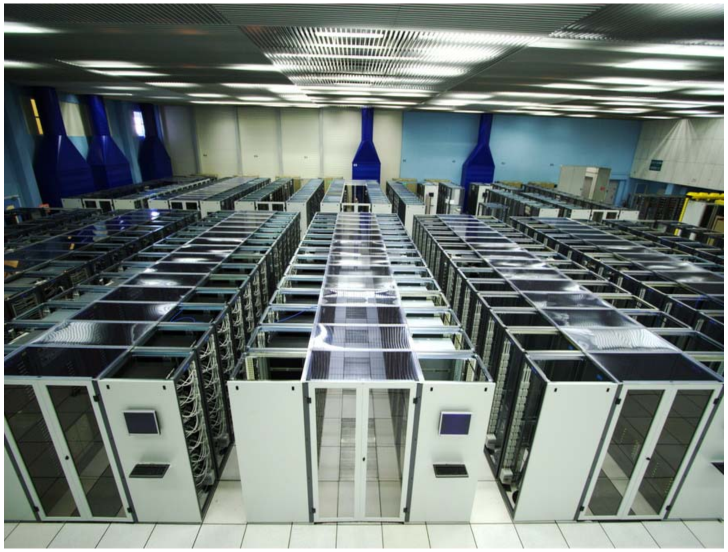
Forget multicore! The future is many-core:

An outlook to the explosion of parallelism likely to occur in the LHC era

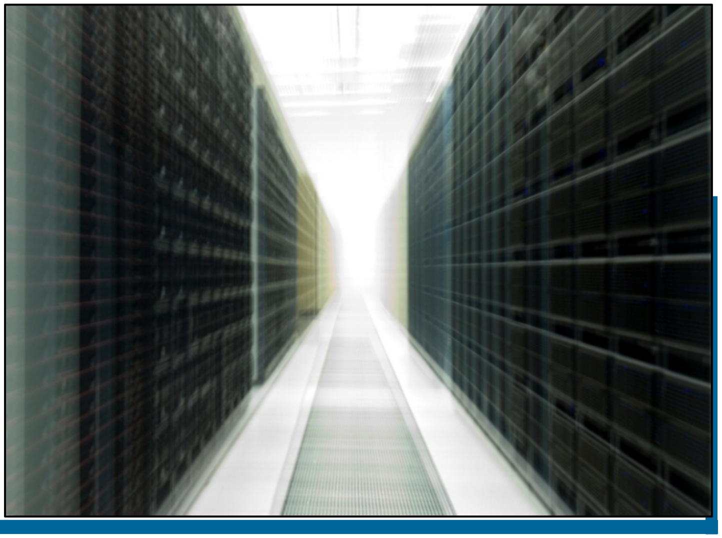


Sverre Jarp

CERN openlab

Erice – 6.11.2008

ACAT 2008



Contents

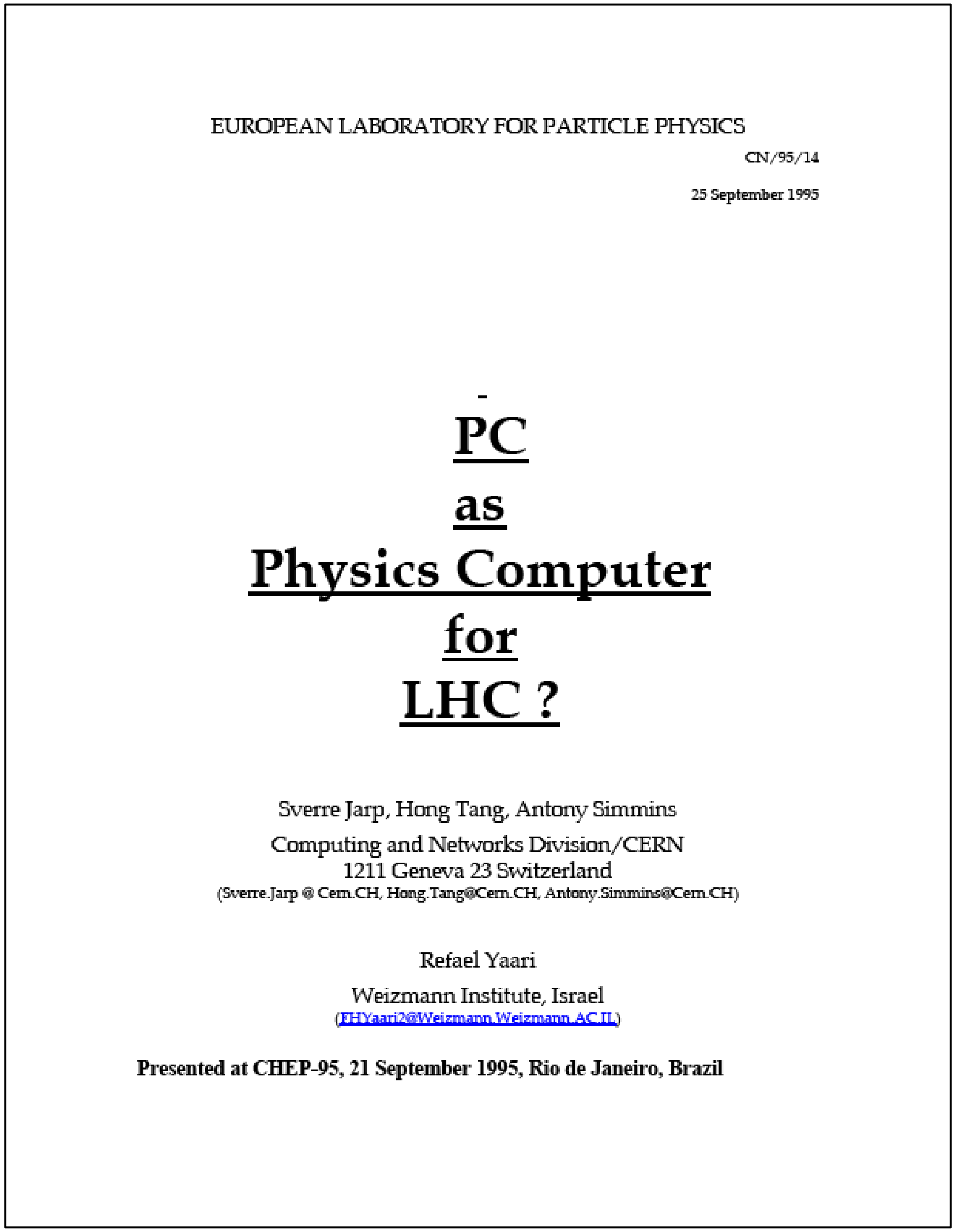
- **Uni-core**
- **Multi-core**
  - The good news
  - The not-so good news
- **Many-core**
  - The ever-increasing transistor count
  - The seven dimensions of performance
  - SW options in HEP w/encouraging examples
- **Conclusions**

We moved successfully to uni-core PCs!

Beginning of the x86 era for HEP

- At CHEP-95, I reported on the first porting and benchmarking of HEP codes (in FORTRAN)
  - CERNLIB
  - CERN benchmarks
  - GEANT3
  - ATLAS DICE (simulation)

- A few years later, PCs became ubiquitous !



EUROPEAN LABORATORY FOR PARTICLE PHYSICS

CN/95/14

25 September 1995

-

PC
as
Physics Computer
for
LHC ?

Sverre Jarp, Hong Tang, Antony Simmins
Computing and Networks Division/CERN
1211 Geneva 23 Switzerland
(Sverre.Jarp @ Cern.CH, Hong.Tang@Cern.CH, Antony.Simmins@Cern.CH)

Refael Yaari
Weizmann Institute, Israel
(FHYaari2@Weizmann.Weizmann.AC.IL)

Presented at CHEP-95, 21 September 1995, Rio de Janeiro, Brazil

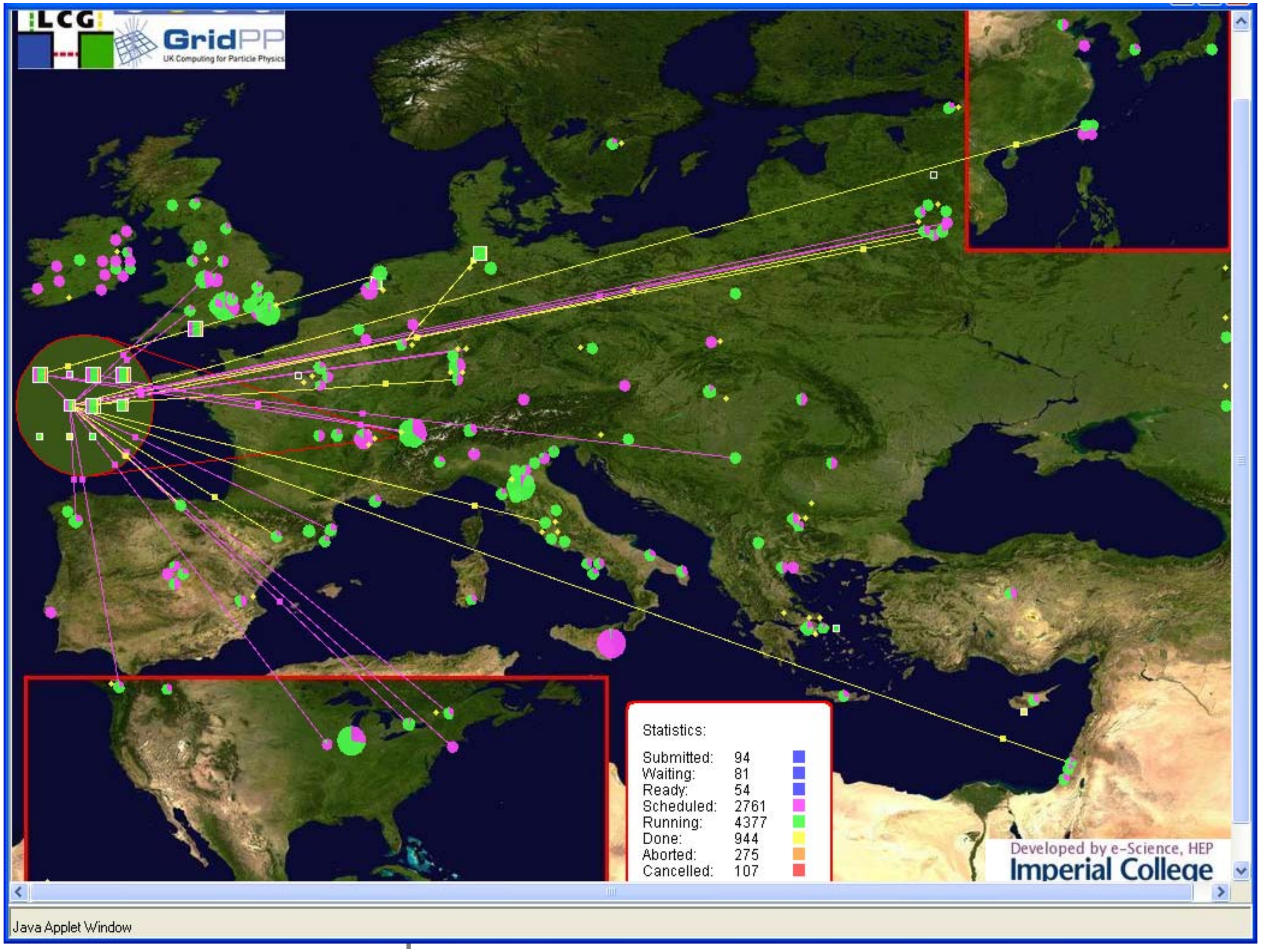
We moved successfully to multi-core!

The good multi-core news

World-wide LHC Computing Grid

- **Largest Grid service in the world !**

- **Around 140 sites in 35 countries**
- **Tens of thousands of Linux PC servers (over 100’000 cores)**
- **Tens of petabytes of storage**



LCG
GridPP
UK Computing for Particle Physics
Developed by e-Science, HEP
Imperial College
Java Applet Window

Statistics:

Submitted:	94	🟦
Waiting:	81	🟦
Ready:	54	🟦
Scheduled:	2761	🟪
Running:	4377	🟩
Done:	944	🟨
Aborted:	275	🟧
Cancelled:	107	🟥

Largest server configuration ever!

- **Latest count in the CERN Computer Centre (Aug. 2008):**

	Systems	Cores
Installed	4400	31’500
On-order	1000	6’700
Planned	950	6’000

Data from H.Meinhard/HEPix/Oct. 2008

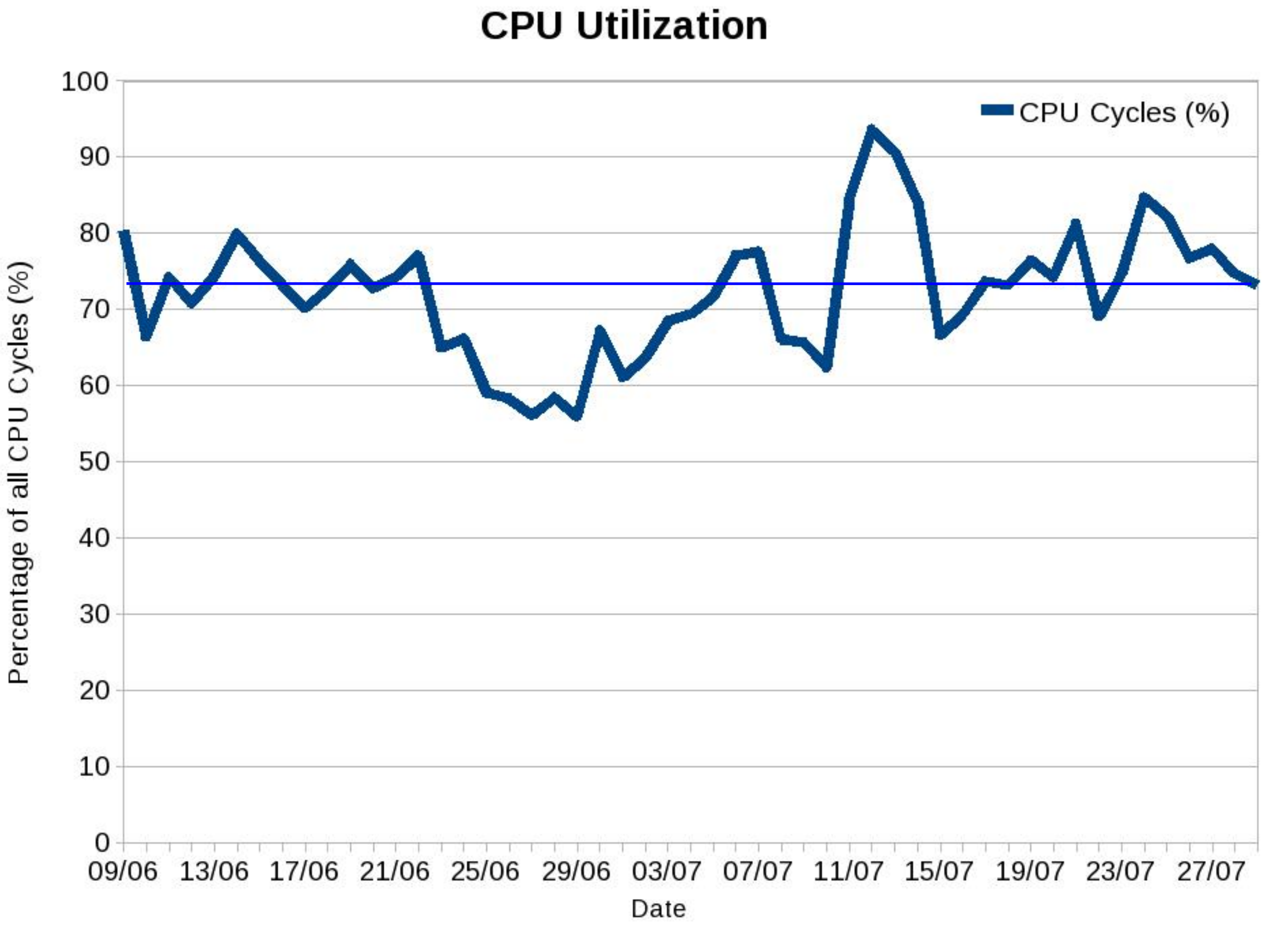
Thermal efficiency has soared !

- **When going from uni-core to quad-core servers:**

Date	Server	Power efficiency	Power efficiency/GHz
Late 05	DP Irvindale, 2.8GHz, 4GB, 1 disk, 7320 chipset	6.63	2.37
Early 07	DP Woodcrest, 3.0GHz, 8 * 1GB, 1 disk, 5000P, 1U	22.3	7.43
Late 08 (not yet installed)	DP Harpertown/L, 2.5GHz, 4 * 4GB, 2 disks, 5100, twin	57.0	22.8

Data from H.Meinhard/HEPix/Oct. 2008

Quad-core CPU utilization



Measurements:

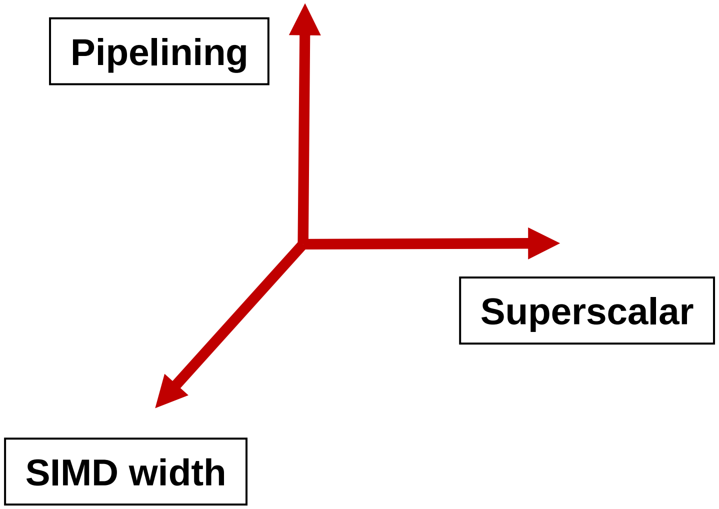
59 LSF production servers during two months (Balazs/Hirstius, CERN openlab)

- **CPU cycles consumed (User + System):**
  - Peak: 93%, Trough: 55%, Average: 72.3%

The not so good multi-core news

We neglect several performance dimensions

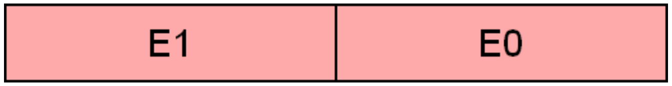
- **First three dimensions:**
  - Superscalar
  - Pipelining
  - Computational width/SIMD



- **We extract only 10-15% of peak execution capability !**

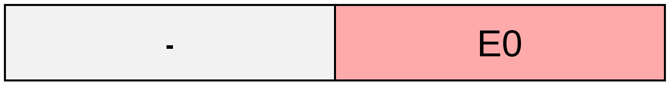
In HEP (offline), we are vectorless !

- **Today:**
- **XMM registers are 128 bits**
  - Each operation (ADD, MUL, etc.) can operate on two 64-bit doubles



E1	E0

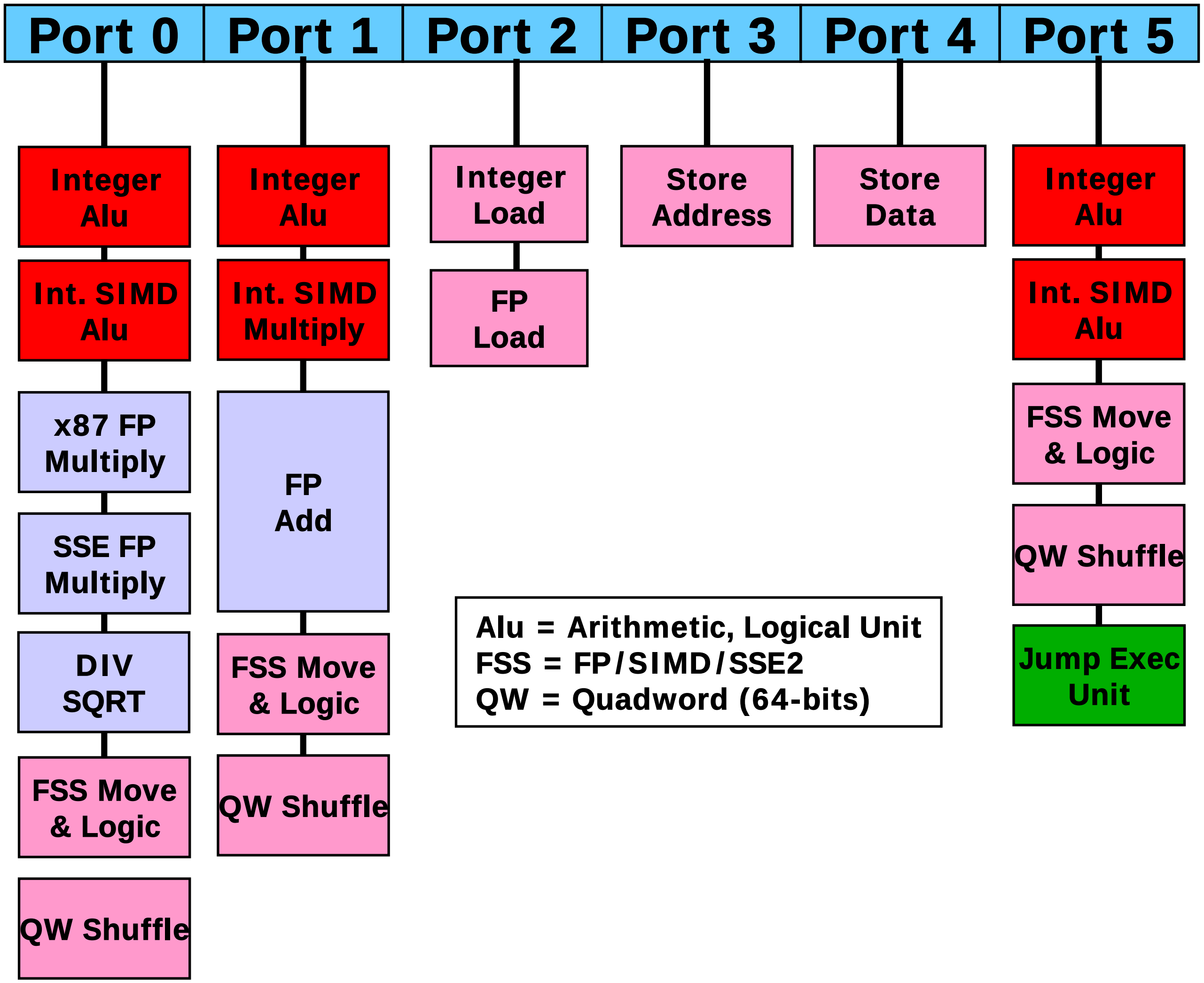
- **We run at half speed!**
  - Each operation done on one double, not two.



-	E0

Core 2 execution ports

- **Intel’s Core microarchitecture can handle:**
  - Four instructions in parallel:
  - Every cycle
  - Data width of 128 bits



Alu = Arithmetic, Logical Unit
FSS = FP/SIMD/SSE2
QW = Quadword (64-bits)

Issue ports in the Core 2 micro-architecture (from Intel Manual No. 248966-016)

HEP code density

- **Averages about 1 instruction per cycle.**
  - This “extreme” example shows even less:

High level C++ code →

```
if (abs(point[0] - origin[0]) > xhalfsz) return FALSE;
```

Assembler instructions →

```
movsd 16(%rsi), %xmm0
subsd 48(%rdi), %xmm0   // load & subtract
andpd _2il0floatpacket.1(%rip), %xmm0 // and with a mask
comisd 24(%rdi), %xmm0 // load and compare
jbe ..B5.3      # Prob 43% // jump if FALSE
```

Same instructions laid out according to latencies on the Core 2 processor →

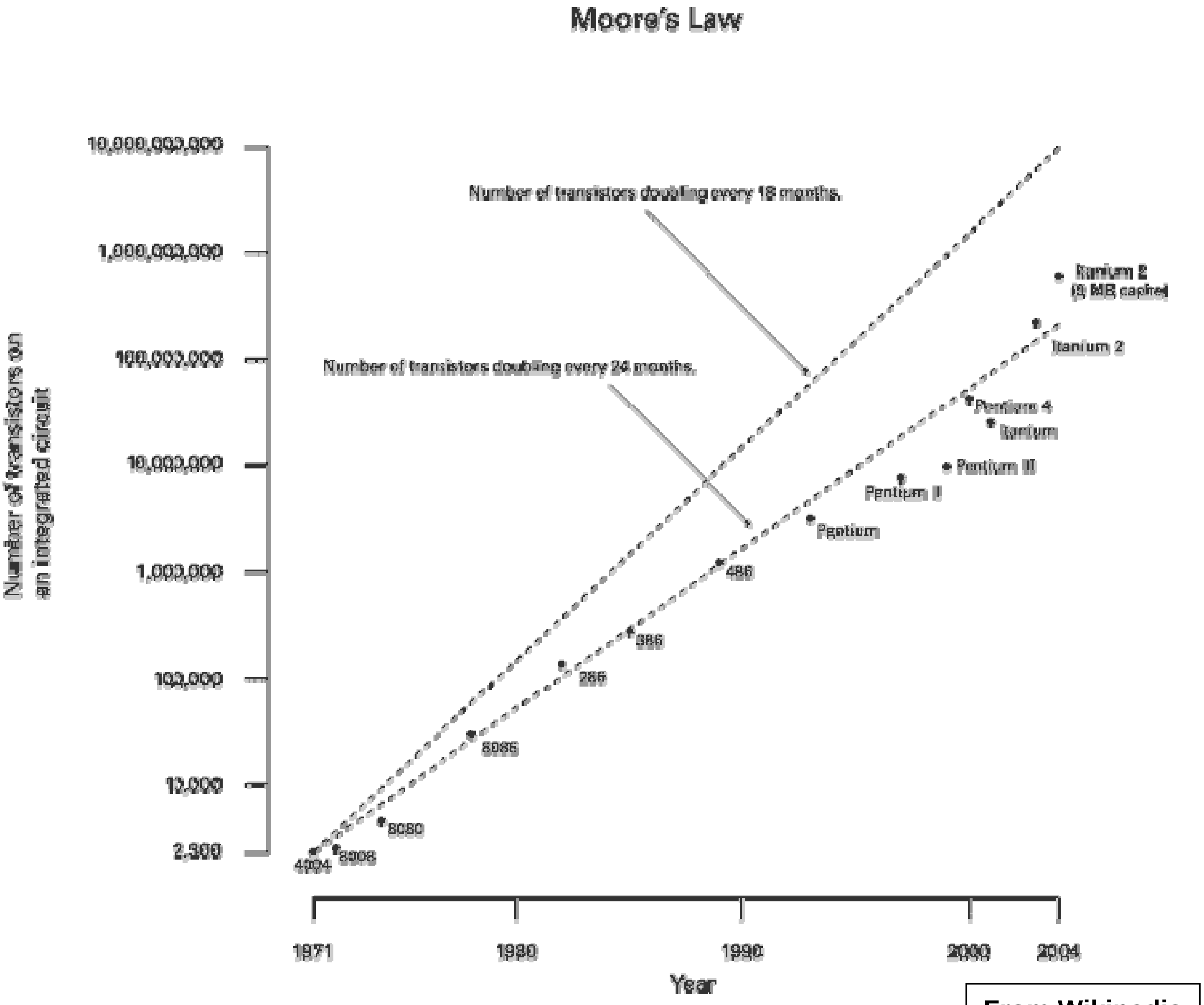
NB: Out-of-order scheduling not taken into account.

Cycle	Port 0	Port 1	Port 2	Port 3	Port 4	Port 5
1			load point[0]			
2			load origin[0]			
3						
4						
5						
6		subsd	load float-packet			
7						
8			load xhalfsz			
9						
10	andpd					
11						
12	comisd					
13						jbe

We want to move successfully to many-core!

The driving force: Moore's law

- **The industry continues to double the number of transistors**
  - Consequence
    - Single core → Multicore → Manycore



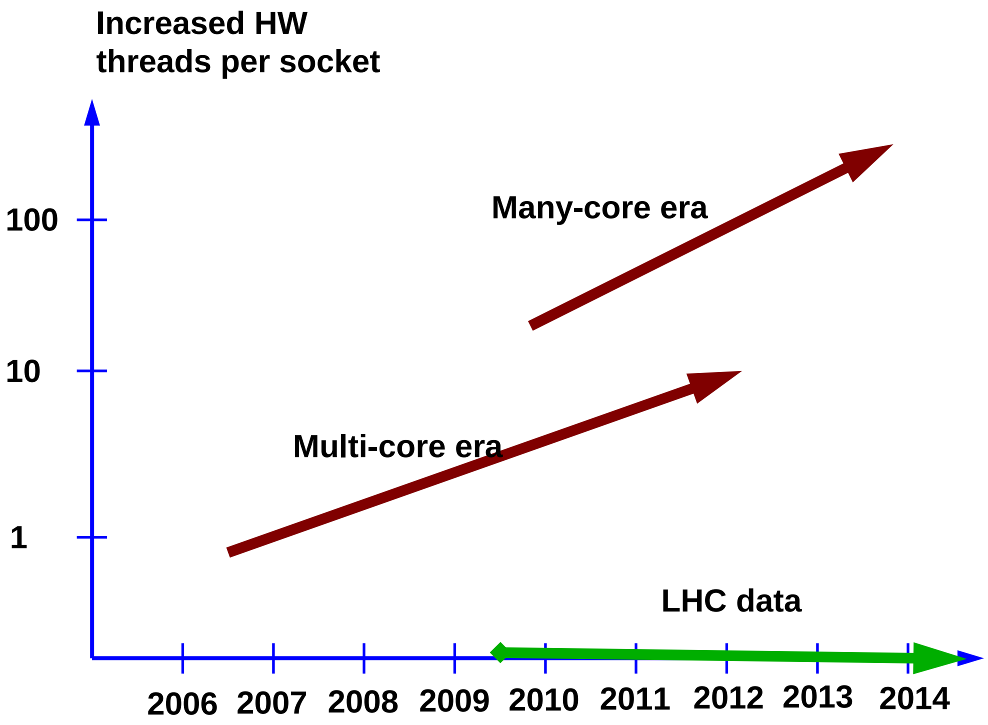
From Wikipedia

Real consequence of Moore’s law

- **We are being “run over” by transistors:**
  - More (and more complex) execution units
  - Longer SIMD/SSE vectors
  - More hardware threading
  - More cores

Intel platform 2015 (and beyond)

- **Today: 45 nm**
- **Already on the roadmap:**
  - 32 nm (2009/10)
  - 22 nm (2011/12)
- **In research:**
  - 16 nm (2013/14)
  - 11 nm (2015/16)
  - 8 nm (2017/18)
- **Source: Bill Camp/Intel HPC**

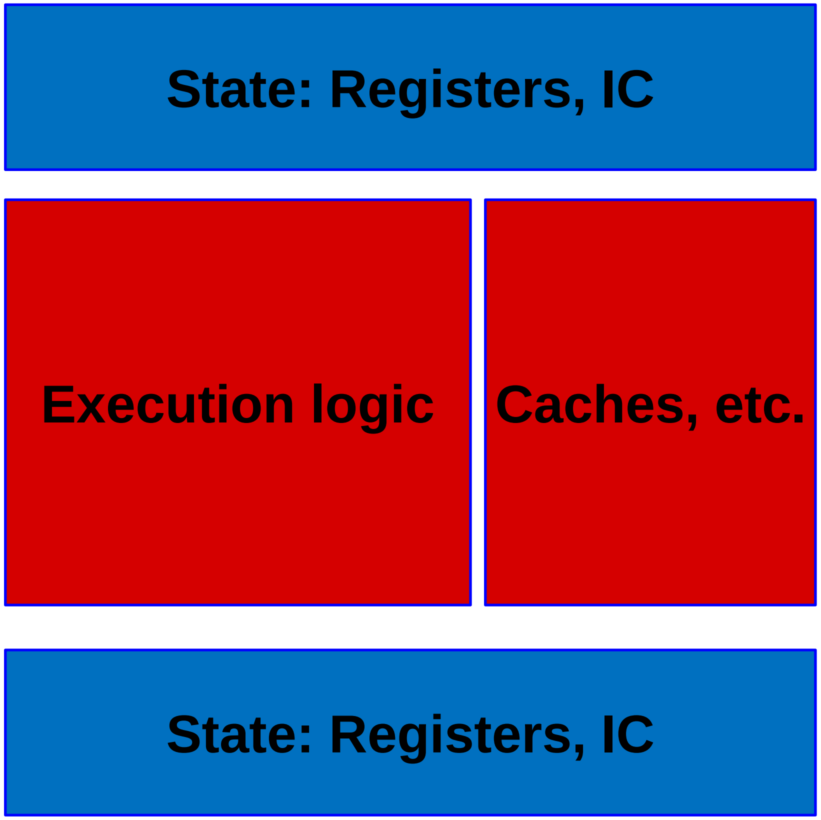


From “Platform 2015: Intel Platform Evolution for the Next Decade” (S.Borkar et al./Intel Corp.)

- **And each generation will push the processor count!**

Definition of a hardware core/thread

- **Core**
  - A complete ensemble of execution logic and cache storage (etc.) as well as register files plus instruction counter (IC) for executing a software process or thread
- **Hardware thread**
  - Addition of a set of register files plus IC
- **Both appear as CPUs**
  - cat /proc/cpuinfo


State: Registers, IC

Execution logic

Caches, etc.

State: Registers, IC

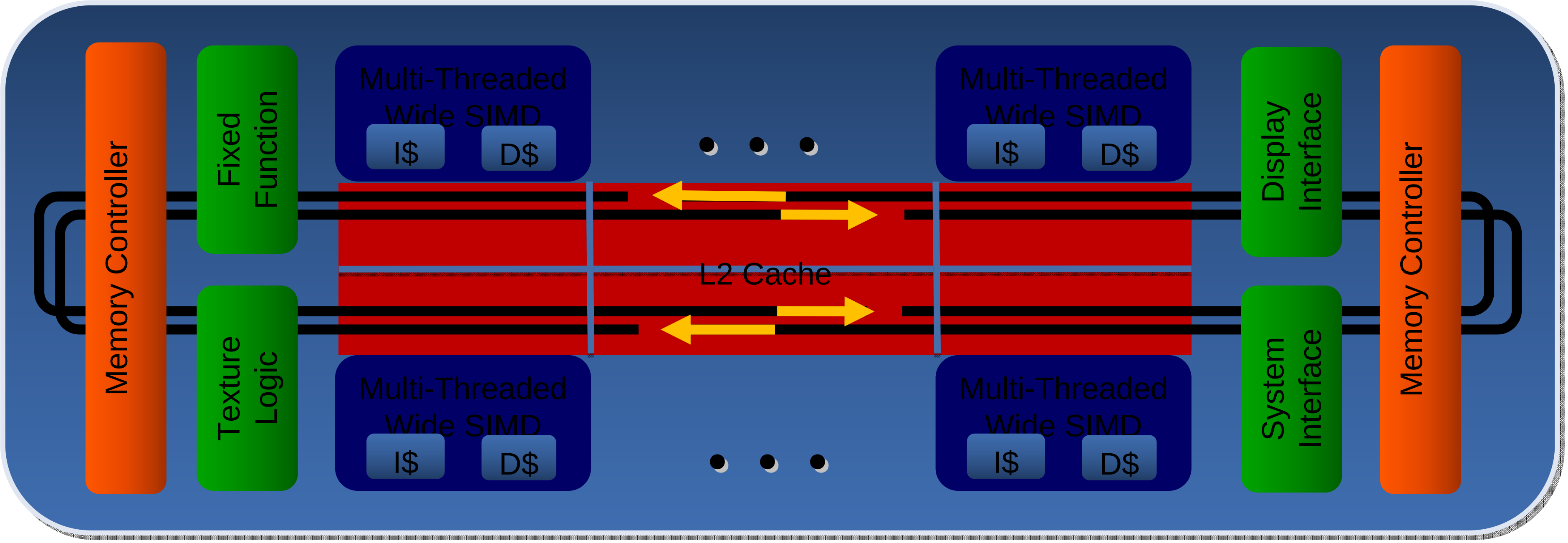
The sharing of the execution logic can be coarse-grained or fine-grained.

The appearance of many-core systems

- **Let's define "dispatch slots": sockets * cores * hw-threads**
  - The total of what you see in /proc/cpuinfo!
  - Conservative:
    - Dual-socket Intel quad-core Harpertown: 2 * 4 * 1 = 8
    - Dual-socket Intel quad-core Nehalem: 2 * 4 * 2 = 16
  - Aggressive:
    - Quad-socket Nehalem "octo-core": 4 * 8 * 2 = 64
    - Quad-socket Sun Niagara (T2+) processors w/8 cores and 8 threads: 4 * 8 * 8 = 256
- **Now, or in the near future: Hundreds of "dispatch slots" !**
- **And, by the time, new software is ready: Thousands !!**

Many-core graphics processor

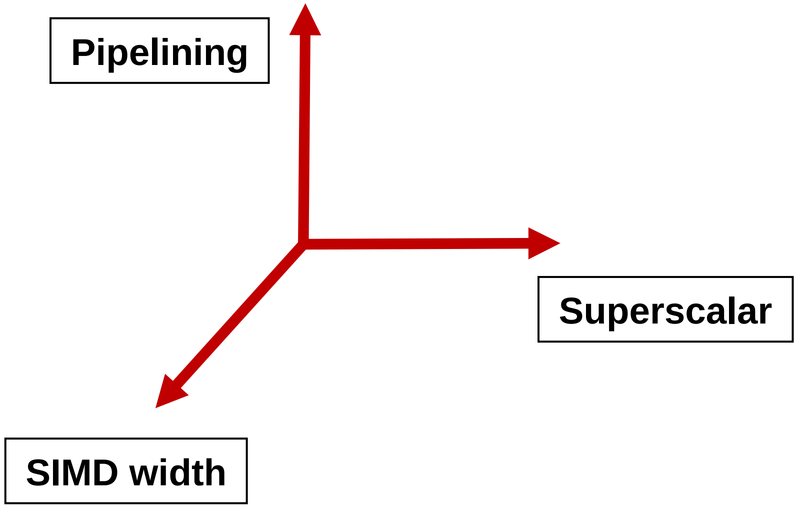
- **Intel's Larrabee:**
  - Already announced at SigGraph 2008!
  - Fully IA (x86-based)
  - Many-core + Multithreaded + vector unit



- **Not forgetting offerings from NVidea, AMD, IBM, etc.**

Seven dimensions of performance

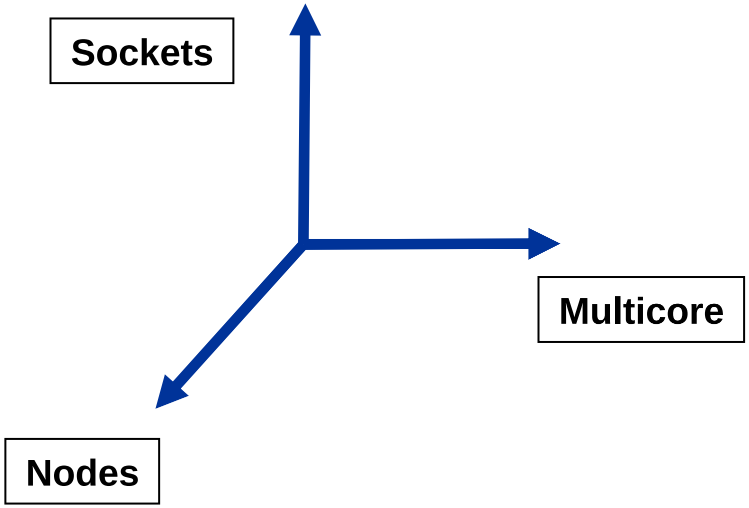
- **First three dimensions:**
  - Superscalar
  - Pipelining
  - Computational width/SIMD
- **Next dimension is a "pseudo" dimension:**
  - Hardware multithreading
- **Last three dimensions:**
  - Multiple cores
  - Multiple sockets
  - Multiple compute nodes



Pipelining

Superscalar

SIMD width



Sockets

Multicore

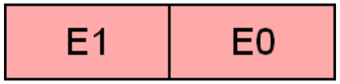
Nodes

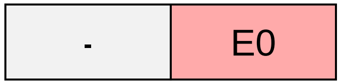
SIMD = Single Instruction Multiple Data

The worry of remaining vectorless !

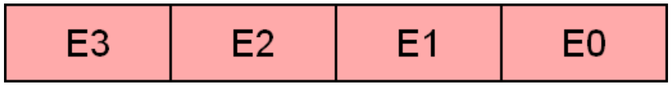
- **Today:**

- **We run at half speed**
  - Each operation done on one double, not two.

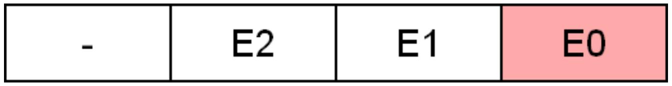

E1 E0


- E0

- **Intel has announced AVX w/256bits:**
  - Advanced Vector Extensions


E3 E2 E1 E0

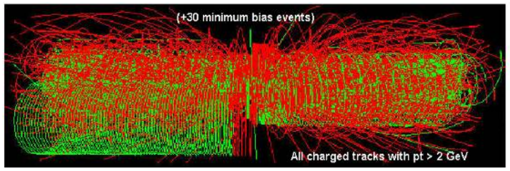
- **In two years' time: ¼**
  - Operating on one double, not four!


- E2 E1 E0

- **And even longer vector will follow.**
  - And we will use: 1/8, 1/16, etc.

Our programming paradigm

- **Event-level parallelism has been used for decades**
  - Process event-by-event in a single process



- **Advantage**
  - Large jobs can be split into N efficient processes, each responsible for processing M events
    - Built-in scalability
- **Disadvantage**
  - Memory must be made available to each process
    - With 2 – 4 GB per process
    - A dual-socket server with Quad-core processors
      - Needs 16 – 32 GB (or more) – we currently buy only 16!

What are the options?

- **There is currently a discussion in the community about the best way forwards (in a many-core world):**

1) Stay with event-level parallelism (and independent processes)
   - Assume that the necessary memory remains affordable
2) Rely on forking:
   - Start the first process
   - Fork N others
   - Rely on the OS to do “copy on write”, in case pages are modified
3) Move to a fully multi-threaded paradigm
   - Using gross-grained (event-level?) parallelism

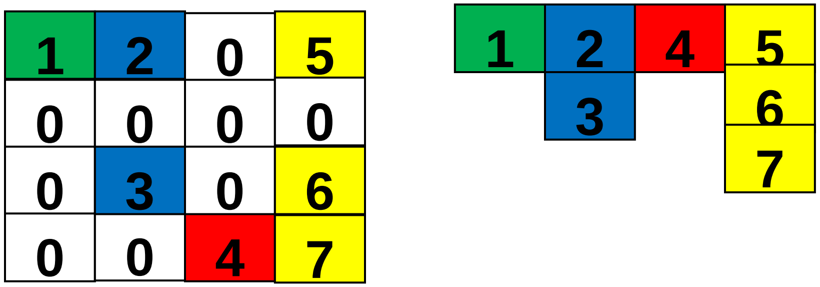
C++ multithreading support

- **Beyond auto-vectorization/auto-parallelization,**
- **Large selection of low-level tools:**
  - OpenMP
  - MPI
  - pthreads/Windows threads
  - Threading Building Blocks (TBB)
  - TOP-C (from NE University)
  - RapidMind
  - Ct (C-throughput)
  - etc.

Ct Language

See talk by A.Ghuloum/Intel

- **New effort by Intel to extend C++ for Throughput Computing**
- **Features:**
  - Addition of new data types (parallel vectors) & operators
    - NeSL/SASAL-inspired: irregularly nested and sparse/indexed vectors
  - Abstracting away architectural details
    - Vector width/Core count/Memory Model: Virtual Intel Platform
      - Forward-scaling (Future-proof!)
    - Nested data parallelism and deterministic task parallelism
- **Incremental adoption path:**
  - Dedicated Ct-enabled libraries
  - Rewritten “kernels” in Ct
  - Pervasive use of Ct



1	2	0	5
0	0	0	0
0	3	0	6
0	0	4	7

1	2	4	5
	3		6
			7

Rethink concurrency in HEP

- **We are “blessed” with lots of it:**
  - Events
  - Particles, tracks and vertices
  - Physics processes
  - I/O streams
  - Buffer manipulations (also compaction, etc.)
  - Fitting and minimization
  - Partial sums, partial histograms
  - and many others …..

But remember: Multithreading is not easy!

- **Several new concepts to master:**
  - Concurrency
  - Decomposition
  - Mapping
  - Communication
  - Synchronization
  - Keeping in mind Amdahl's law:

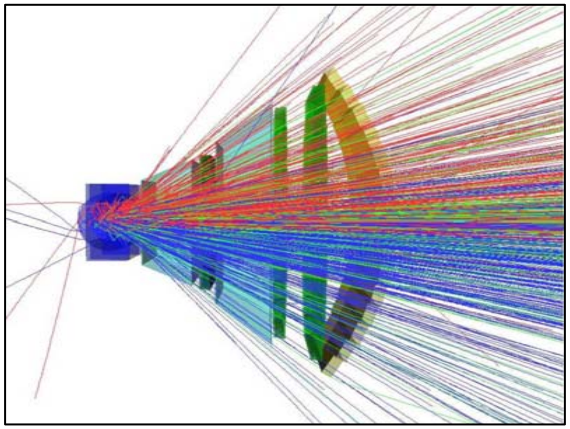
$$S_p^{\max}(n) = \frac{1}{1-p+\frac{p}{n}}$$

On the other hand, the Blue Brain talk reminded us that other disciplines can scale into the thousands!

Examples of parallelism: CBM track fitting

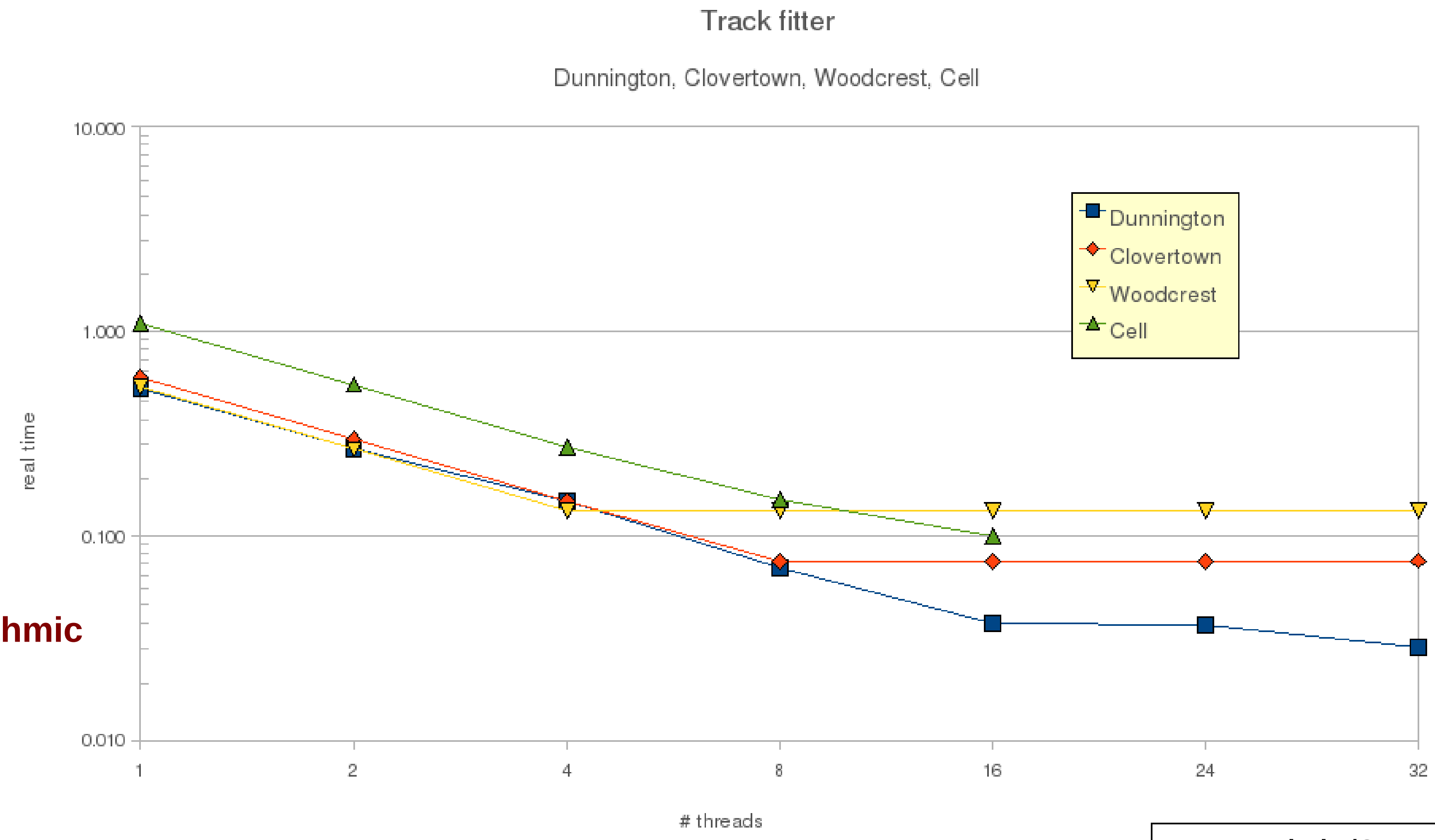
See talk by Ivan Kisel

- **Extracted from CBM's High Level Trigger Code**
  - Originally ported to IBM's Cell processor
- **Tracing particles in a magnetic field**
  - Embarrassingly parallel code
- **Re-optimization on Intel Core systems**
  - Step 1: use SSE vectors instead of scalars
    - Operator overloading allows seamless change of data types, even between primitives (e.g. float) and classes
    - Two classes
      - P4_F32vec4 – packed single; operator + = _mm_add_ps
      - P4_F64vec2 – packed double; operator + = _mm_add_pd
  - Step 2: add multithreading (via TBB)
    - Enable scaling with core count



CBM HLT benchmark runs

- **Real fit time/track (µs) as a function of the core count:**



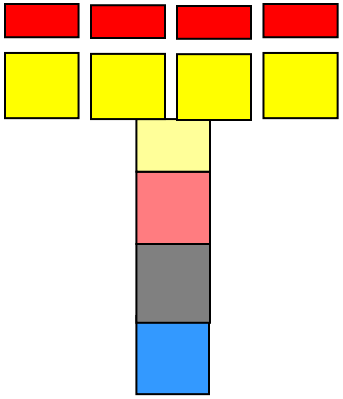
Logarithmic scale!

From H.Bjerke/CERN openlab

Examples of parallelism: GEANT4

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

- **New development re-implements thread-safe event-level parallelism inside a multi-core node**
  - Done by NEU PhD student Xin Dong: Using FullCMS example
  - Required change of lots of existing classes:
    - Especially *global*, "*extrn*", and *static* declarations
  - The geometry was converted (this summer)
  - Latest news:
    - Physics tables are now in the process of being shared

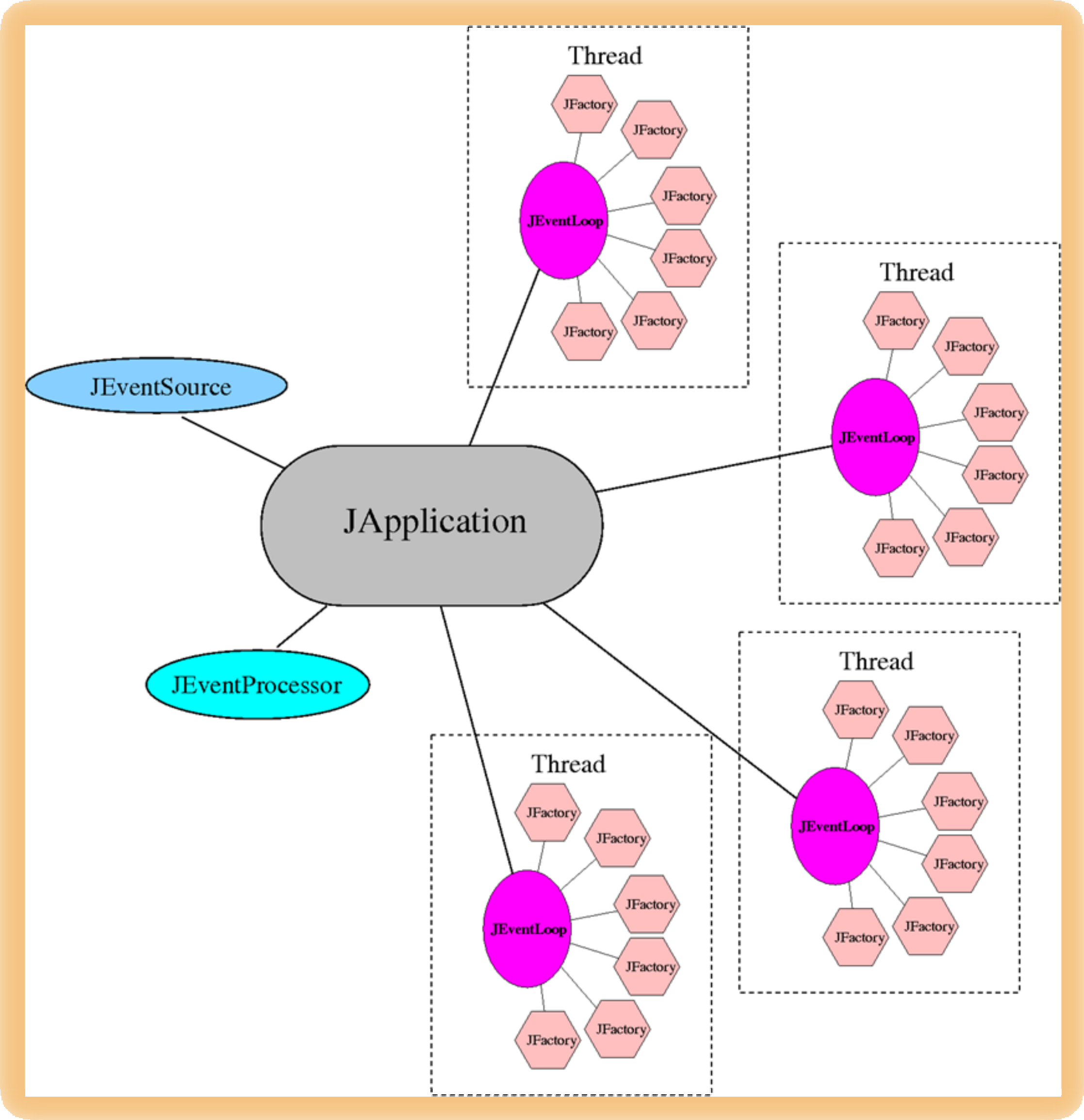


- **Additional memory: Only 22MB/thread (!)**

More work is needed, but extremely interesting first steps!

Examples of parallelism: JANA

- Each thread in JANA is composed of its own event processing loop and a complete set of factories
- Reconstruction of a given event is done entirely inside of a single thread
- No mutex locking is required by authors of reconstruction code
- Threads work asynchronously to maximize rates at the expense of not maintaining the event order on output

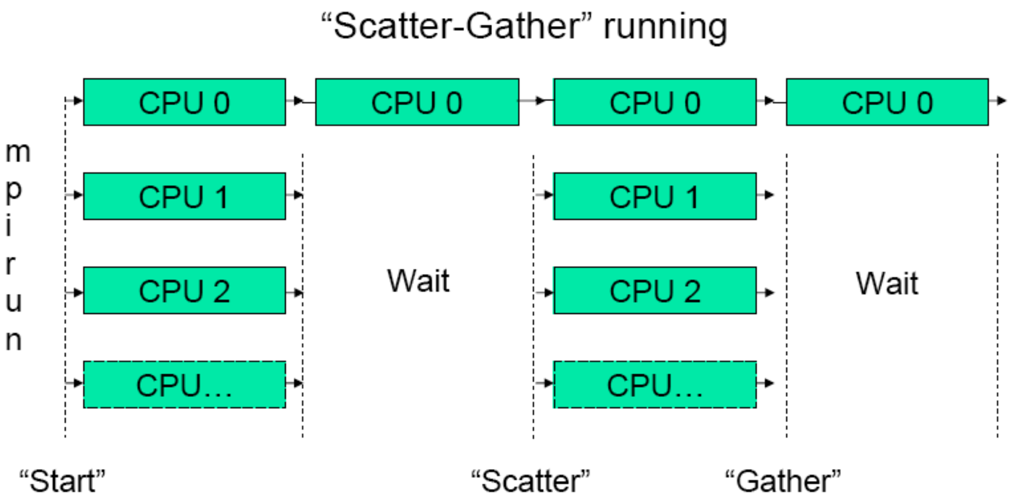


Talk by D.Lawrence/JLAB on Monday

Examples of parallelism: RooFit (1)

- **Example of Data Analysis (Fitting) in BaBar**
  - Uses MPI to run scatter/gather
    - Based on the Negative-Log Likelihood function which requires the calculation of separate values for each free parameter in each minimization step

$$NLL = \ln\left(\sum_{j=1}^{s} n_j\right) - \sum_{i=1}^{N}\left(\ln\sum_{j=1}^{s} n_j \mathcal{P}_j^i\right)$$



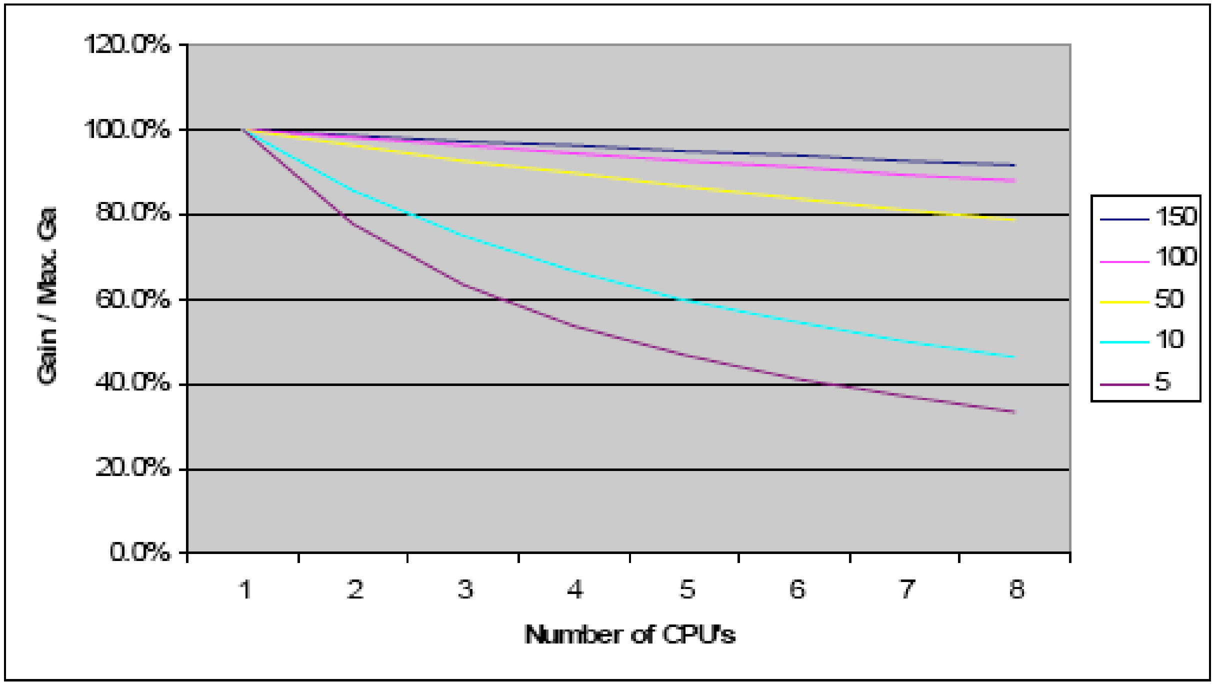
Talk by A.Lazzaro on Wednesday

RooFit (2)

- **It works well in case of large number of parameters**

Gain ~ NCPU*(NPAR + 2) / (NPAR + 2*NCPU)

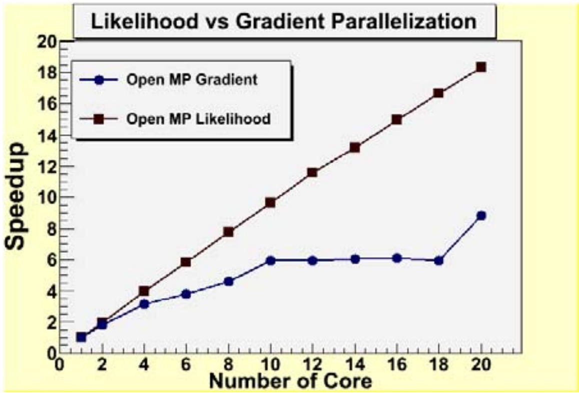
Max. Gain = NCPU

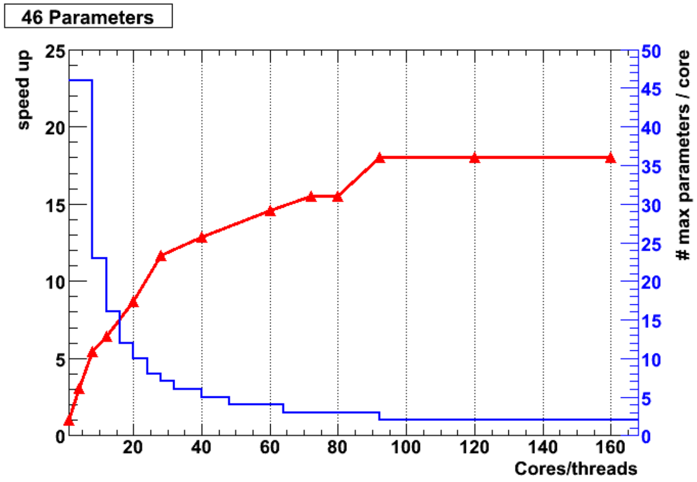


Example: ROOT minimization and fitting

- **Minuit parallelization is independent of user code**
- **Log-likelihood parallelization (splitting the sum) is more efficient**
  - more demanding on thread safety of provided code
- **Example: unbinned fit with 20 parameters**

$$\ln L = \sum_i \ln f(x_i, \theta)$$



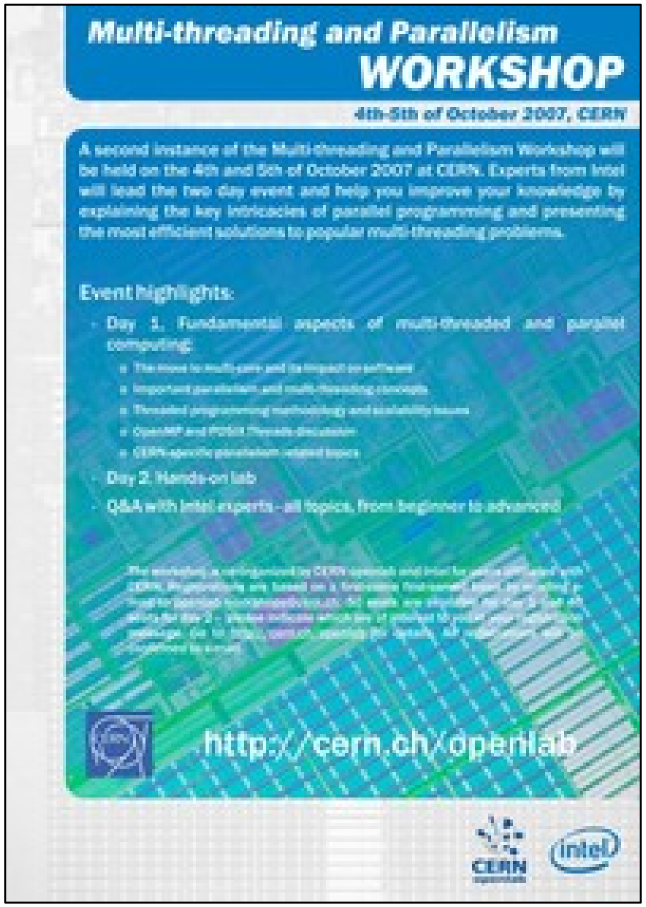


complex BaBar fitting provided by A. Lazzaro and parallelized using MPI

- **Can have combination on both**
  - parallelization via multi-threading in a multi-core CPU
  - multiple process in a distributed computing environment

Embracing parallelism

- **CERN openlab contribution:**
  - Two workshops arranged together w/Intel every year
- **Each event:**
  - 2 half-days w/lectures, 2 half-days w/exercises
  - Multiple lecturers (Intel + CERN); 30-40 participants
  - Next workshop: 11-12 November 2008
- **Large HP/Intel Blade quad-core cluster available for exercises**
- **Licenses for the Intel Threading Tools (and other SW products) available**
  - to all CERN users



Multi-threading and Parallelism
WORKSHOP

4th-5th of October 2007, CERN

A second instance of the Multi-threading and Parallelism Workshop will be held on the 4th and 5th of October 2007 at CERN. Experts from Intel will lead the two day event and help you improve your knowledge by explaining the key intricacies of parallel programming and presenting the most efficient solutions to popular multi-threading problems.

Event highlights:

- Day 1. Fundamental aspects of multi-threaded and parallel computing:
  - The move to multi-core and its impact on software
  - Important parallelism and multi-threading concepts
  - Threaded programming methodology and scalability issues
  - OpenMP and POSIX Threads discussion
  - CERN-specific parallelism related topics
- Day 2. Hands-on lab
- Q&A with Intel experts - all topics, from beginner to advanced

[illegible]

http://cern.ch/openlab

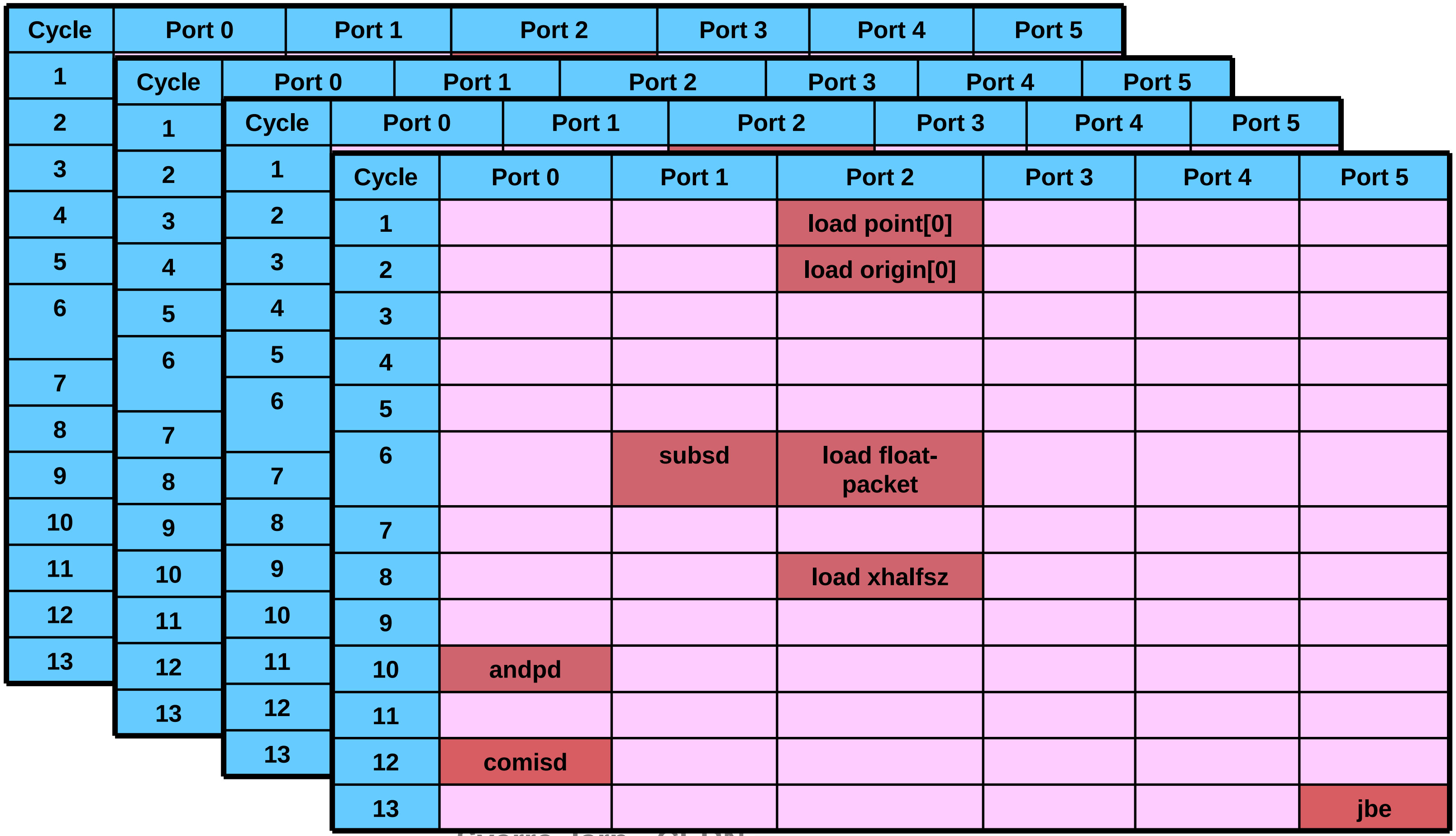
Conclusions

- **The transistors are here:**
  - Vectors, HW threads, Many-core (in 7 dimensions)
- **Fortunately, there are more and more HEP software examples of multi-/many-core scalability**
- **Nevertheless, obtaining scalability for larger and larger core counts will be difficult:**
  - Now is the time to act!
  - Identify the software model that works for you!
- **Will you be ready for 100++ cores ??**
- **We mastered uni- and multi-core. We now want to master many-core as well!**

BACKUP

We could bet on Symmetric Multithreading

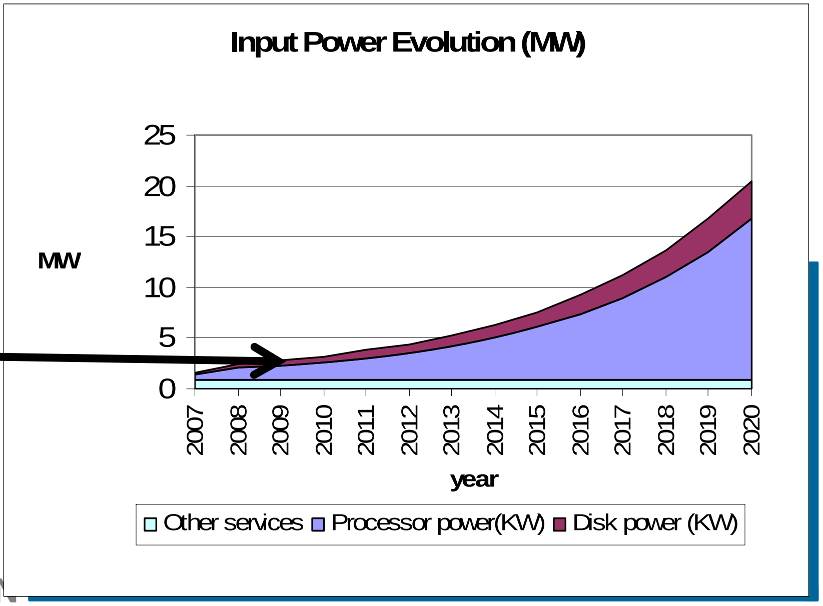
- **Because we have thin instruction streams, we could profit from SMT, provided the memory issue is under control**
  - We could easily tolerate up to 4 hardware threads!



Cycle	Port 0	Port 1	Port 2	Port 3	Port 4	Port 5
1			load point[0]			
2			load origin[0]			
3						
4						
5						
6		subsd	load float-packet			
7						
8			load xhalfsz			
9						
10	andpd					
11						
12	comisd					
13						jbe

The Power Wall

- **The CERN Computer Centre can “only” supply 2.5MW of electric power**
  - Plus 2MW to remove the corresponding heat!
- **Spread over a complex infrastructure:**
  - CPU servers; Disk servers
  - Tape servers + robotic equipment
  - Database servers
  - Other infrastructure servers
  - Network switches and routers
- **This limit will be reached in 2009!**
  - Even after a 15% increase


Input Power Evolution (MW)
MW
year
□ Other services
■ Processor power(KW)
■ Disk power (KW)