Introduction to High-Performance Computing

Dr. Axel Kohlmeyer

Scientific Computing Expert

Information and Telecommunication Section
The Abdus Salam International Centre
for Theoretical Physics

http://sites.google.com/site/akohlmey/

akohlmey@ictp.it

The Abdus Salam
International Centre
for Theoretical Physics

Why use Computers in Science?

- Use complex theories without a closed solution:
  solve equations or problems that can only be solved numerically, i.e. by inserting numbers into expressions and analyzing the results
- Do “impossible” experiments:
  study (virtual) experiments, where the boundary conditions are inaccessible or not controllable
- Benchmark correctness of models and theories:
  the better a model/theory reproduces known experimental results, the better its predictions

What is High-Performance Computing (HPC)?

- Definition depends on individual person
  > HPC is when I care how fast I get an answer
- Thus HPC can happen on:
  - A workstation, desktop, laptop, smartphone!
  - A supercomputer
  - A Linux/MacOS/Windows/... cluster
  - A grid or a cloud
  - Cyberinfrastructure = any combination of the above
- HPC also means High-Productivity Computing

Parallel Workstation

- Most computers today are parallel workstations => multi-core processors
- Running Linux OS (or MacOS X) allows programming like traditional Unix workstation
- All processors have access to all memory
  - Uniform memory access (UMA): 1 memory pool for all, same speed for all
  - Non-uniform memory access (NUMA): multiple pools, speed depends on “distance”

An HPC Cluster is...

- A cluster <u>needs</u>:
  - Several computers, <u>nodes</u>, often in special cases for easy mounting in a rack
  - One or more networks (<u>interconnects</u>) to hook the nodes together
  - Software that allows the nodes to communicate with each other (e.g. <u>MPI</u>)
  - Software that reserves resources to individual users
- A cluster <u>is</u>: all of those components <u>working together</u> to form one big computer

What is Grid Computing?

- Loosely coupled network of compute resources
- Needs a “middleware” for transparent access to inhomogeneous resources, find matching ones
- Modeled after power grid
  => share resources not needed right now
- Run a global authentication framework
  => Globus, Unicore, Condor, Boinc
- Run an application specific client
  => SETI@home, Folding@home

What is Cloud Computing?

- Simplified: “Grid computing made easy”
- Grid: use “job description” to match calculation request to a suitable available host, use “distinguished name” to uniquely identify users, opportunistic resource management
- Cloud: provide virtual server instance on shared resource as needed with custom OS image, commercialization (cloud service providers, dedicated or spare server resources), physical location flexible, web frontend

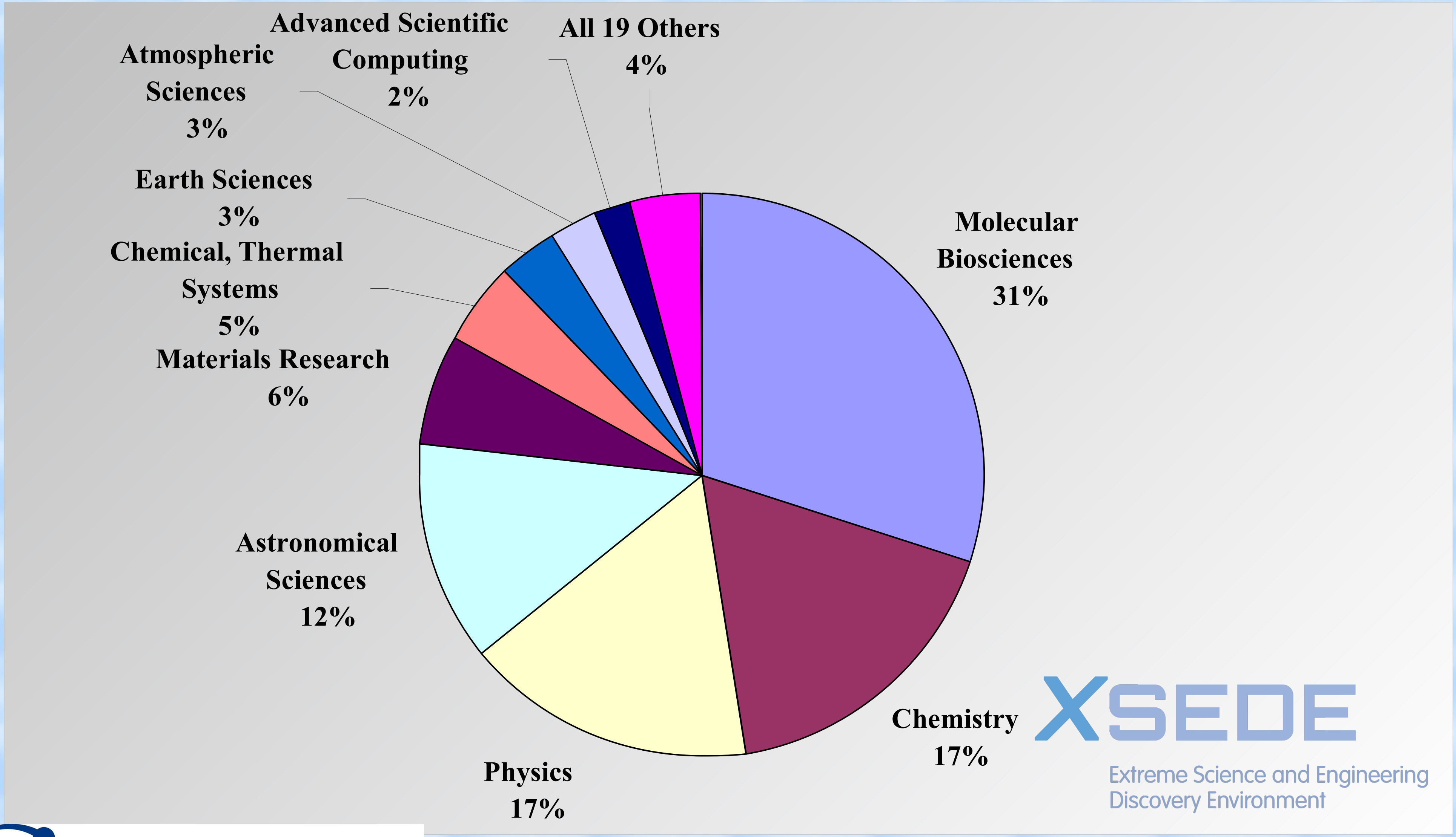
What is Supercomputing (SC)?

- The most visible manifestation of HPC
- Programs run on the fastest and largest computers in the world (=> Top500 List)
- Desktop vs. Supercomputer in 2012 (peak):
  - Desktop processor (1 core): ~10 GigaFLOP/s
  - Tesla C2050 GPU (448 cores): >500 GigaFLOP/s
  - “K” supercomputer: >10 PetaFLOP/s
- Sustained vs. peak: “K” 93%, “Jaguar” 75%, “Nebulae” 43%, “Roadrunner” 76%, BG/P, 82%

Why would HPC matter to you?

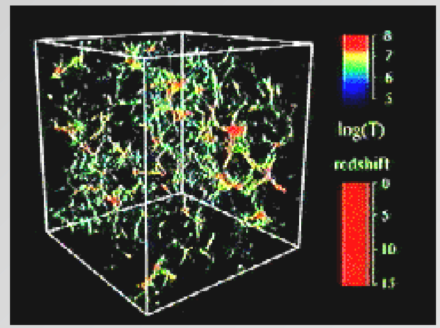
- Scientific computing is becoming more important in many research disciplines
- Problems become more complex, need teams of researchers with diverse expertise
- Scientific (HPC) application development limited often limited by lack of training
- More knowledge about HPC leads to more effective use of HPC resources and better interactions with (computational) colleagues

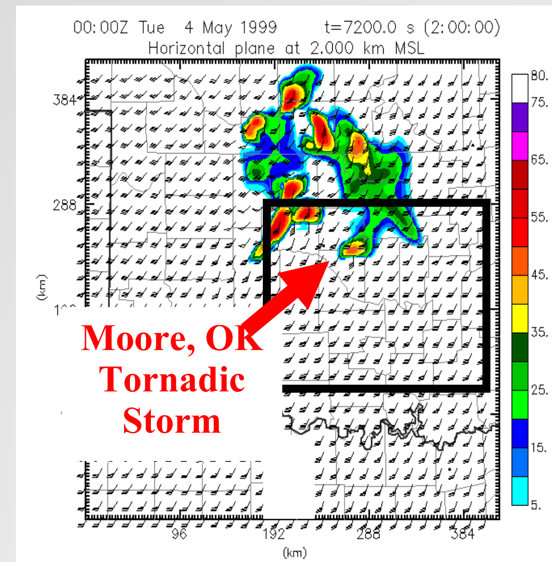
Research Disciplines in HPC



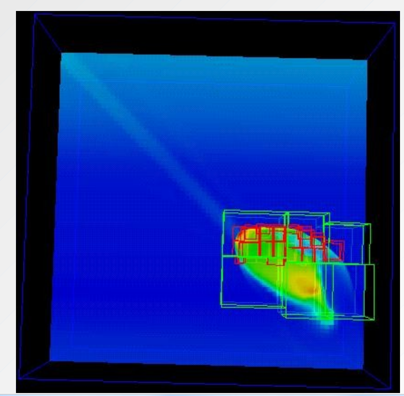
Some Examples...

- Simulation of physical phenomena:
  - Climate modeling
  - Galaxy formation



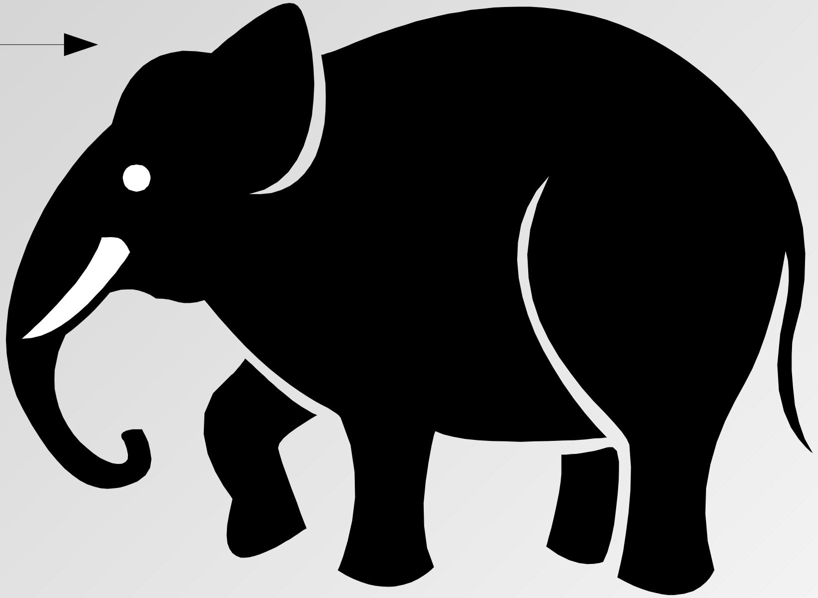


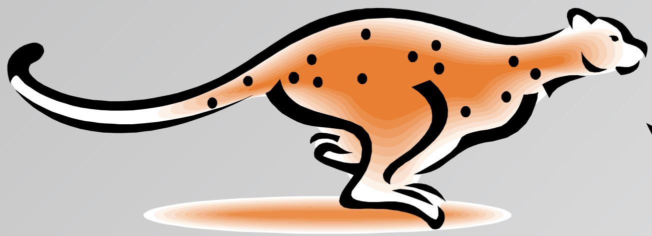
- Data mining
  - Gene sequencing
  - Detecting potential Tornados
- Visualization
  - Reducing large data sets into pictures a scientist understands



Why Would I Need HPC?

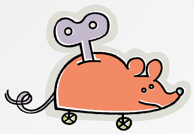
- My problem is big





- My problem is complex

- My computer is too small and too slow



- My software is not efficient and/or not parallel

HPC vs. Computer Science

- Most people in HPC are no computer scientists
- Software has to be correct first and (then) efficient; packages can be over 30 years “old”
- Technology is a mix of “high-end” & “stone age” (Extreme hardware, MPI, Fortran, C/C++)
- So what skills do I need to for HPC:
  - Common sense, cross-discipline perspective
  - Good understanding of calculus and (some) physics
  - Patience and creativity, ability to deal with “jargon”

HPC is a Pragmatic Discipline

- Raw performance is not always what matters:
  how long does it take me to get an answer?
- HPC is more like a craft than a science:
  => practical experience is most important
  => leveraging existing solutions is preferred over inventing new ones requiring rewrites
  => a good solution today is worth more than a better solution tomorrow
  => a readable and maintainable solution is better than a complicated one

How to Get My Answers Faster?

- Work harder
  => get faster hardware (get more funding)
- Work smarter
  => use optimized algorithms (libraries!)
  => write faster code (adapt to match hardware)
  => trade convenience for performance
  (e.g. compiled program vs. script program)
- Delegate parts of the work
  => parallelize code, (grid/batch computing)
  => use accelerators (GPU/MIC CUDA/OpenCL)

What Determines Performance?

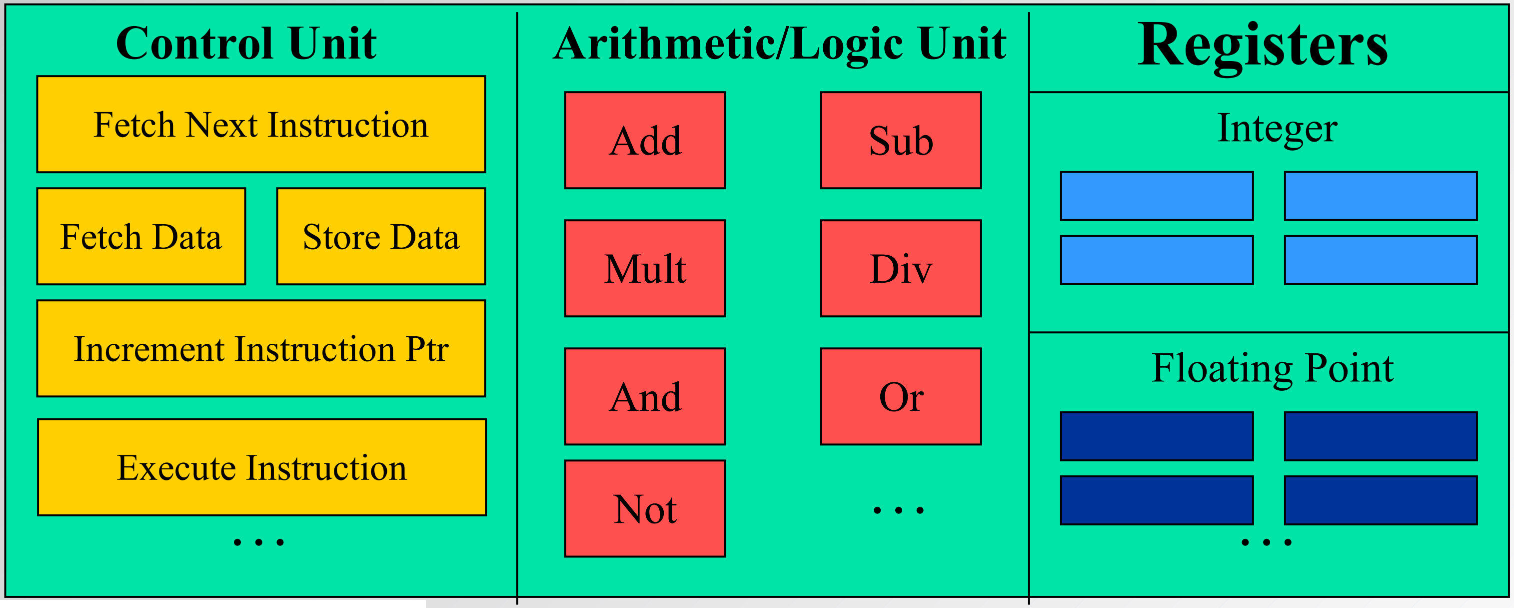
- How fast is my CPU?
- How fast can I move data around?
- How well can I split work into pieces?
  Very application specific:
  => never assume that a good solution for one problem is as good a solution for another
  => always run benchmarks to understand requirements of your applications and properties of your hardware
  => respect Amdahl's law

A Simple Calculator

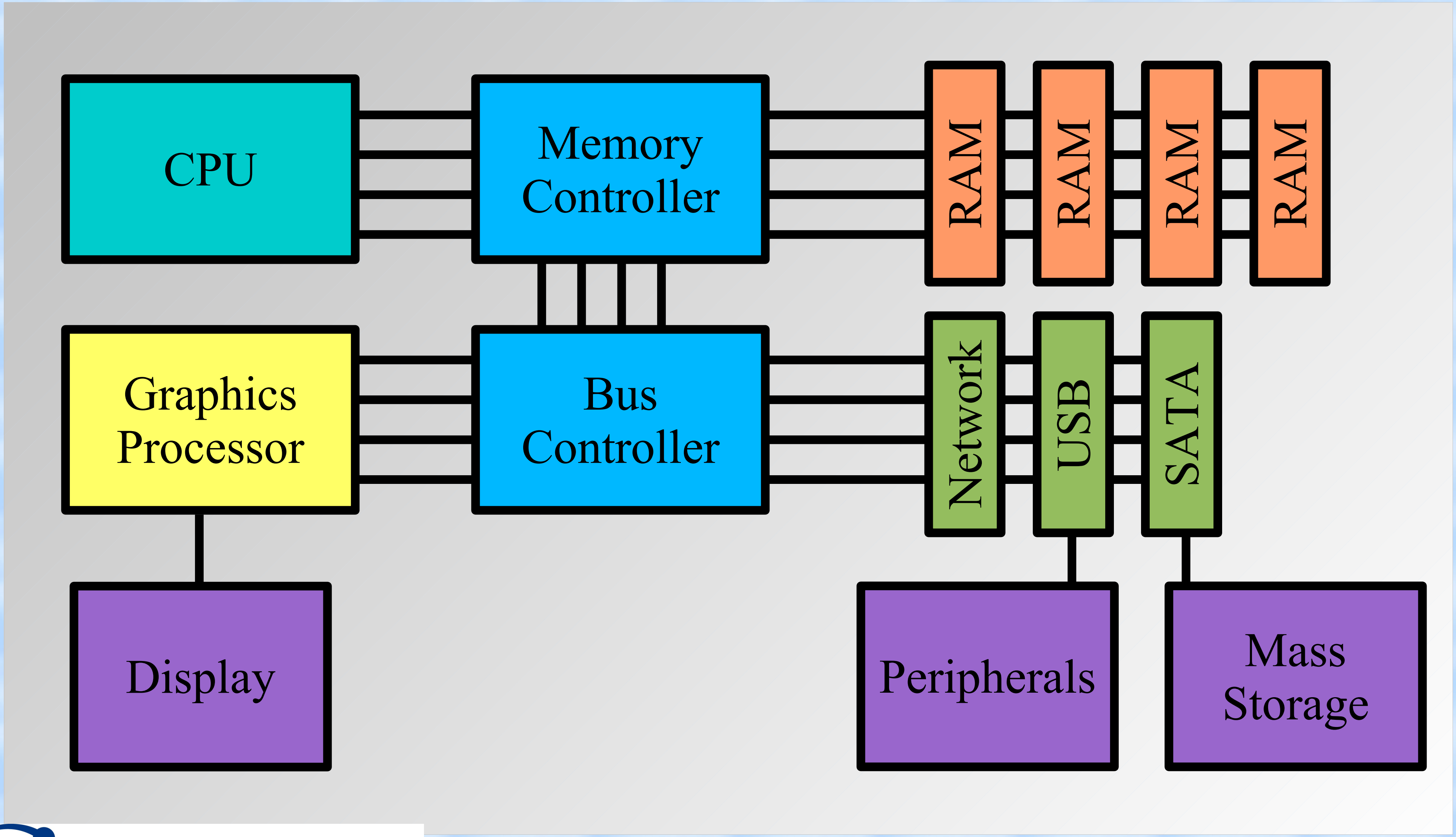
1) Enter number on keyboard => register 1
2) Turn handle forward = add backward = subtract
3) Multiply = add register 1 with shifts until register 2 is 0
4) Register 3 = result

A Simple CPU

- The basic CPU design is not much different from the mechanical calculator.
- Data still needs to be fetched into registers for the CPU to be able to operate on it.



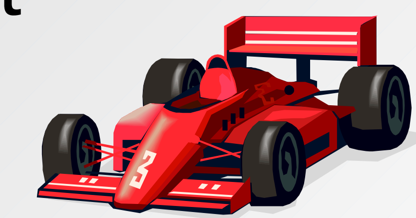
A Typical Computer



Running Faster v1: Cache Memory

- Registers are very fast, but very expensive
- Loading data from memory is slow, but RAM is cheap and there can be a lot of it

- Cache memory = small buffer of fast memory between regular memory and CPU; buffers blocks of data



- Cache can come in multiple “levels”, L#:
  L1: fastest/smallest <-> L3: slowest/largest
  can be within CPU, or external

The Abdus Salam International Centre for Theoretical Physics

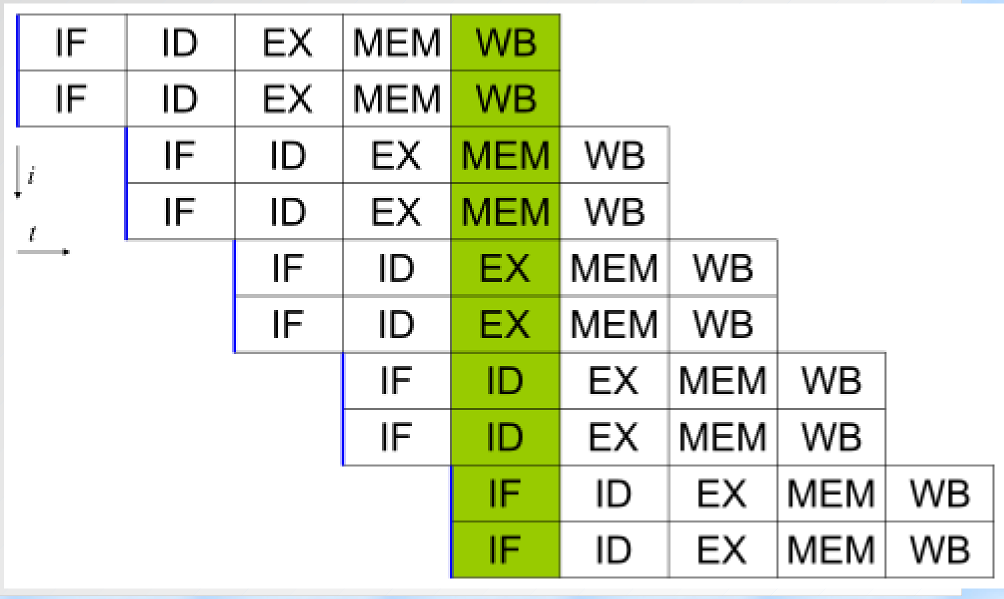
Running Faster v2: Pipelining

- Multiple steps in one CPU “operation”: fetch, decode, execute, memory, write back => multiple functional units
- Using a pipeline can improve their utilization, allows for faster clock
- Dependencies and branches can stall the pipeline
  => branch prediction
  => no “if” in inner loop

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

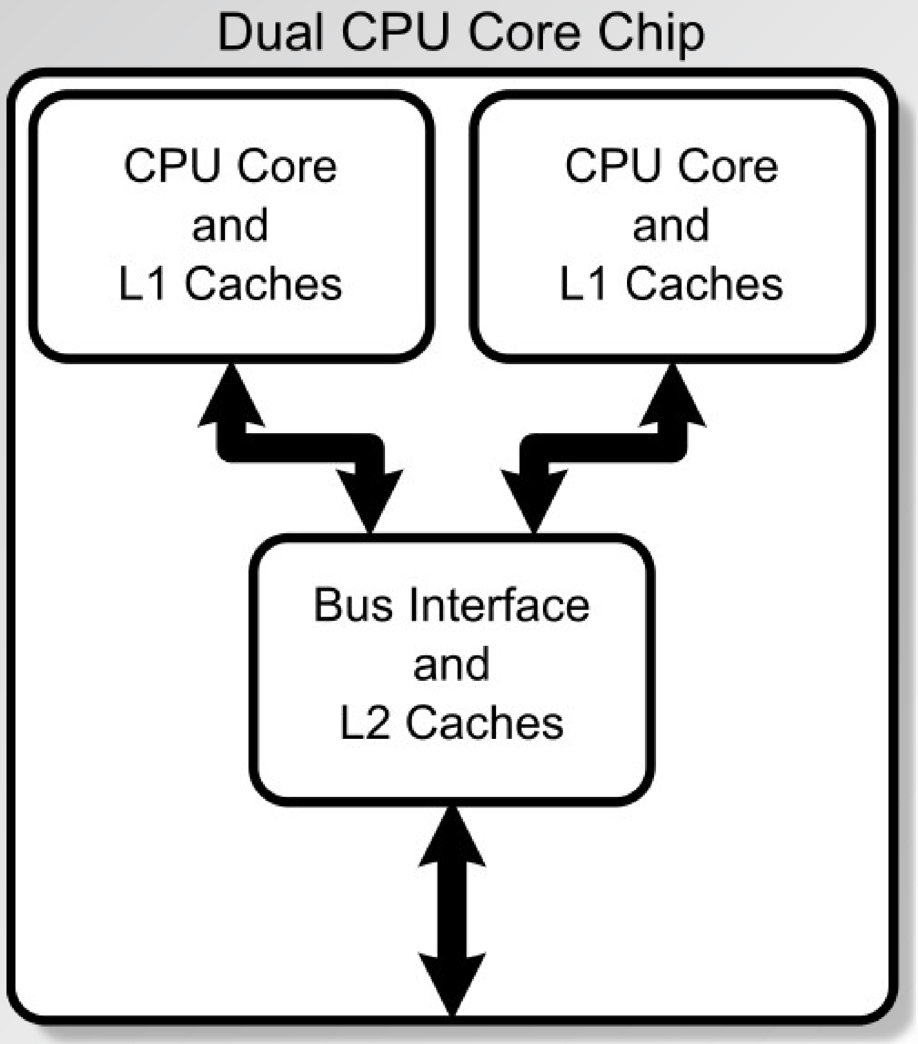
Running Faster v3: Superscalar

- Superscalar CPU => instruction level parallelism
- Some redundant functional units in single CPU => multiple instructions executed at same time
- Often combined with pipelined CPU design
- No data dependencies, no branches
- Not SIMD/SSE/MMX
- Optimization:
  => loop unrolling



Running Faster v4: Multi-core

- Maximum CPU clock rate limited by physics
- Implement multiple complete, pipelined, and superscalar CPUs into one processor
- Need parallel software to take advantage
- Memory speed limiting



The Abdus Salam
International Centre
for Theoretical Physics

How Do We Measure Performance?

- For numerical operations: FLOP/s
  = Floating-Point Operations per second
- Theoretical maximum (**peak**) performance:
  clock rate x number of double precision addition and/or multiplications completed per clock
  => 2.5 Ghz x 4 FLOP/clock = 10 GigaFLOP/s
  => can never be reached (data load/store)
- Real (**sustained**) performance:
  => very application dependent
  => Top500 uses Linpack (linear algebra)

Fast and Slow Operations

- Fast (6): add, subtract, multiply
- Medium (40): divide, modulus, sqrt()
- Slow (300): most transcendental functions
- Very slow (1000): power (x^y for real x and y)

Often only the fastest operations are pipelined, so code will be the fastest when using only add and multiply => linear algebra
=> BLAS (= Basic Linear Algebra Subroutines)
plus LAPACK (Linear Algebra Package)

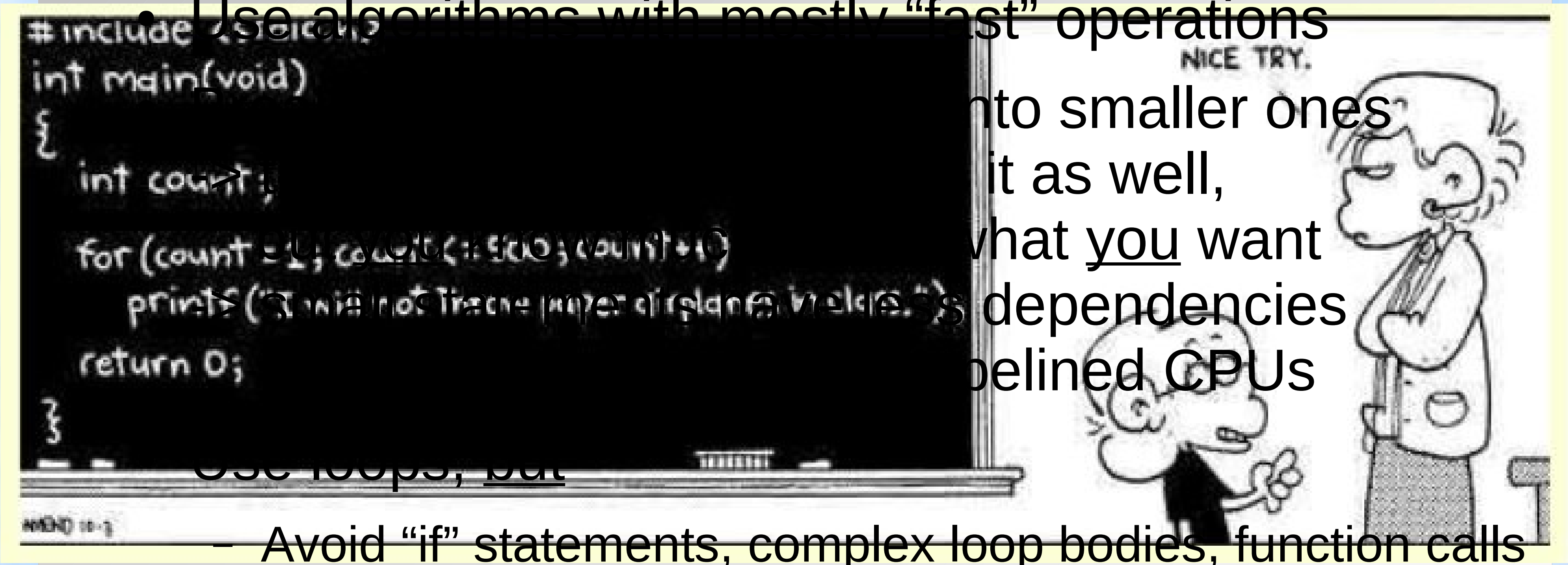
Software Optimization

- Writing maximally efficient code is hard:
  => most of the time it will not be executed exactly as programmed, not even for assembly
- Maximally efficient code is not very portable:
  => cache sizes, pipeline depth, registers, instruction set will be different between CPUs
- Compilers are smart (but not too smart!) and can do the dirty work for us, but can get fooled

  => modular programming: generic code for most of the work plus well optimized kernels

Tips For Efficient Software

- Write "compiler-friendly" code:
  - Use algorithms with mostly "fast" operations



nto smaller ones
it as well,
what you want
dependencies
pelined CPUs

Use loops, but

  - Avoid "if" statements, complex loop bodies, function calls
  - Try to access data in forward order, not random
- Use kernels in optimized (performance) libraries

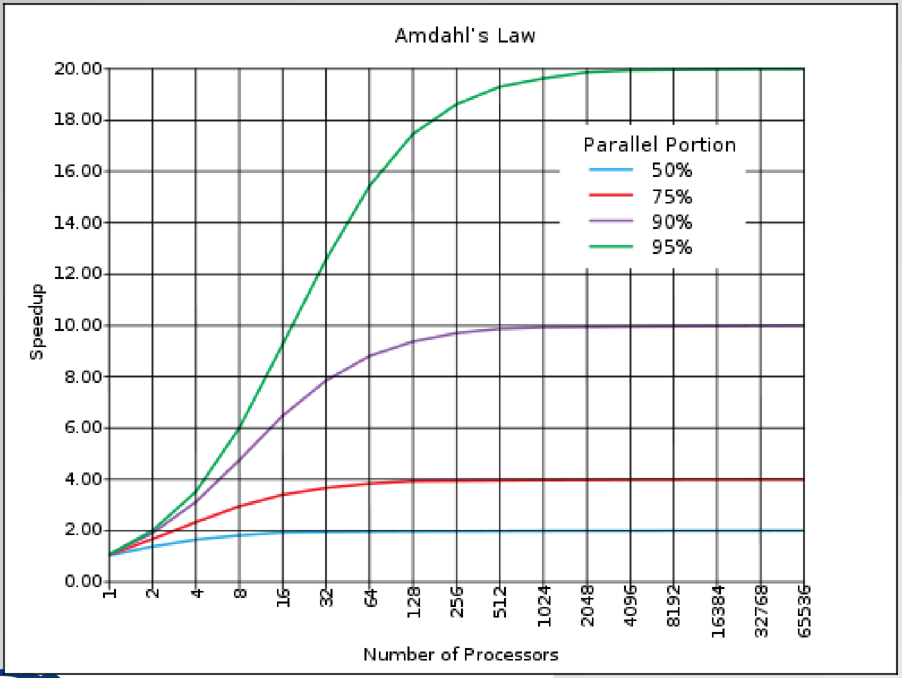
A High-Performance Problem

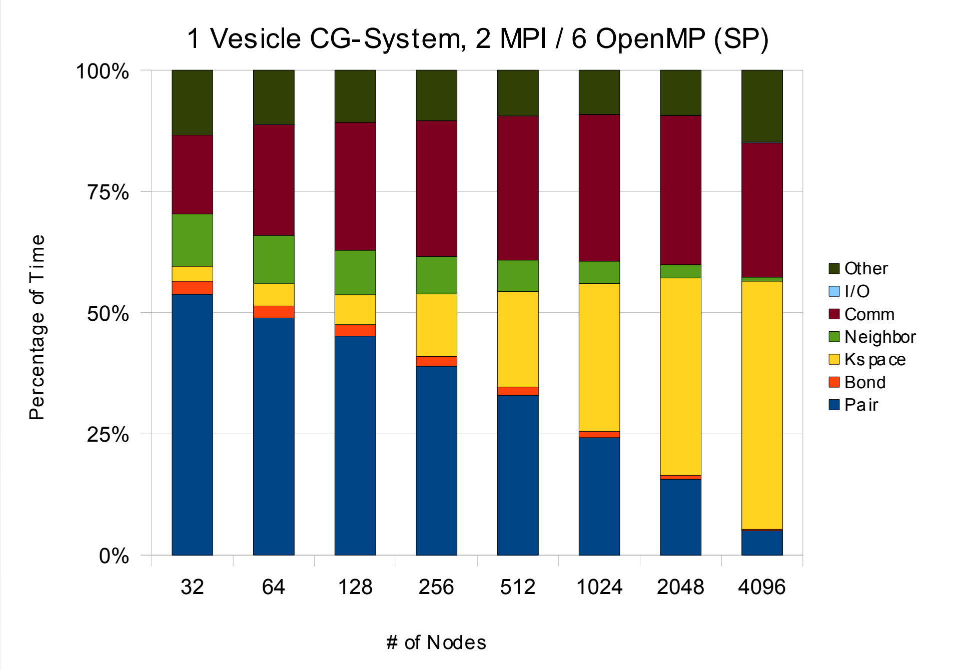
Two Types of Parallelism

- Functional parallelism: different people are performing different tasks at the same time
- Data parallelism: different people are performing the same task, but on different equivalent and independent objects

Amdahl's Law vs. Real Life

- The speedup of a parallel program is limited by the sequential fraction of the program.
- This assumes perfect scaling and no overhead

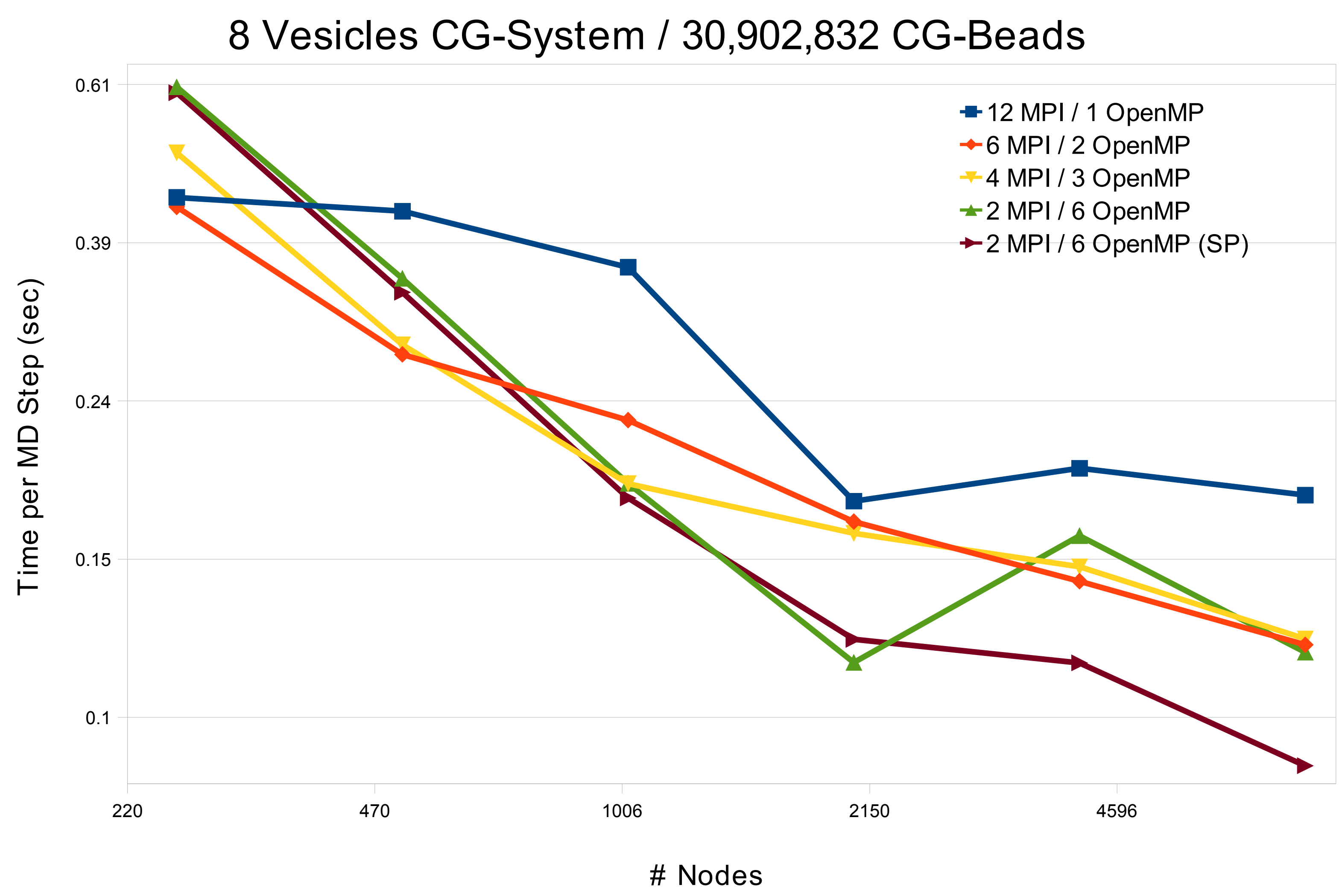




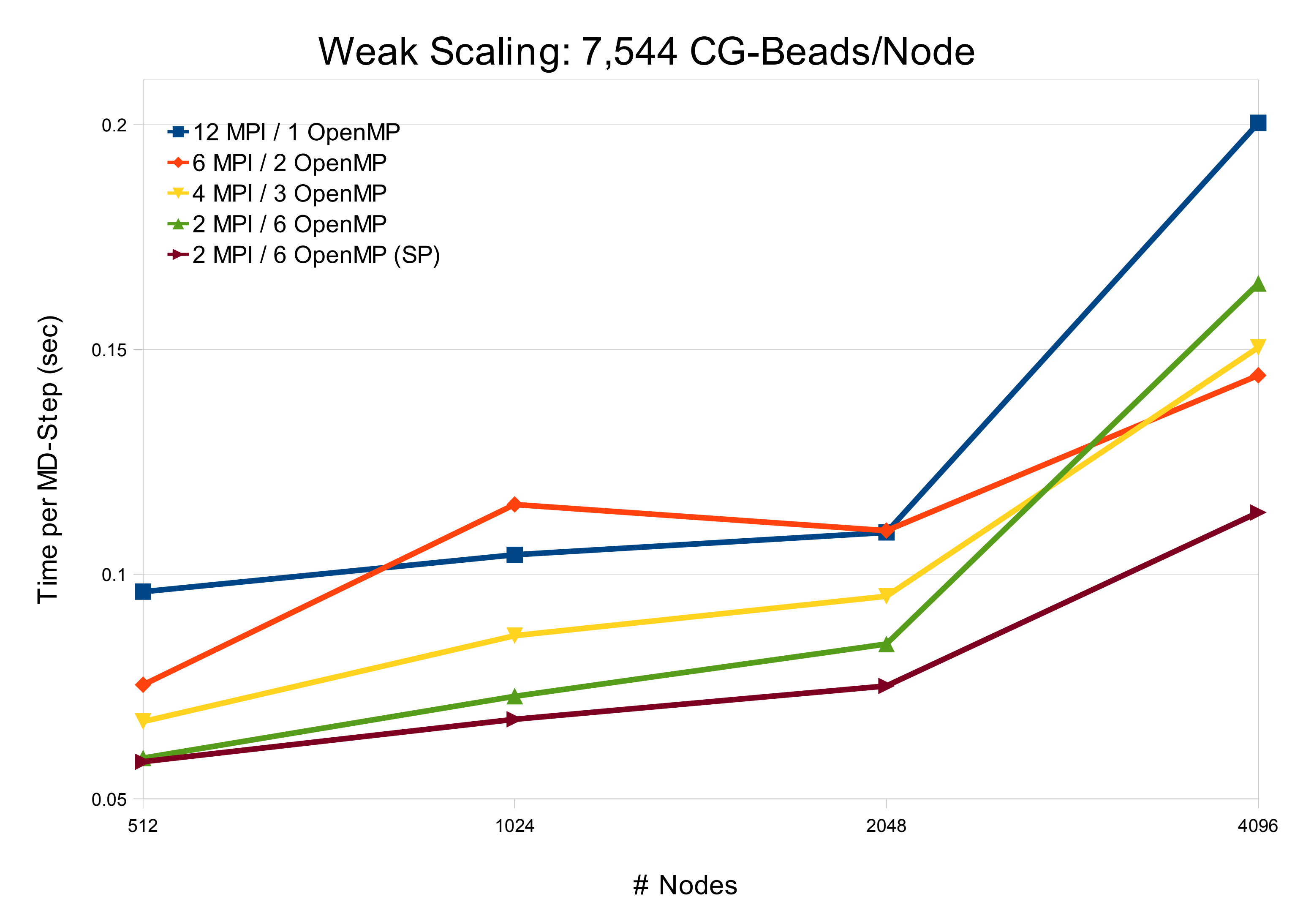
Performance of SC Applications

- Strong scaling: fixed data/problem set; measure speedup with more processors
- Weak scaling: data/problem set increases with more processors; measure if speed is same
- Linpack benchmark: weak scaling test, more efficient with more memory => 50-90% peak
- Climate modeling (WRF): strong scaling test, work distribution limited, load balancing, serial overhead => < 5% peak (similar for MD)

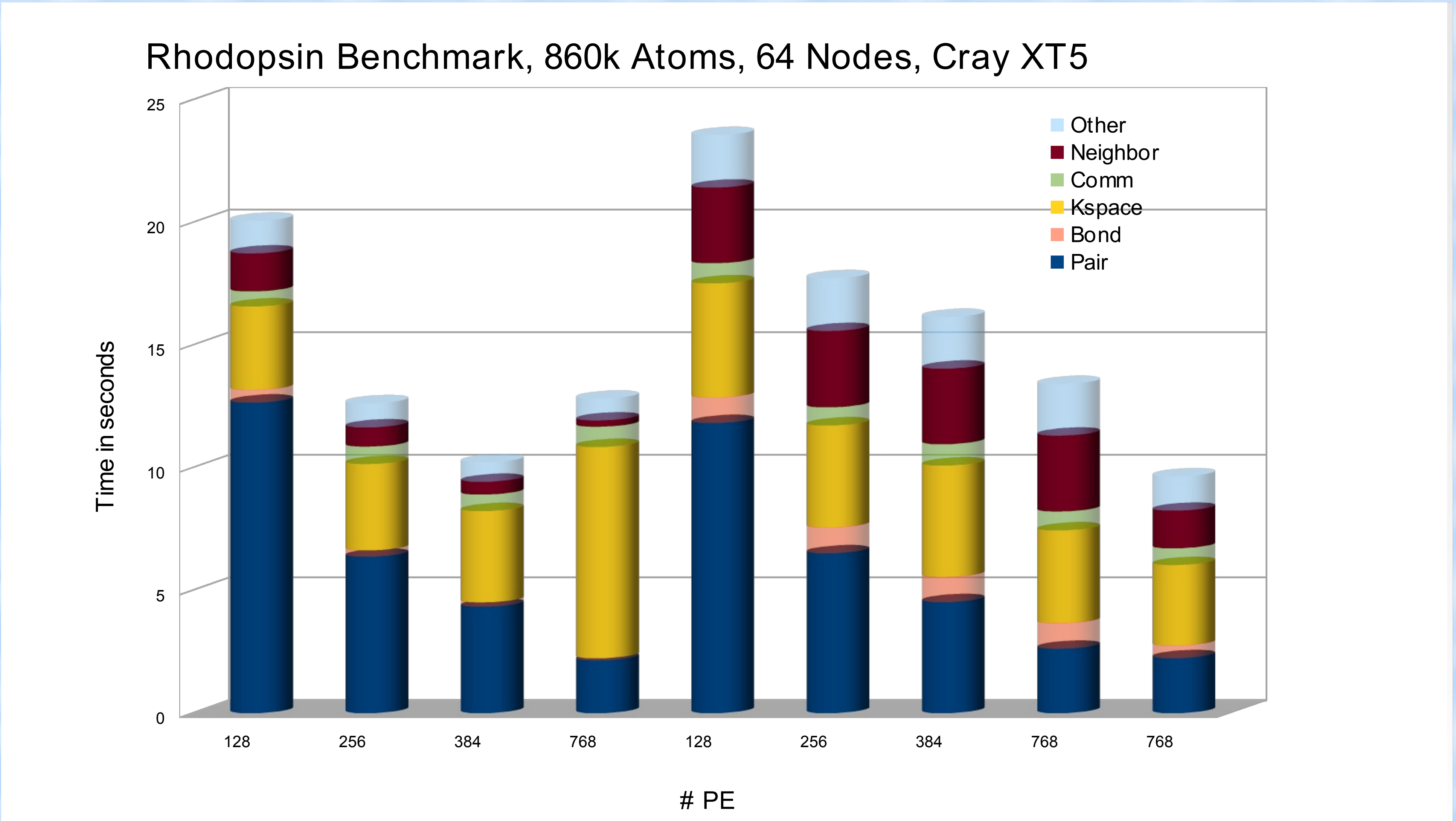
Strong Scaling Graph



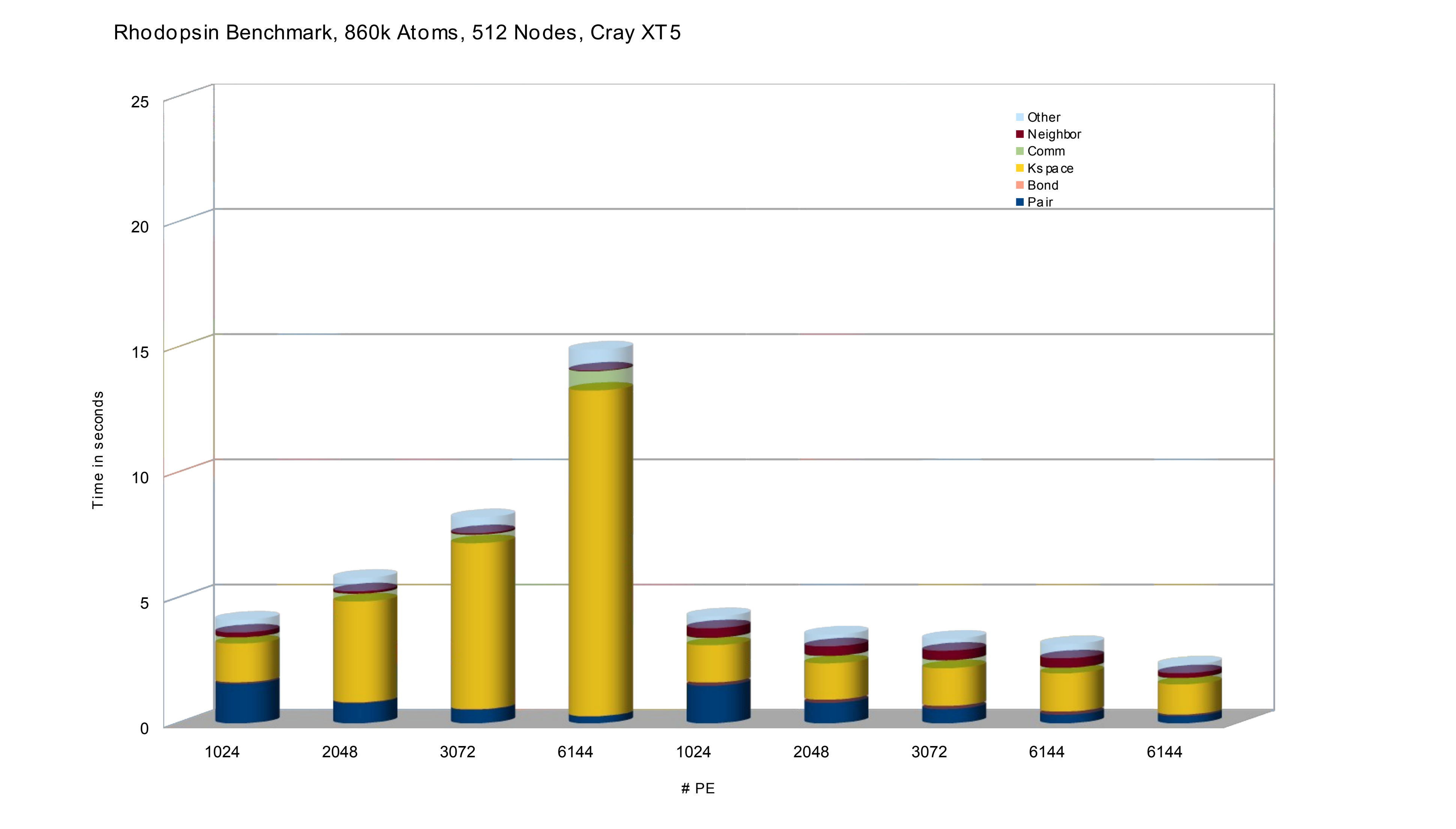
Weak Scaling Graph



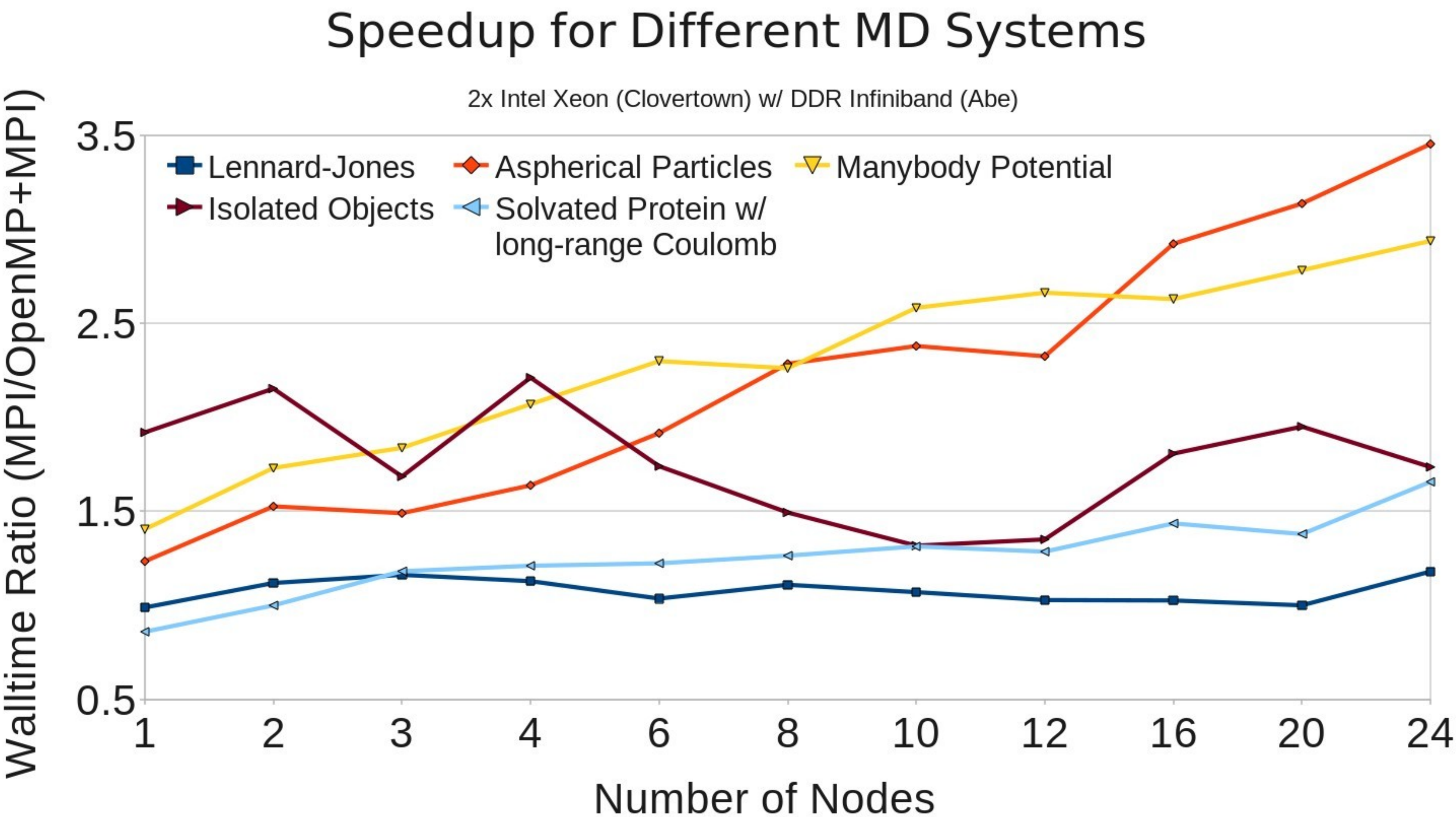
Performance within an Application



Multi-core MPI Performance vs. MPI+OpenMP



Parallel Efficiency vs. Physics

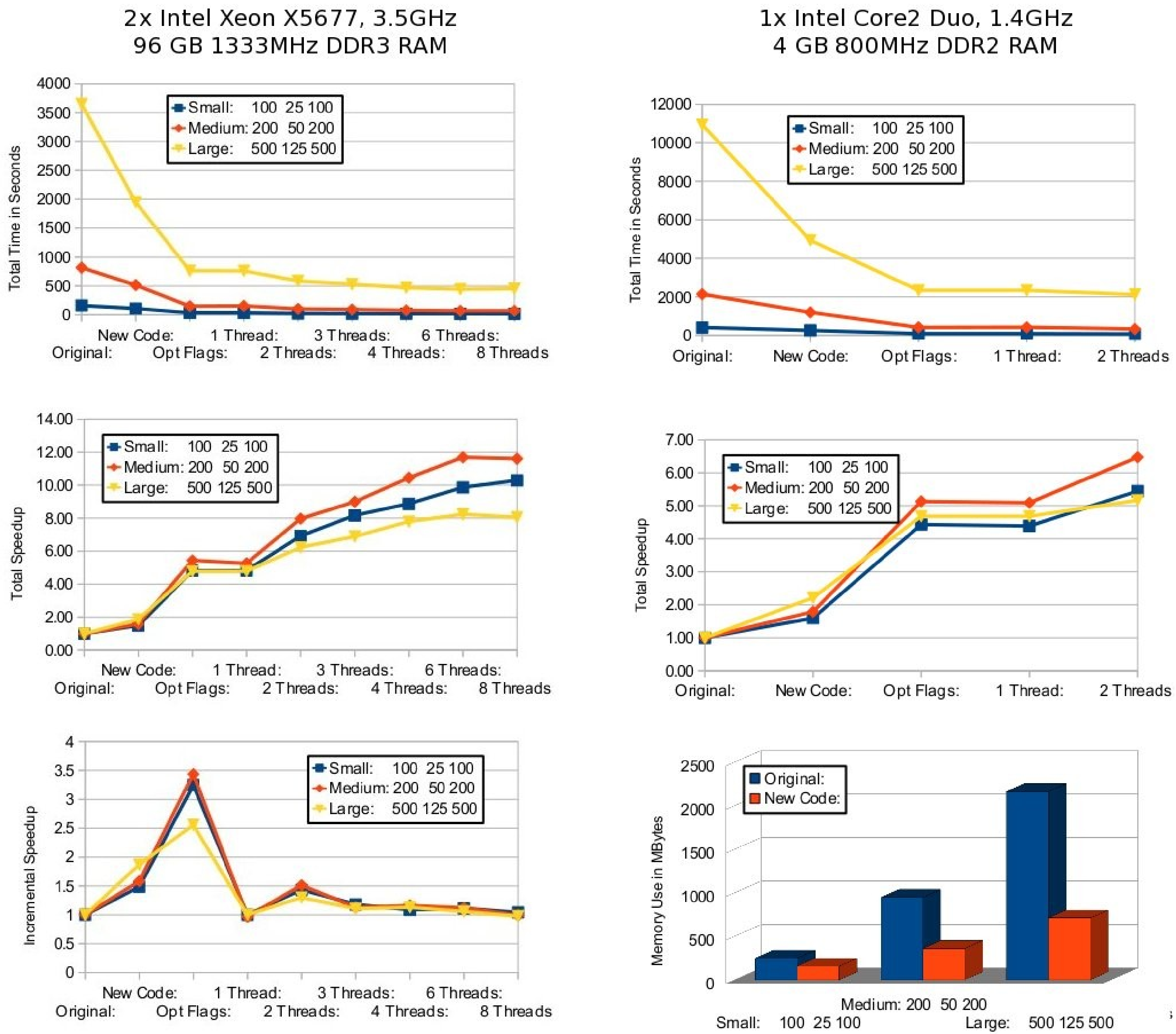


A Real Life HPC Problem

- C code to study relations in social networks
- Two steps:
  1) construct a large matrix with yes/no information (1 or 0)
  2) process matrix by pruning lines and inserting corresponding entries into a second matrix
- Input parameters for block sizes (relation depth)
- 80% of time in one (small) subroutine
- Program too slow and needs too much RAM

What To Do

- Profiling to confirm performance info (true, except for very large blocks, then a different step becomes dominant)
- Since only 1/0 information is stored, replace “unsigned long” (64-bit) with “char” (8-bit)
- Add OpenMP multi-threading, since critical subroutine has loops that are suitable
- Test on different hardware to determine sensitivity to CPU vs. memory performance


2x Intel Xeon X5677, 3.5GHz
96 GB 1333MHz DDR3 RAM
1x Intel Core2 Duo, 1.4GHz
4 GB 800MHz DDR2 RAM
Total Time in Seconds
Small: 100 25 100
Medium: 200 50 200
Large: 500 125 500
Original:
New Code:
Opt Flags:
1 Thread:
2 Threads:
3 Threads:
4 Threads:
6 Threads:
8 Threads
2 Threads
Total Speedup
Incremental Speedup
Memory Use in MBytes
Original:
New Code:
Small: 100 25 100
Medium: 200 50 200
Large: 500 125 500

Introduction to High-Performance Computing

Dr. Axel Kohlmeyer

Scientific Computing Expert

Information and Telecommunication Section
The Abdus Salam International Centre
for Theoretical Physics

http://sites.google.com/site/akohlmey/

akohlmey@ictp.it

The Abdus Salam
International Centre
for Theoretical Physics