

# PH/SFT Many-Core Workshop

**(Some of) the Issues facing CERN and HEP  
in the many-core computing era**

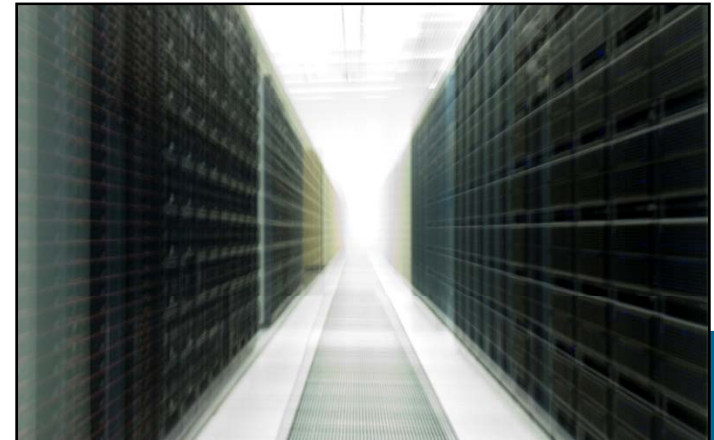
**“Capacity computing in seven dimensions”**



Sverre Jarpe

CERN openlab

CERN – 14 April 2008



# Contents

- **The good news**
  - CERN's computing infra-structure for LHC is ready
  - New silicon processes are on their way
- **The lurking issues**
  - and, there are several!
- **Possible remedies**
  - But are they painless?
- **Conclusions**

# Introduction

# The Computer Centre and the Grid

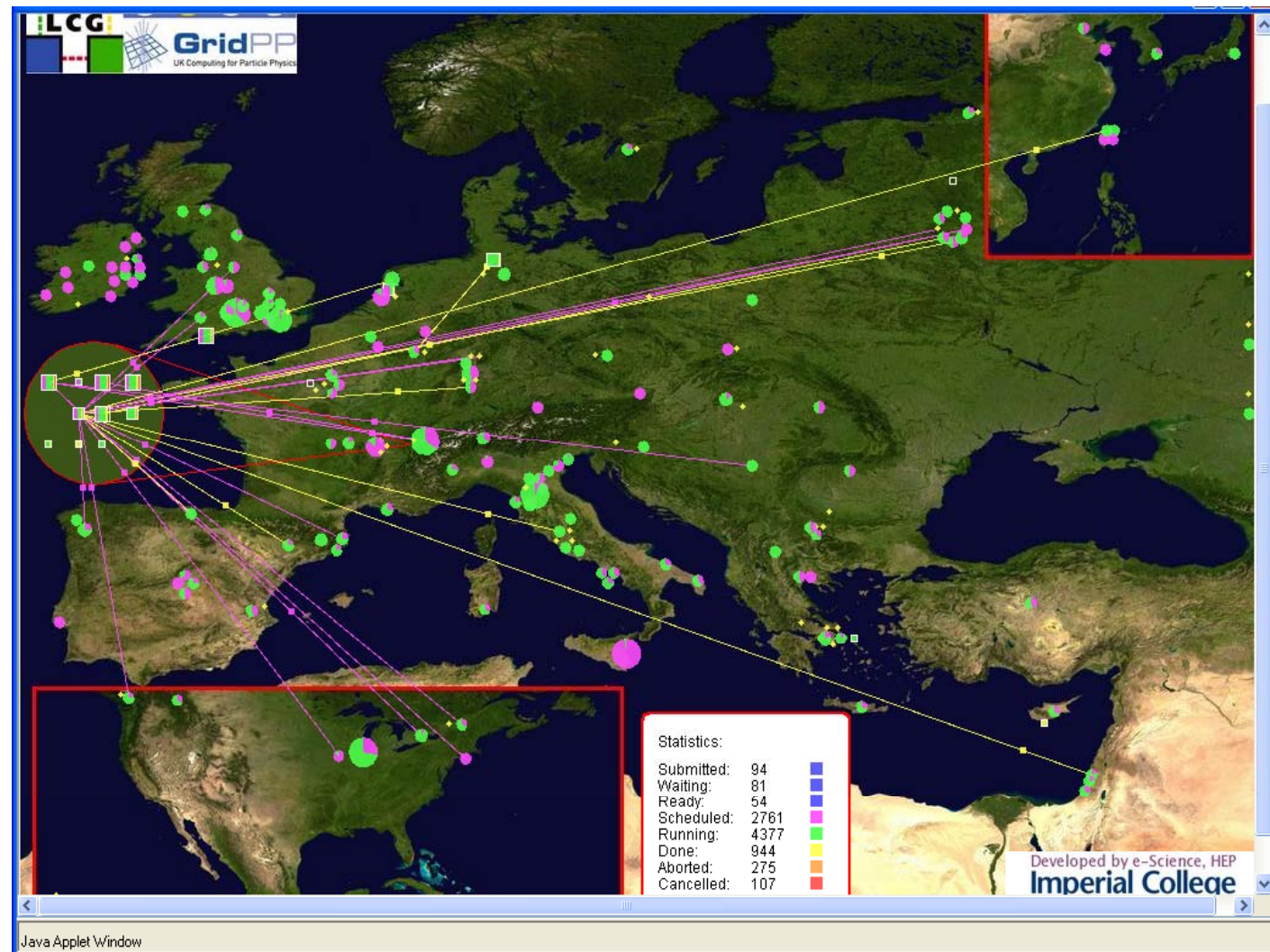
- In the Computing Centre, we are ready!



# LHC Computing Grid

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

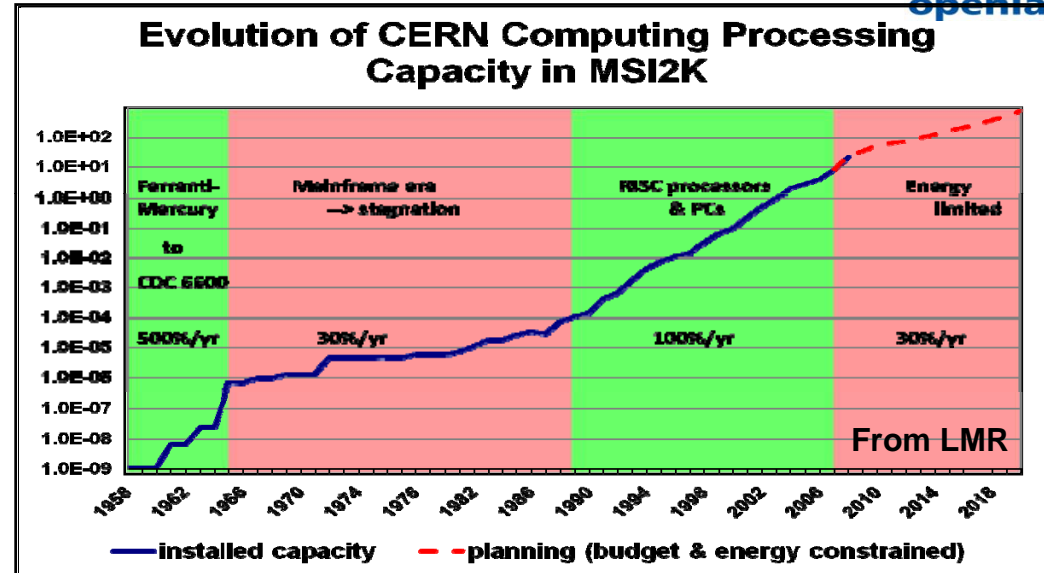
- Over 200 sites in 40 countries
- Tens of thousands of Linux servers
- Tens of petabytes of storage



# The issues

# Evolution of CERN's computing capacity

- During LEP era (1989 – 2000):
  - Doubling of compute power every year
  - Initiated with the move from mainframes to RISC systems
  
- At the CHEP-95 conference in Rio:
  - We made the first recommendation to move to PCs



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)

Rafael Yaari  
Weizmann Institute, Israel  
(R.Yaari@weizmann.ac.il)

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

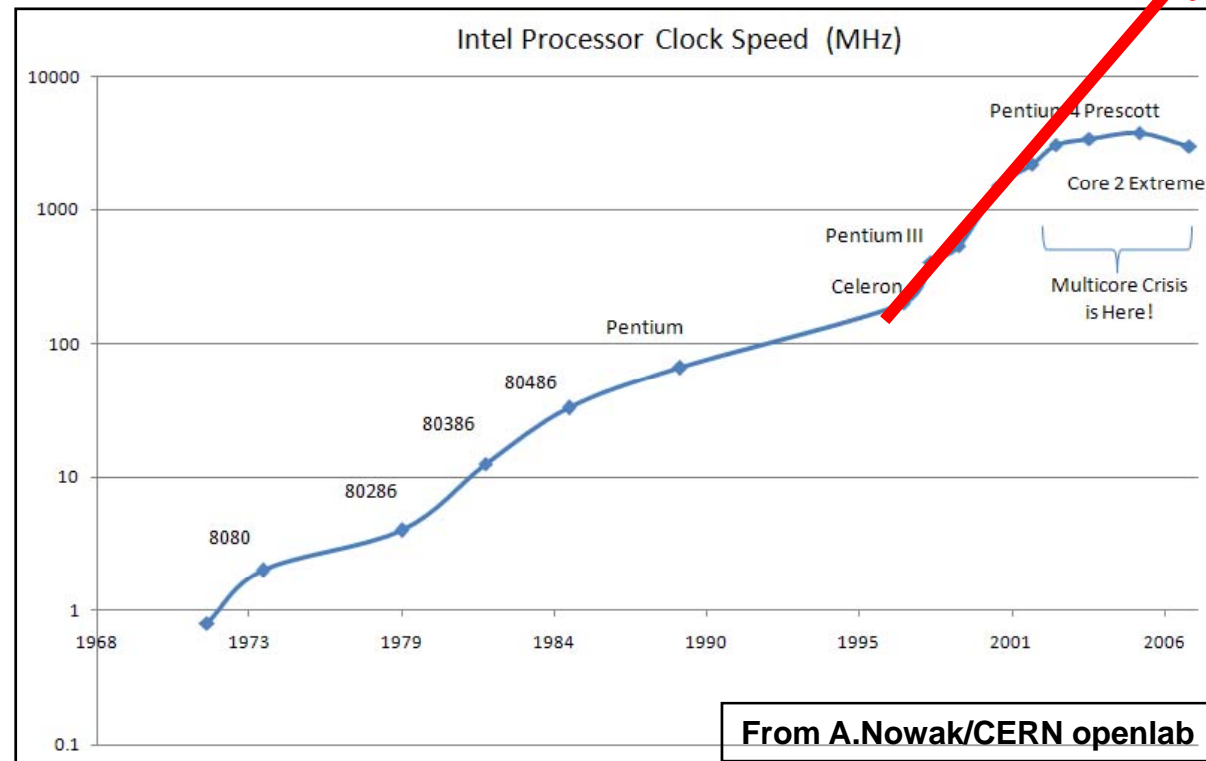
# 1) Frequency scaling is over!

- **The 7 “fat” years of frequency scaling in HEP**

- Pentium Pro (1996): 150 MHz
- Pentium 4 (2003): 3.8 GHz (~25x)

- **Since then**

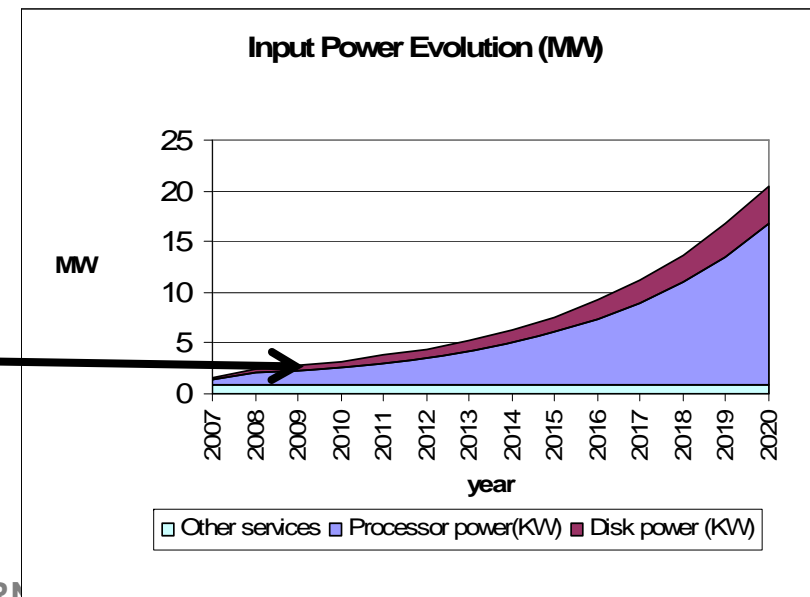
- Core 2 systems:
  - ~3 GHz
  - Quad-core





## 2) 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
    - AFS, LSF, Windows, Build, etc.
  - Network switches and routers

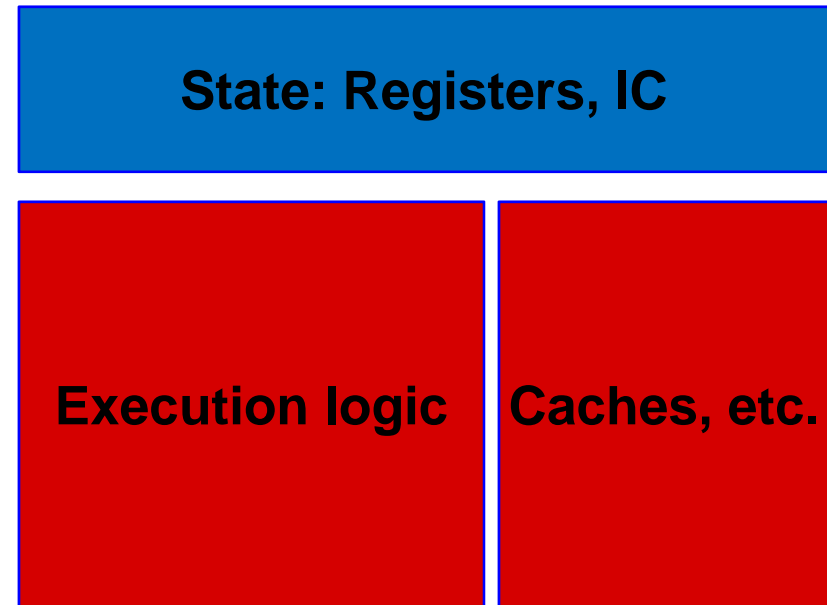


- **This limit will be reached in 2009!**

# Definition of a hardware core/thread

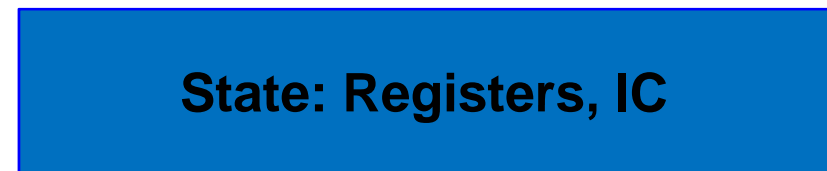
## ■ Core

- 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**



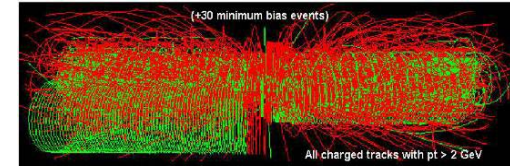
**The sharing of the execution logic can be coarse-grained or fine-grained.**

# The move to many-core systems

- **Examples of process slots: Sockets \* Cores \* HW-Threads**
  - **Conservative:**
    - Dual-socket Intel quad-core (Harpertown):
      - $2 * 4 * 1 = 8$
    - Dual-socket Intel quad-core (Nehalem):
      - $2 * 4 * 2 = 16$
  - **Aggressive:**
    - Dual-socket Sun Niagara (T2) processors w/8 cores and 8 threads
      - $2 * 8 * 8 = 128$
    - Quad-socket Intel Nehalem “octocore” with dual threading
      - $4 * 8 * 2 = 64$
    - Single-socket “Larrabee”
      - $1 * 24 * 4 = 96 (?)$
- **In the near future: Hundreds of process slots!**

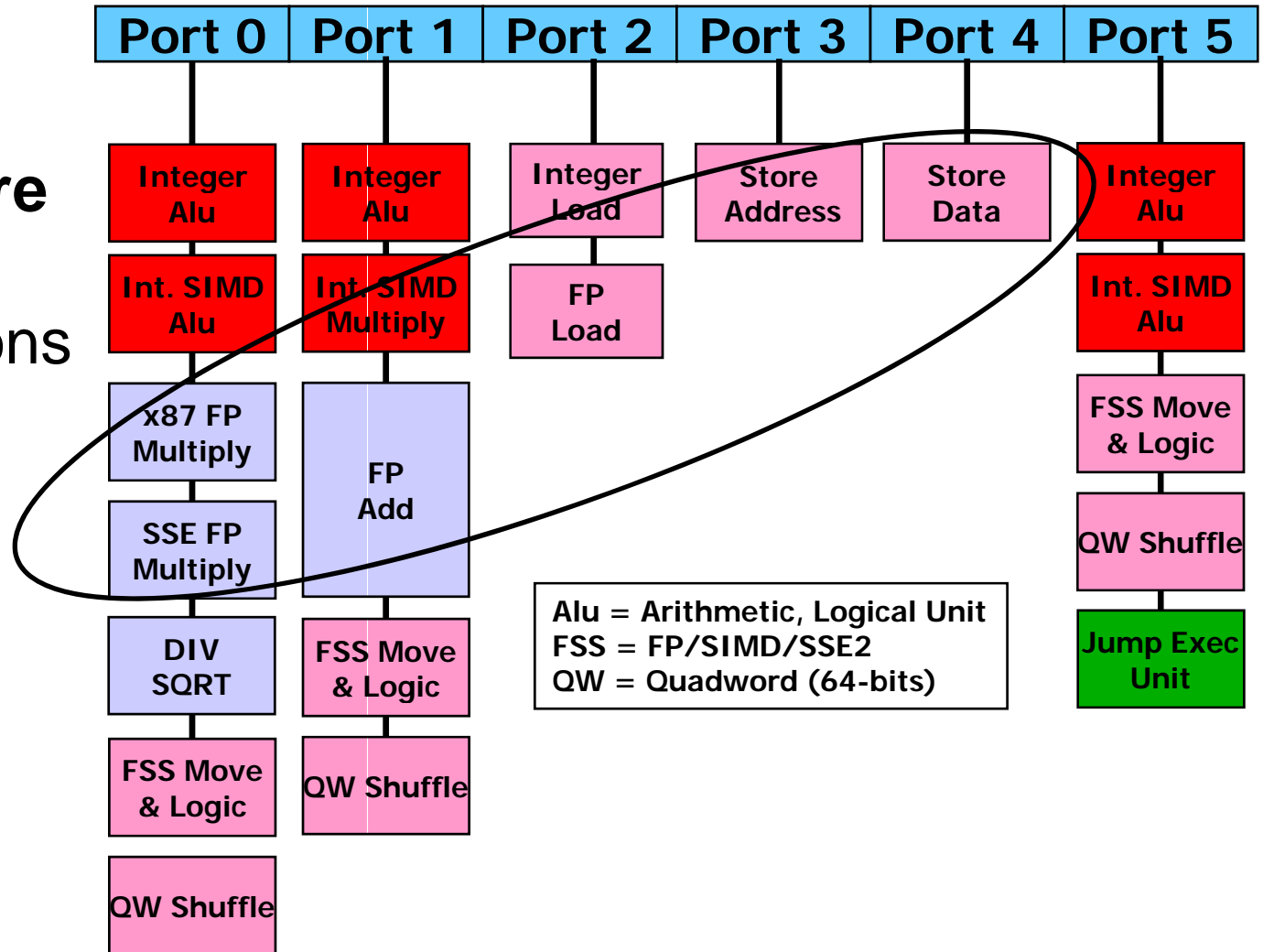
## 3) Our programming paradigm

- **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!



# Core 2 execution ports

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



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

# 4) 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 →

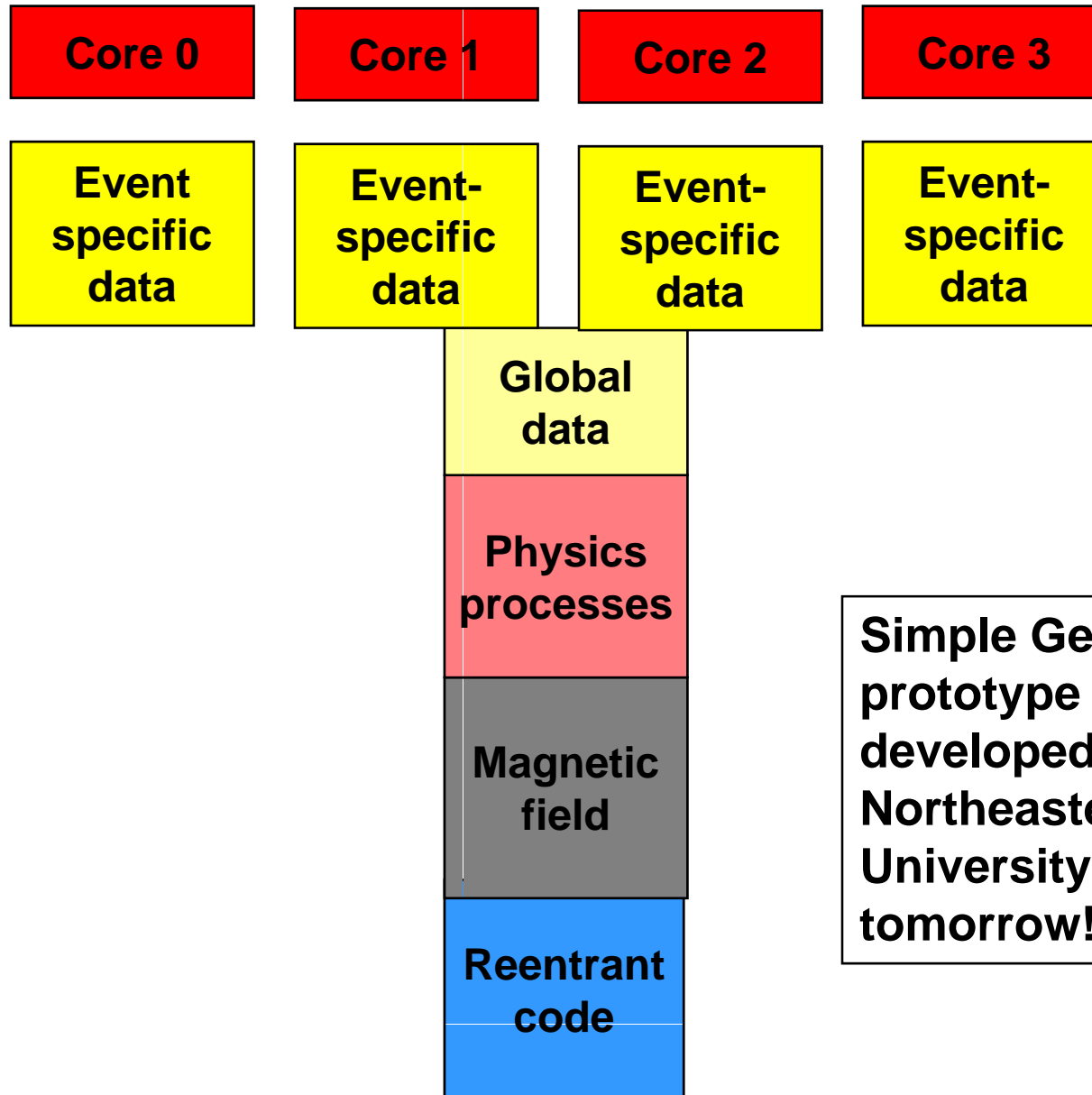
**NB: Out-of-order scheduling not taken into account.**

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

# Possible remedies

# 1) More efficient memory footprint

- As follows:



Simple Geant4  
prototype  
developed at  
Northeastern  
University – More  
tomorrow!



## 2) Embrace parallelism

- **Our contribution:**
- **Two openlab workshops arranged together w/Intel in 2007**
- **Each event:**
  - 1 day lectures, 1 day exercises
  - Multiple lecturers (Intel + CERN), 45 participants, 20 people oversubscribed
  - Survey: 100% said expectations met
  - Next workshop: 29/30 May 2008
- **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 hardware multi-processor and multiprocessor software
- Improved parallelism and multi-threading concepts
- Threaded programming techniques and security issues
- OpenMP and Intel Threading Building Blocks
- CERN-specific performance-related issues

**Day 2: Hands-on labs**

Q&A with Intel experts – all topics, from beginner to advanced

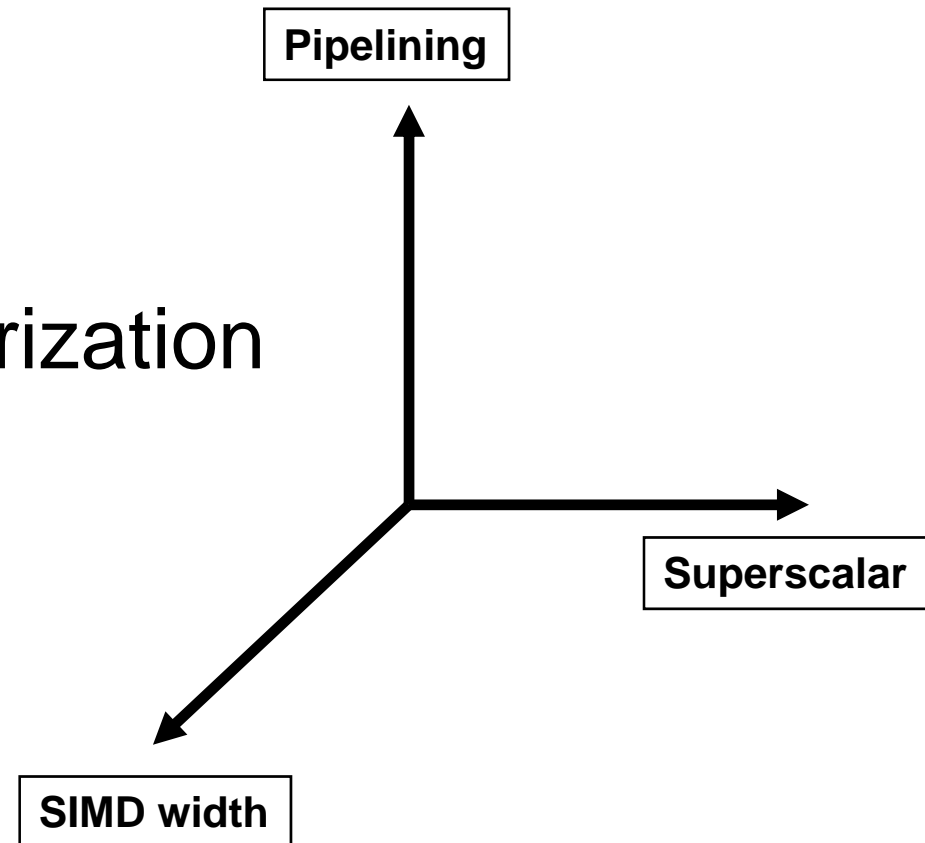
<http://cern.ch/openlab>

CERN openlab Intel



# Part 1: Opportunities for scaling performance inside a core

- **First three dimensions:**
- **Data parallelism via**
  - Loop/straight-line vectorization
  - Superscalar: Fill the ports
  - Pipelined: Fill the stages
  - SIMD: Fill the register width



SIMD = Single Instruction Multiple Data  
SSE = Streaming SIMD Extensions

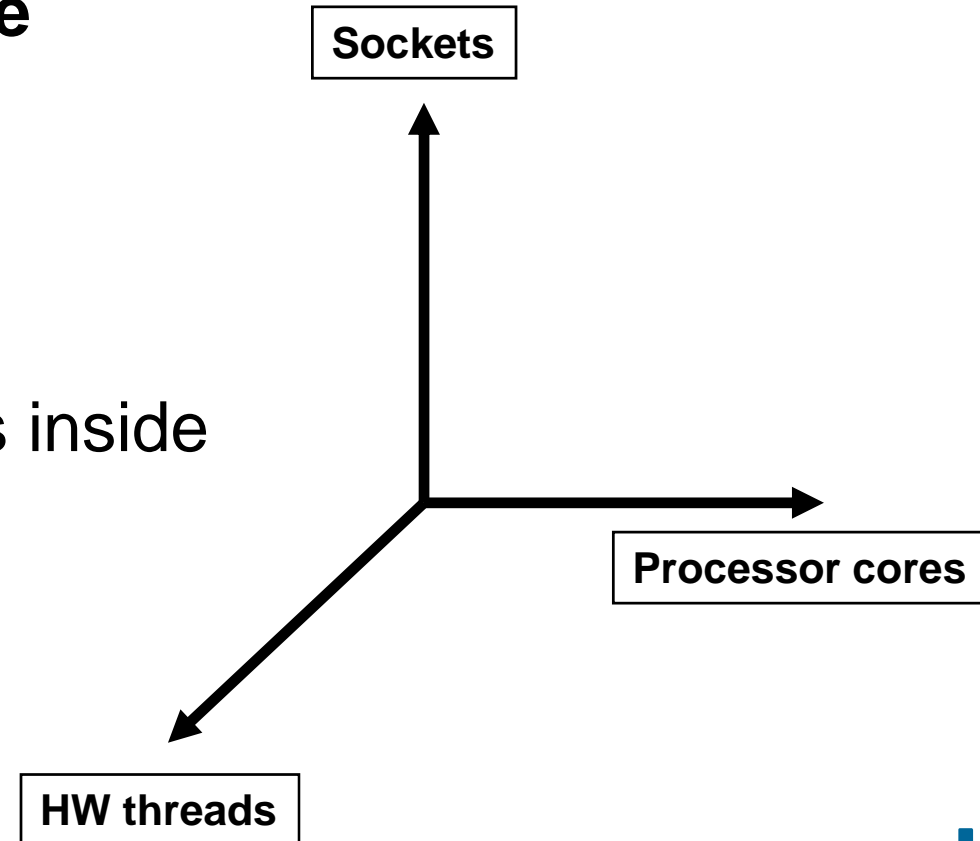
# 3) Bet on Symmetric Multithreading

- **Provided the memory issue is solved**
  - We could easily tolerate 4x SMT !

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

# Part 2: Parallel execution across multithreaded cores

- **After having tried to maximize capacity within a core**
  - First three dimensions
- **We move to the next level:**
  - Three additional dimensions inside a node:
    - HW threads
    - Processor cores
    - Sockets
- **Ideal for thread-level parallelism**



**Seventh dimension represented by multiple nodes.**

# Rethink concurrency in HEP

- **We are “blessed” with lots of it:**
  - Events
  - Particles, tracks and vertices
  - Physics processes
  - I/O streams (Trees, branches)
  - Buffer manipulations (also compaction, etc.)
  - Fitting variables
  - Partial sums, partial histograms
  - (Your favorite comes here)

# 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 (in preparation)
  - etc.

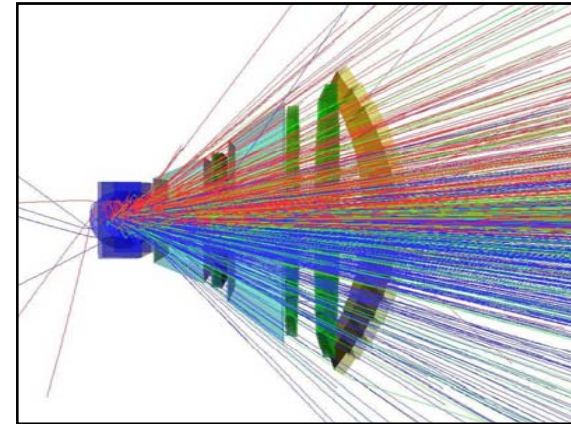
**Complementary tools available at CERN:**  
**Intel Thread Checker, Thread Profiler**  
**Linux perfmon2 (Stéphane Eranian)**

# Examples of parallelism:

## CBM track fitting

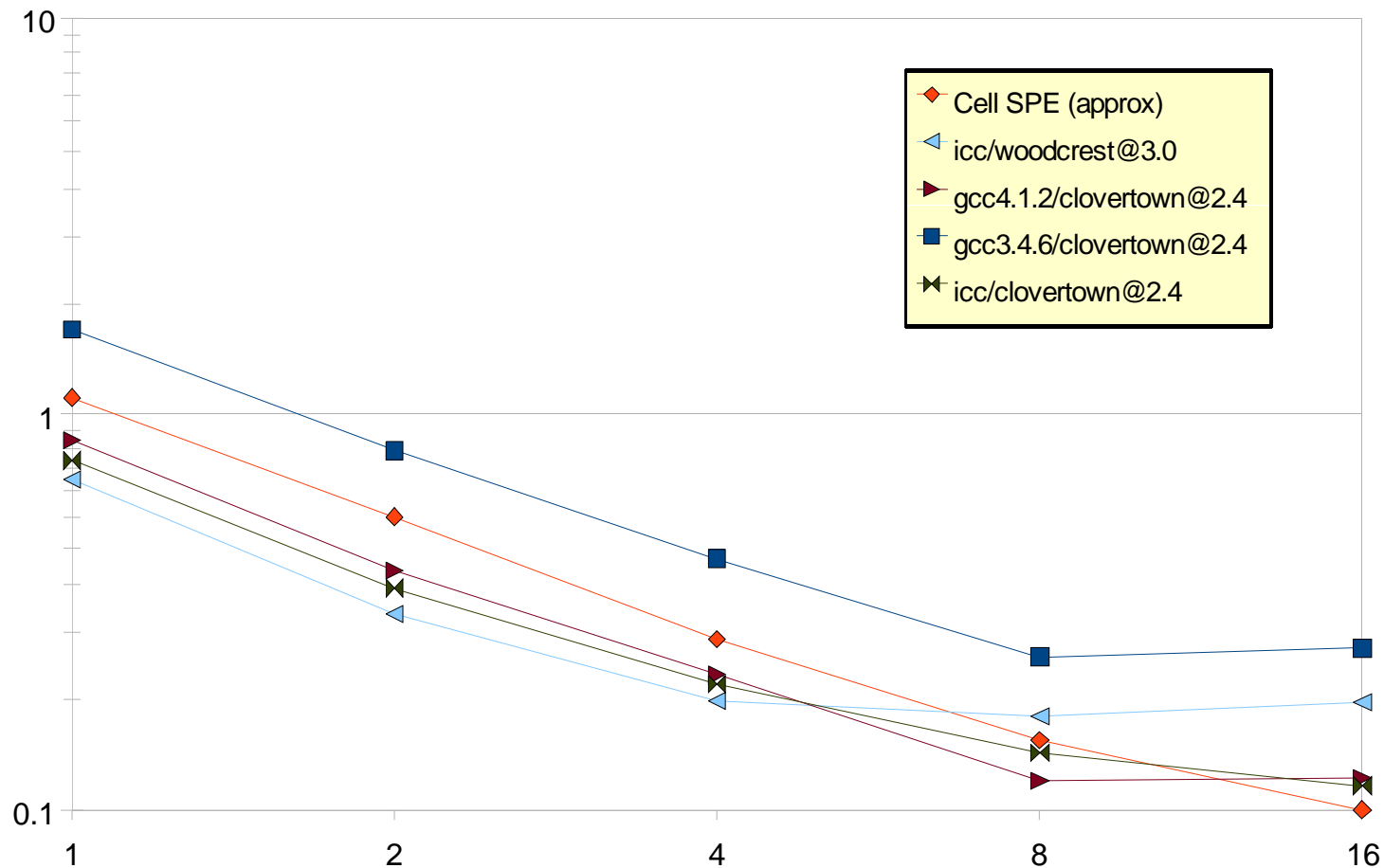
I.Kisel/GSI: “Fast SIMDized Kalman filter based track fit”  
<http://www-linux.gsi.de/~ikisel/reco/CBM/DOC-2007-Mar-127-1.pdf>

- **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 ( $\mu\text{s}$ ) as a function of the core count:





# Examples of parallelism:

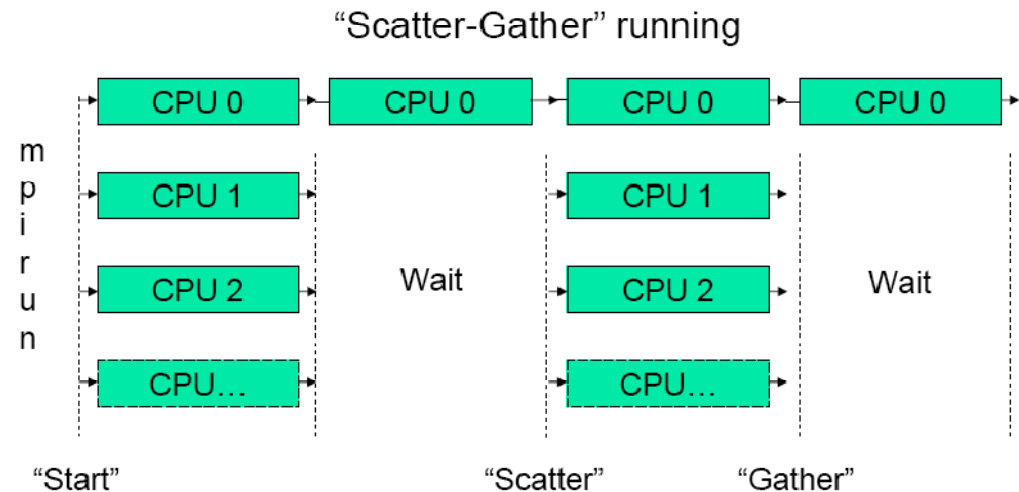
## RooFit (1)

- **Example of Data Analysis (Fitting) in BaBar (SLAC)**

- 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

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



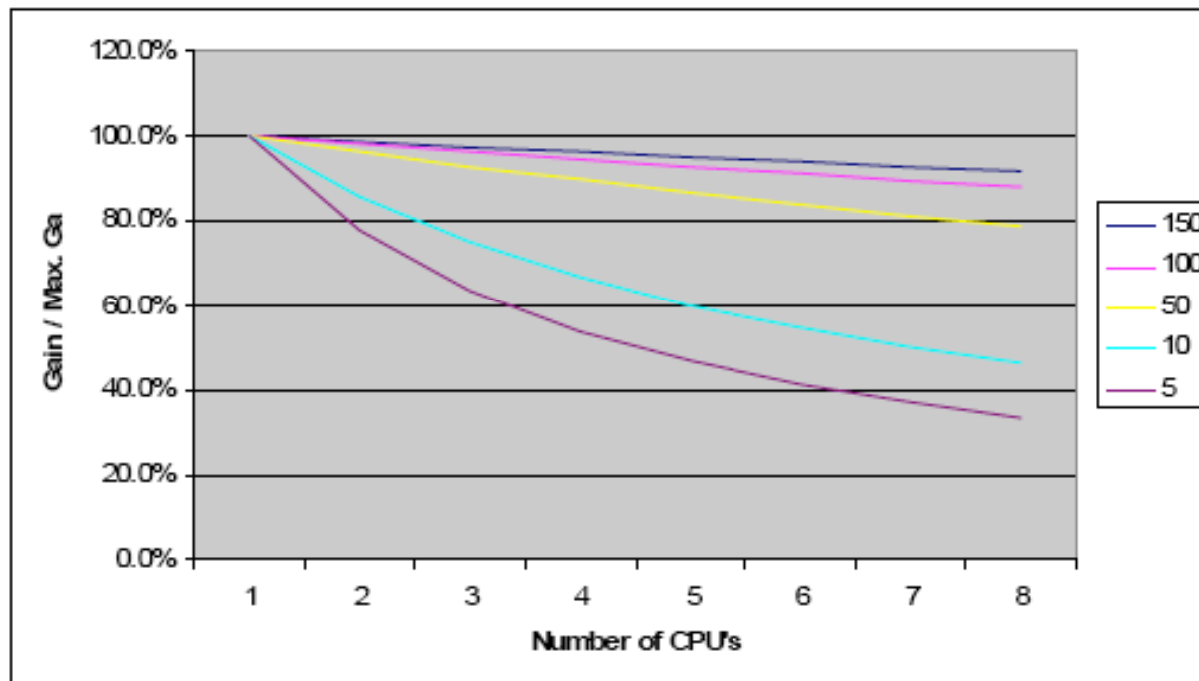
From B.Meadows's talk at RooFit Mini Workshop @ SLAC (December 2007):  
[http://www.slac.stanford.edu/BFROOT/www/doc/Workshops/2007/BaBar\\_RooFit/Agenda.html](http://www.slac.stanford.edu/BFROOT/www/doc/Workshops/2007/BaBar_RooFit/Agenda.html)

# RooFit (2)

- It works well in case of large number of parameters

Gain  $\sim \text{NCPU} * (\text{NPAR} + 2) / (\text{NPAR} + 2 * \text{NCPU})$

Max. Gain = NCPU



# Programming strategies/priorities

- **As I see them:**
  - Get memory usage (per process) under control
    - To allow higher multiprogramming level per server
  - Draw maximum benefit from hardware threading
  - Introduce coarse-grained software multithreading
    - To allow further scaling with large core counts
  - Revisit data parallel constructs at the very base
    - Gain performance inside each core
  - Use appropriate tools (perfmon2/Thread Profiler, etc.)
    - To monitor detailed program behaviour

# Conclusions

**In spite of the fact that we have thousands of servers locally (and tens of thousands in the Grid), we are confronted by several computing issues!**

- **Some solutions are easier than others:**
  - Switch on SMT in the BIOS
  - Reduce memory foot-print
  - Gradually introduce parallelism (across threads/cores/sockets)
  - Build an additional (5 MW?) computer centre
  - Revisit vectors (data parallelism)
- **But, above all, maintain software scalability and portability**
- **In all cases, we must master the 7 hardware dimensions!**

# Q & A