How to Exploit MultiCore

Kickoff workshop April ‘08

High Performance Computing for High Energy Physics

WP8: The Project

- 4 years
- Today 2x0.5 FTE + visitors
  - Possible evolution to 2 FTE including contribution from EU
- Hardware resources to perform tests
  - in a controlled environment
  - using leading edge HW, OS, OS-tuning and tools not available on lxplus
- Collaboration
  - LHC Experiments
  - OpenLab
  - Other HPC projects in HEP (and elsewhere?)

The Challenge

Exploit modern computing architecture for HPC

- Inside a core
  - Superscalar : Fill the ports (maximize instruction per cycle)
  - Pipelined : Fill the stages (avoid stalls)
  - SIMD : Fill the register width (exploit SSE)
- Inside a Box
  - HW threads: Fill up a core (share core & caches)
  - Processor cores: Fill up a processor (share of low level resources)
  - Sockets: Fill up a box (share high level resources)
- LAN & WAN
  - Optimize scheduling and resource sharing on the Grid

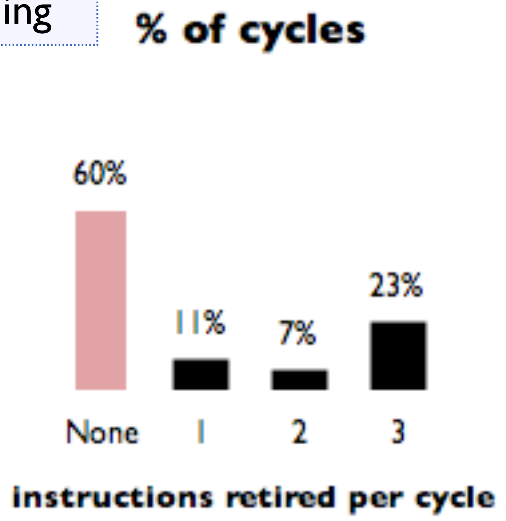
- HEP has been traditionally good (only) in the latter

CMS System-level performance analysis

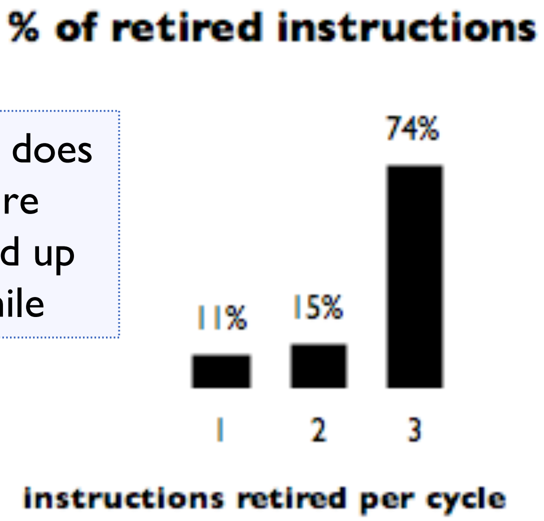
- This is a view complementary to *valgrind* that helps us:
  - Design algorithms and data structures suited well to systems we use
  - Purchase systems best suited to our applications
  - Based on hardware-level performance monitoring, plus perfctr or oprofile
- We have done a preliminary study of what actually happens at the system hardware level when the operating system says CPU is 100% busy
  - The answer turns out to be "*Nothing,* most of the time"!
- On next slide, results on 2 x dual core AMD Opteron 270, "lots of" memory
  - Analysis of **CMSSW 1.3.0 (early 2007)** reconstruction of 100 B_JETS events.
  - The CPU doesn't have enough to do: it is swamped by memory accesses.
  - Also reveals a new issue: code is too large and code locality is poor. Nearly a third of the application memory footprint turned out to be for code (~150 MB).
  - Bizarrely enough, despite requiring ~500 MB memory, the application does not seem to access very much of it: the L2 cache seems more than sufficient. This is confirmed in a number of different ways.

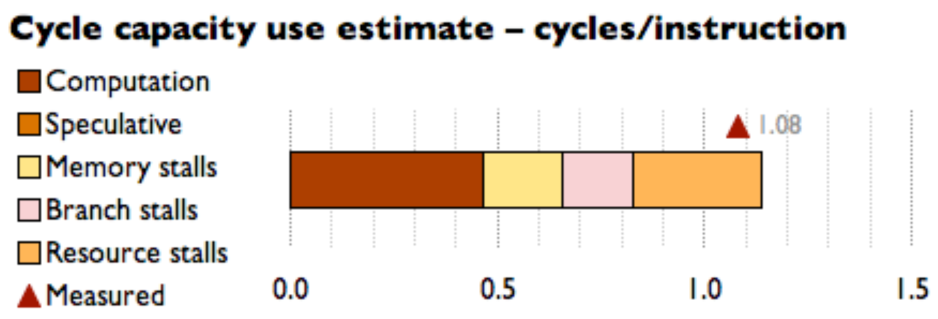
by Lassi Tuura

1. Most of the time, CPU does nothing

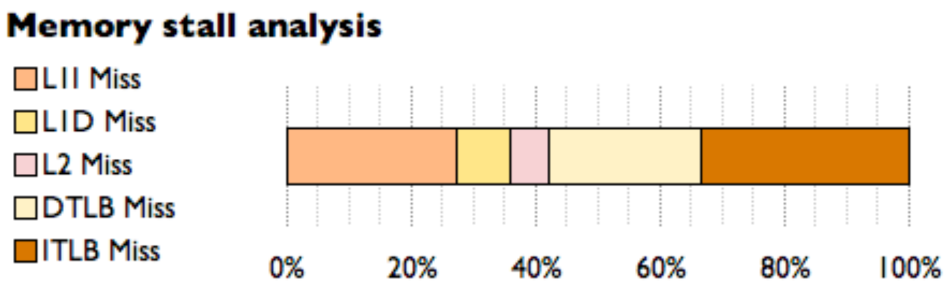


2. But when it does work, the entire pipeline is filled up for a short while

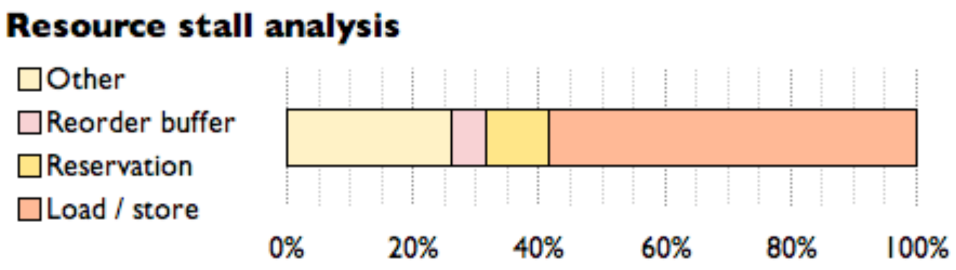




3. Lots of unpleasant stalls… Breakdown follows.



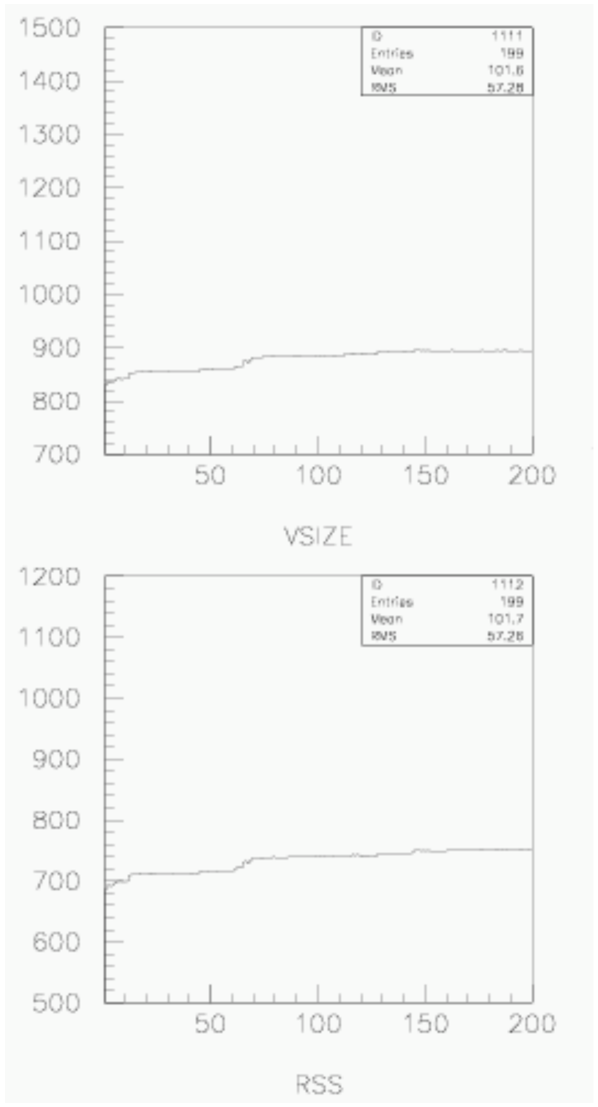
4. Memory stalls 60% for code, 60% for page table



5. CPU is mainly clogged up with memory accesses

CMS Reconstruction Memory Footprint

- Reconstruction modules ~400 MB
  - » 200 ttbar events in 17X
  - » Only few modules use more than 2MB/event
- About ~400 MB in condition payloads
  - » pixel payload in 2.0.0 down to 67MB from 132MB
  - » CSC payload down to 8MB from ~20 M
- Output module adds up to 400 MB when storing all reco and HLT products
  - » Almost 12000 branches with old HLT data model
  - » Large increase due to use of split mode in ROOT
    - Better I/O but larger memory consumption
  - » Starting with 1_8_0 non-split mode for RECO
    - Reduced footprint in production environment
  - » Split-mode only for objects in AOD tier
    - Better performance for end user



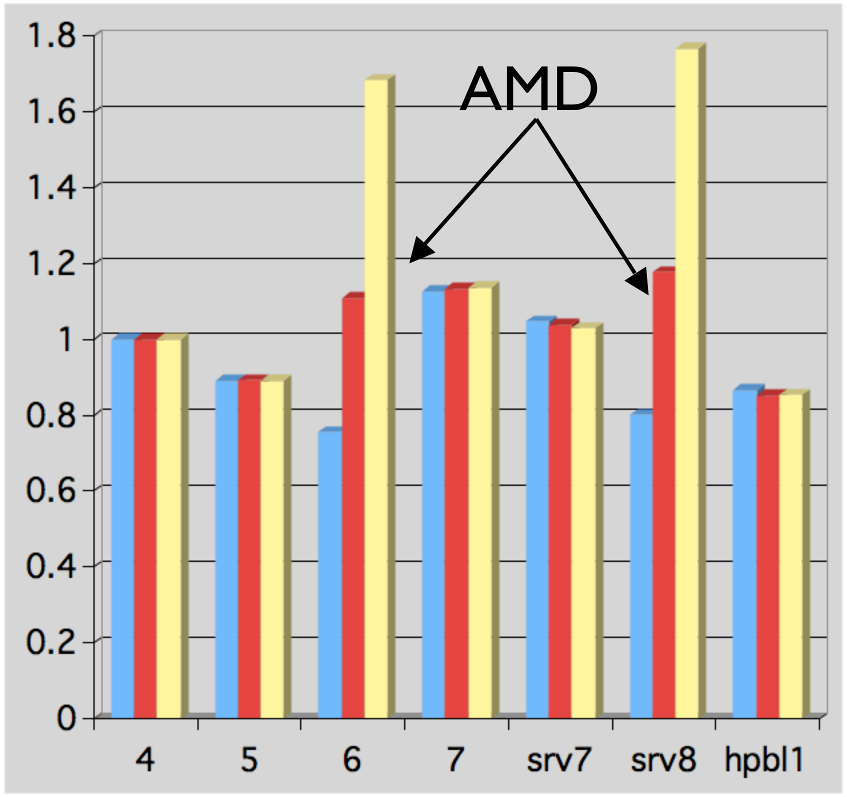
CMS Reconstruction Timing

– Intel(R) Xeon(R) CPU E5345 @ 2.33GHz

process	Init (from files!) *seconds*	Reco *seconds/Ev*	Output *seconds/Ev*
Single pi 1000 GeV	30	1.2	0.05
QCD low-pt 20-120 GeV	30	5.3	0.25
ttbar	30	10.5	0.35

Behavior on Multicore

- Running M cms reconstruction on N node (and N-M scimark2)
  - » Difference less than 2% in Event/s between M=1 and M=N (up to N=8)
- Difference in performance of the very same code on different architectures depending of the complexity of the problem (size and density of the event)



CMS Reco
Secods/Event

by Gabriele Benelli

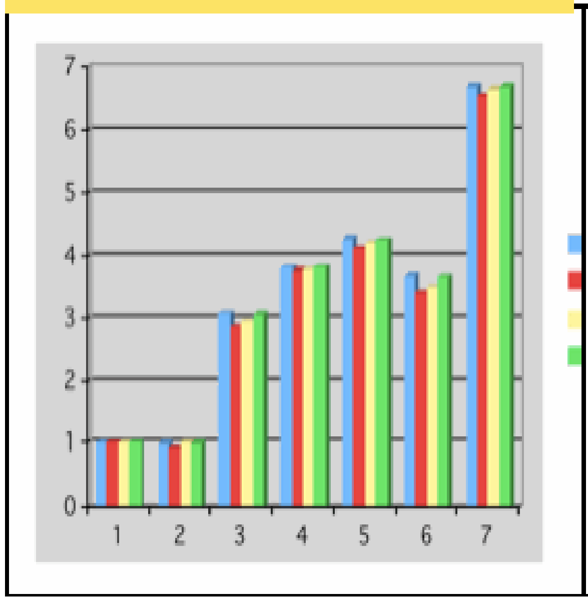
HEPIX Benchmarks

	lxbench01	lxbench02	lxbench03	lxbench04	lxbench05	lxbench06	lxbench07
Number of cores	2	2	4	4	4	4	8
Frequency (GHz)	2.8	2.8	2.2	2.66	3.0	2.6	2.33
Cache (could be L2/L3) (MB)	1	2	2	4	4	2	8
Memory (GB)	2	4	2	8	8	8	16
Processor	Nocona	Irvingdale	Opteron 275	Woodcrest	Woodcrest	Opteron 2218 Rev.F	Clovertown

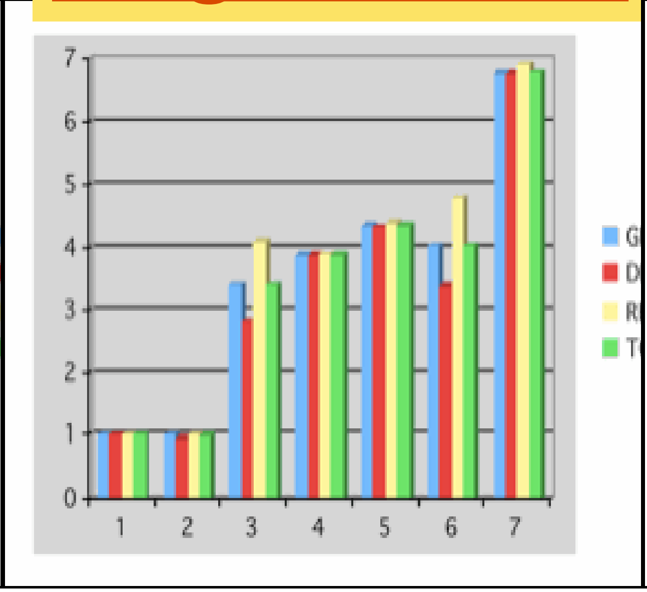
Performance normalized to lxbench01

Look at the yellow bar

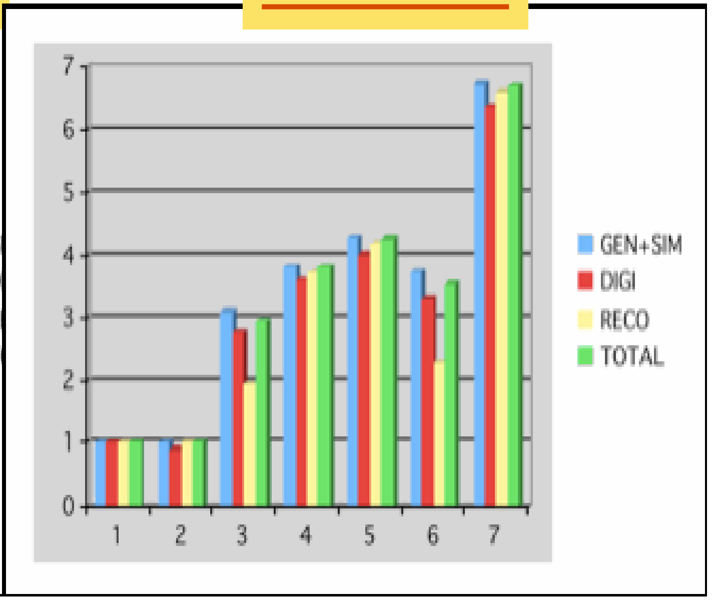
QCD_80_120



SinglePi-1000



TTbar



WP1:

- Objective
  - Investigate current and future multi-core architectures.
  - Evaluate tools to measure performance.
  - Develop a measurement and analysis methodology.
- Deliverables
  - Assessment of industry trend in multi-core architectures.
  - Recommendations on tools, metrics and methodology to assess the performance of LHC physics application software on such architectures
- Short term issues
  - Deploy a stable environment to measure LHC production code
  - Provide a set of platforms where to perform such measurements
  - Make available tools and documentation

WP2

- Objective
  - Measure and analyze performance of current LHC physics application software on multi-core architectures
  - Identify bottlenecks
  - Prototype solutions at the level of **system** and **core** libraries
- Deliverables
  - Reports on performance of current LHC physics application software
  - Recommendations on best practices to avoid bottlenecks and best exploit multi-core architectures
  - Eventual materialization in software library components to implement them
- Short term issues
  - Make sure we use an optimized OS and Compiler tuning

WP3

- Objective
  - Investigate solutions to parallelize current LHC physics software at **application framework** level
  - Identify reusable design patterns and implementation technologies to achieve parallelization
  - produce prototypes
- Deliverables
  - Recommendations on reusable design patterns and implementation technologies to use to achieve parallelization
  - Eventual materialization in software library components to implement them
- Short term issues
  - Evaluate pro&cons of various alternatives
  - Solve key showstoppers

WP4

- Objectives
  - Investigate solutions to parallelize **algorithms** used in current LHC physics application software
  - Identify reusable design patterns and implementation technologies to achieve effective **high granularity** parallelization
  - produce prototypes
- Deliverables
  - Recommendations on reusable design patterns and implementation technologies to use to achieve effective high granularity parallelization
  - Eventual materialization in software library components to implement them
- Short term issues
  - Evaluate gcc4.3 and port foundation and experiment code to it
  - Experiment with posix-thread, OMP and parallel gcclib

Summary & Conclusions

– On Wednesday!

Organizational Matters

- HyperNews, mailing-list?
- Meeting: Biweekly
  - Monday 3pm, Thursday 2pm (alternate with AF)
  - EVO, phone?
    - Do we need a physical room at cern???