# Introduction to High-Performance Computing


## Dr. Axel Kohlmeyer

Assistant Dean for High-Performance Computing
Associate Director, ICMS
Associate Director, TMI
College of Science and Technology
Temple University, Philadelphia
**axel.kohlmeyer@temple.edu**

External Scientific Associate
International Centre for Theoretical Physics, Trieste, Italy
**akohlmey@ictp.it**

# 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
  - A 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 desktops today are parallel workstations => multi-core processors (up to 64 cores soon)
- Running Linux OS (or MacOS X) allows programming like traditional Unix workstation (but Apple is gradually making things different, OTOH containers make them more uniform)
- 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 needs:
  - Several computers, often in special cases for easy mounting in a rack (one node ~= one mainboard)
  - One or more networks (interconnects) to access the nodes and for inter-node communication
  - Software that orchestrates communication between parallel processes on the nodes (e.g. MPI)
  - Software that reserves resources to individual users
- A cluster is: all of those components working together to form one big computer

# What is Grid Computing?

- Loosely coupled network of compute resources, conceived to process large amounts of data
- Needs a “middleware” for transparent access for inhomogeneous resources
- 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 or container on shared resource as needed with custom OS image, commercialization (cloud service providers, dedicated or spare server resources), physical location flexible

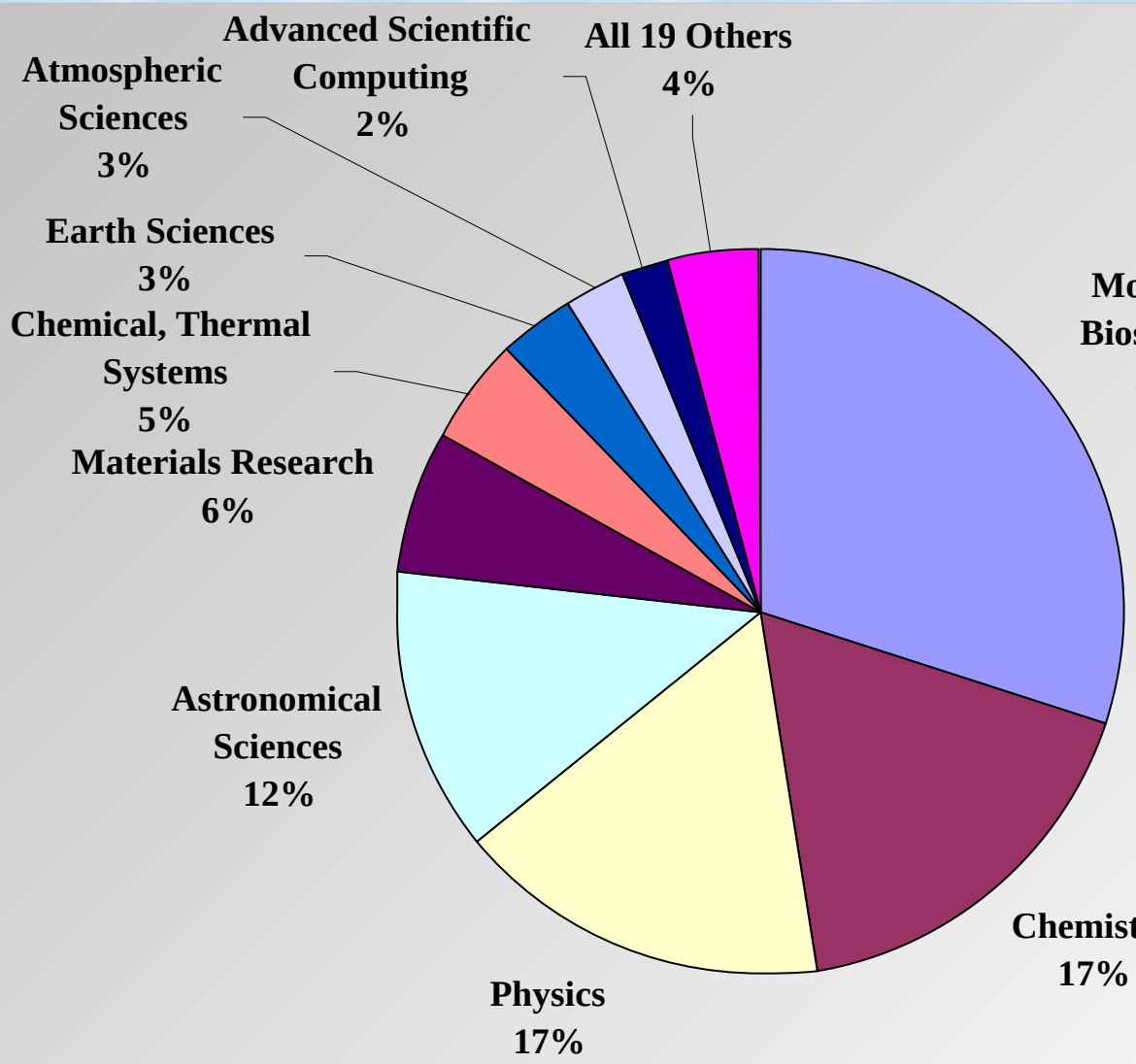
# What is Supercomputing (SC)?

- The most visible manifestation of HPC (=> Top500 List)
- Is “super” due to large size, extreme technology
- Desktop vs. Supercomputer in 2019 (peak, DP):
  - Desktop processor (1 core): ~50 GigaFLOP/s
  - Tesla V100 GPU: >7 TeraFLOP/s
  - #1 supercomputer on Top500: >200 PetaFLOP/s
- Sustained vs. Peak: “K” 93%, “Summit” 74%, “Tianhe-2a” 61%, Cluster 65-90%

# 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
  → complex SW packages with dependencies
- Scientific (HPC) application development often limited by lack of training
- More knowledge about HPC leads to more effective use of HPC resources and better interactions with colleagues

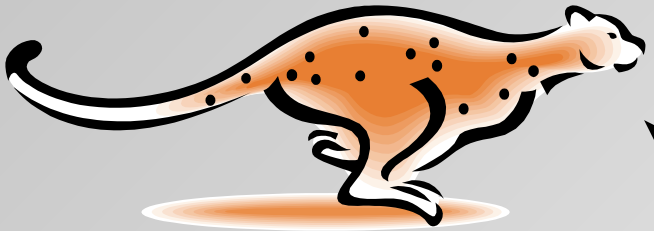
# Research Disciplines in HPC

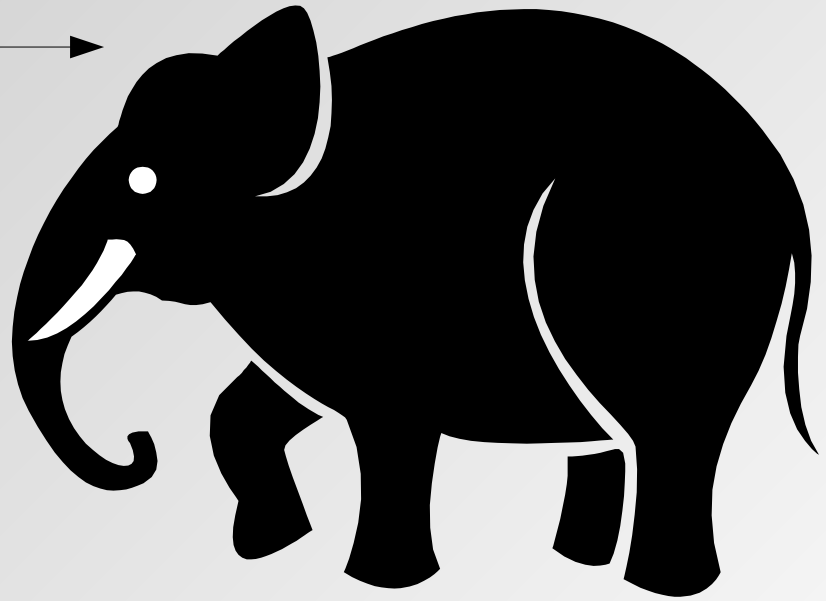



XSEDE
Extreme Science and Engineering Discovery Environment

# Why Would I Care About HPC?

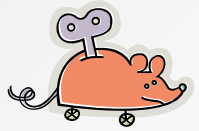
- My problem is big





- My problem is complex

- My computer is too small and too slow



- My software is not efficient and/or not parallel
  -> often scaling with system size the problem

# HPC vs. Computer Science

- Most people in HPC are not 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”

# My Background

- Undergraduate training as chemist (physical & organic), PhD in Theoretical Chemistry, University Ulm, Germany
- Postdoctoral Research Associate, Center for Theoretical Chemistry, Ruhr-University Bochum, Germany
- Associate Director, Center for Molecular Modeling, University of Pennsylvania, Philadelphia, USA
- Assistant Dean/Research Faculty in Math/Chemistry,CST, Associate Director, Inst. for Comp. Molecular Science, Temple University, Philadelphia (2009-2012, since 2014)
- Scientific Computing Expert, International Centre for Theoretical Physics, (2012/13); now external associate
- Lecturer at ICTP/SISSA International Master for HPC

# 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
  => but 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 performance for convenience
  (e.g. compiled program vs. script program)
- Delegate parts of the work
  => parallelize code, (grid/batch computing)
  => use accelerators (GPU/MIC CUDA/OpenCL)

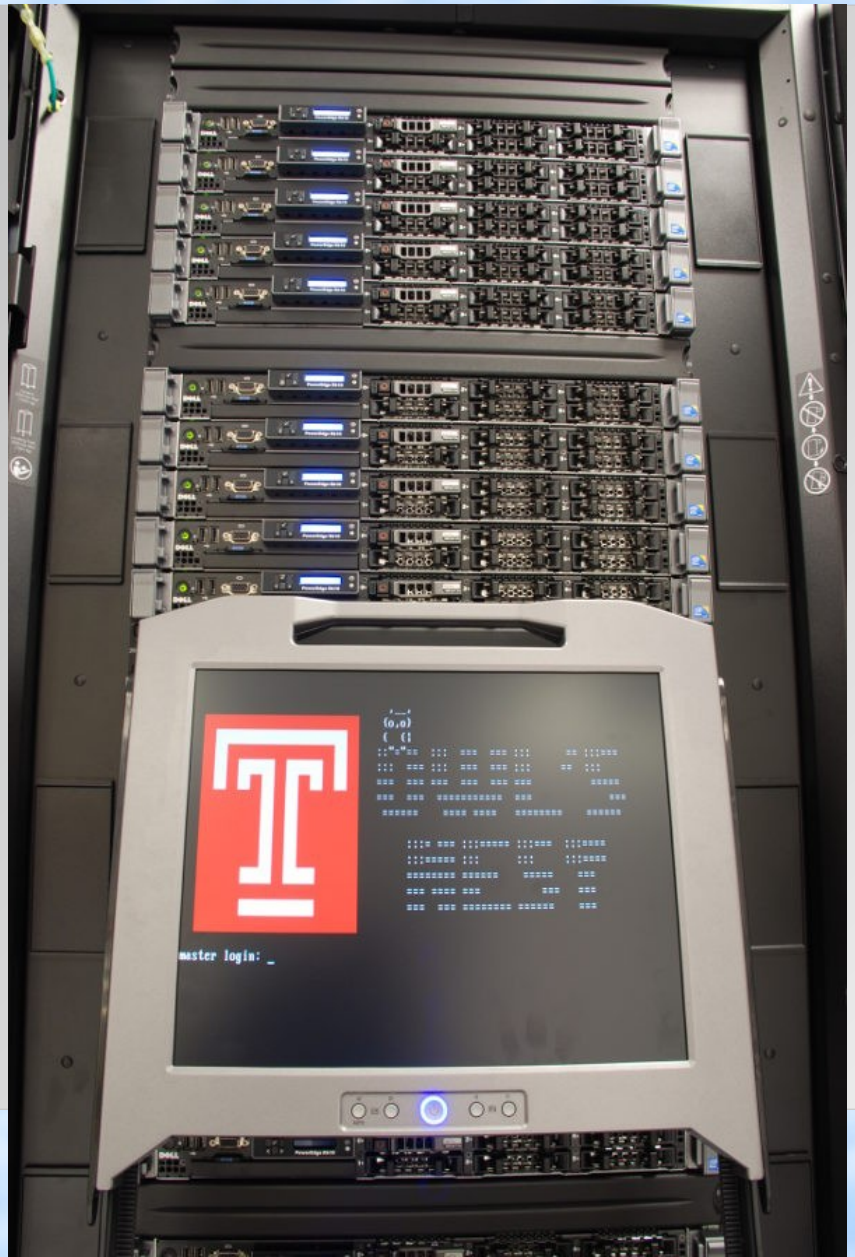
# How Do We Measure Performance?

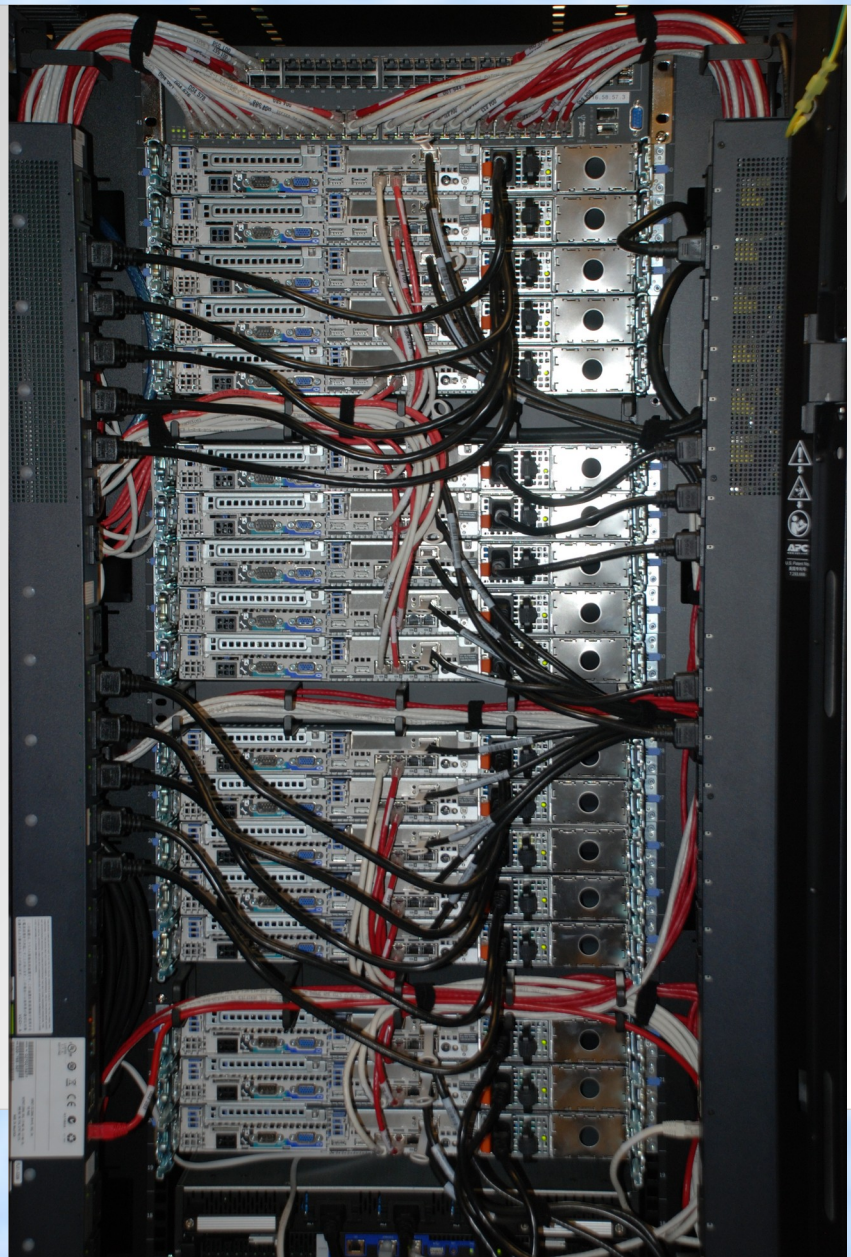
- For numerical operations: FLOP/s
  = Floating-Point Operations per second
- Theoretical maximum (**peak**) performance:
  clock rate x number of double precision addition and/or multiplications completed per clock
  => 2.5 Ghz x 8 FLOP/clock = 20 GigaFLOP/s
  => can never be reached (data load/store)
- Real (**sustained**) performance:
  => very application dependent
  => Top500 uses Linpack (linear algebra)

# HPC Cluster in 2002 / The Good

# HPC Cluster in 2002 / The Bad

# HPC Cluster in 2012



# HPC Cluster in 2019

## >250 nodes, >6500 cores, >2PB storage

# A High-Performance Problem

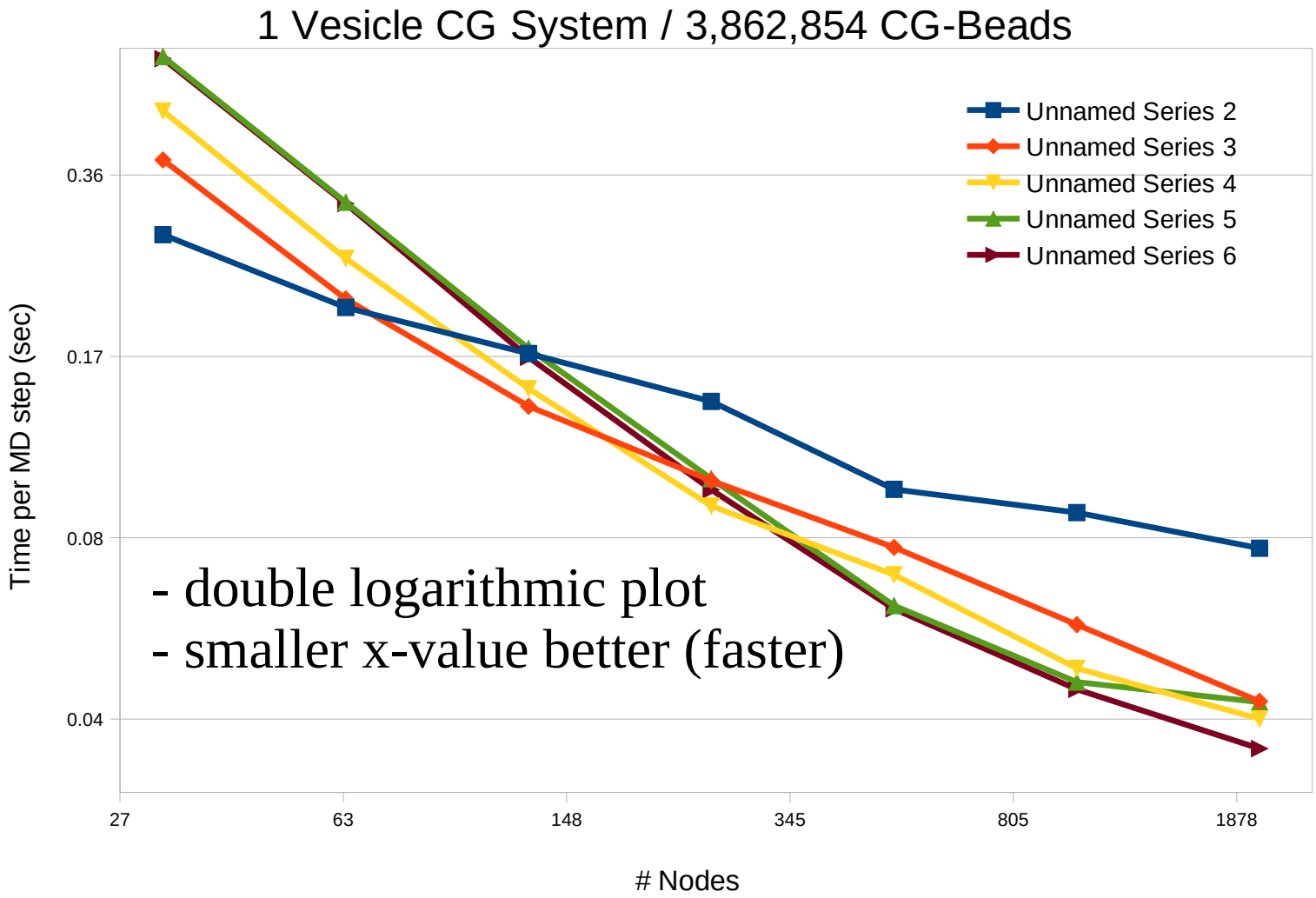
# Two Types of Parallelism

- 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

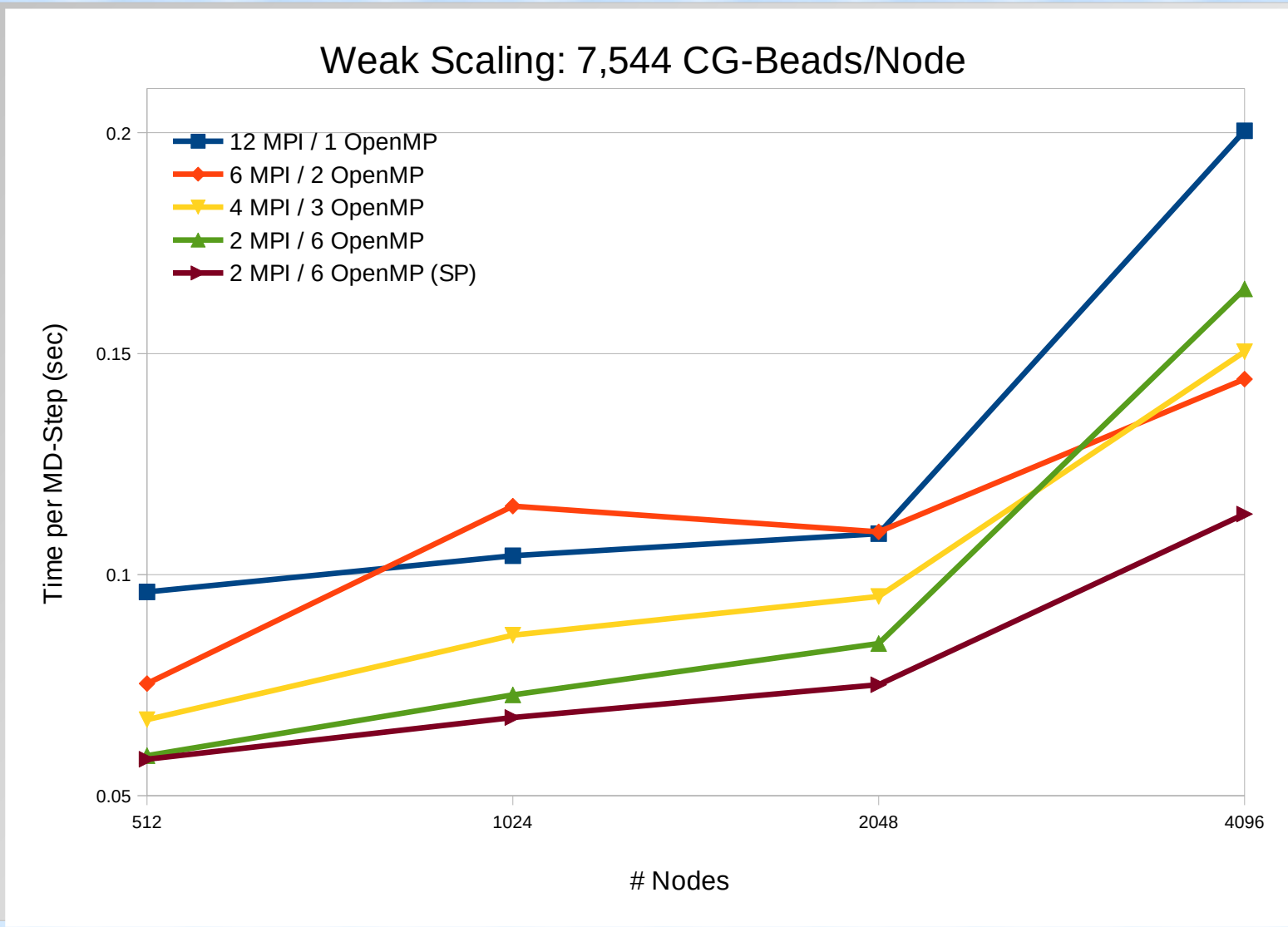
# Performance of SC Applications

- 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)

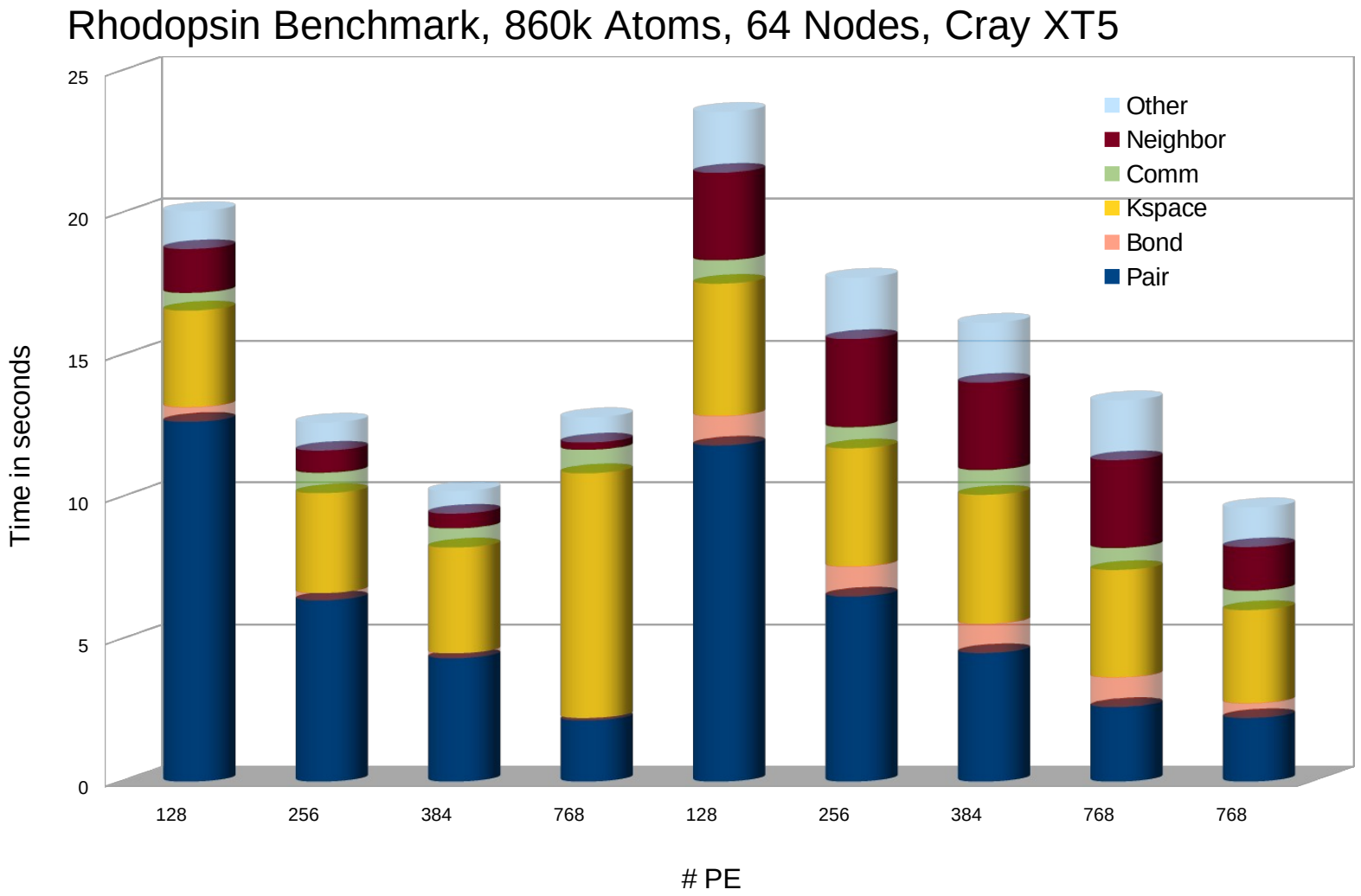
# Strong Scaling Graph



# Weak Scaling Graph

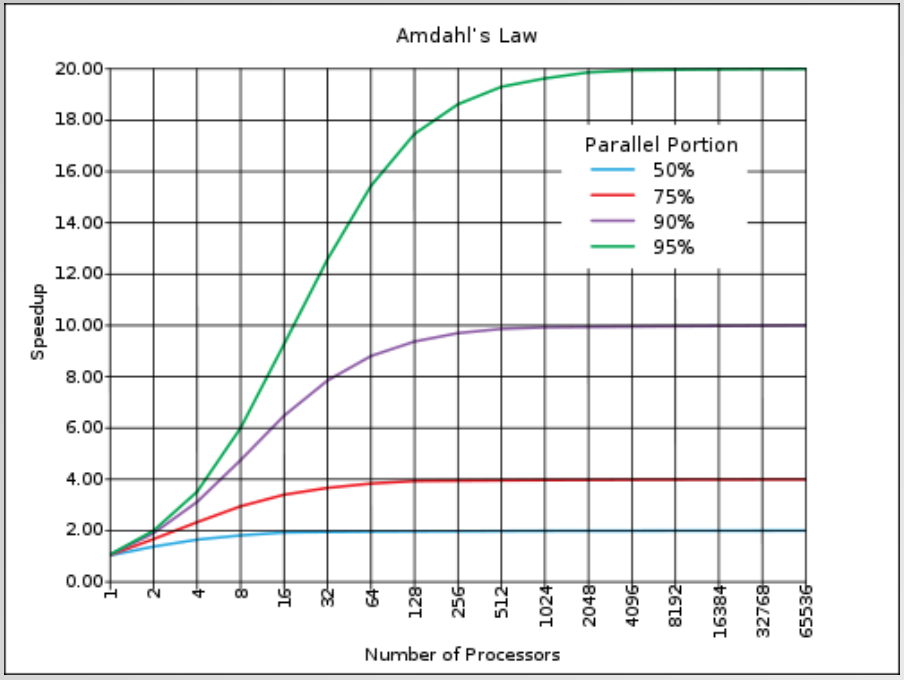


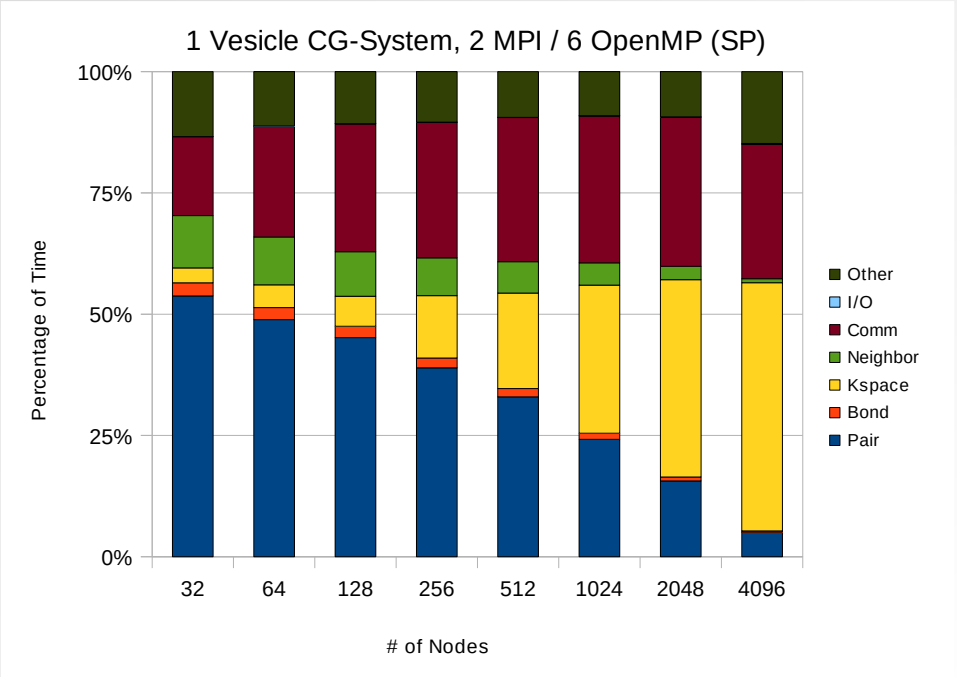
# Performance within an Application



# 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



# 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

# Workflows

- Maximally optimized software and running on a top supercomputer is not always enough
- Scientific computational projects are increasingly multi-step procedures with dependencies or where individual calculations parallelize differently or run at different speed
- The human effort to organize calculations and the margin for errors grows with complexity
- Thus need for tools to manage workflows and optimize the work distribution on resources

# Workflow Optimization Example

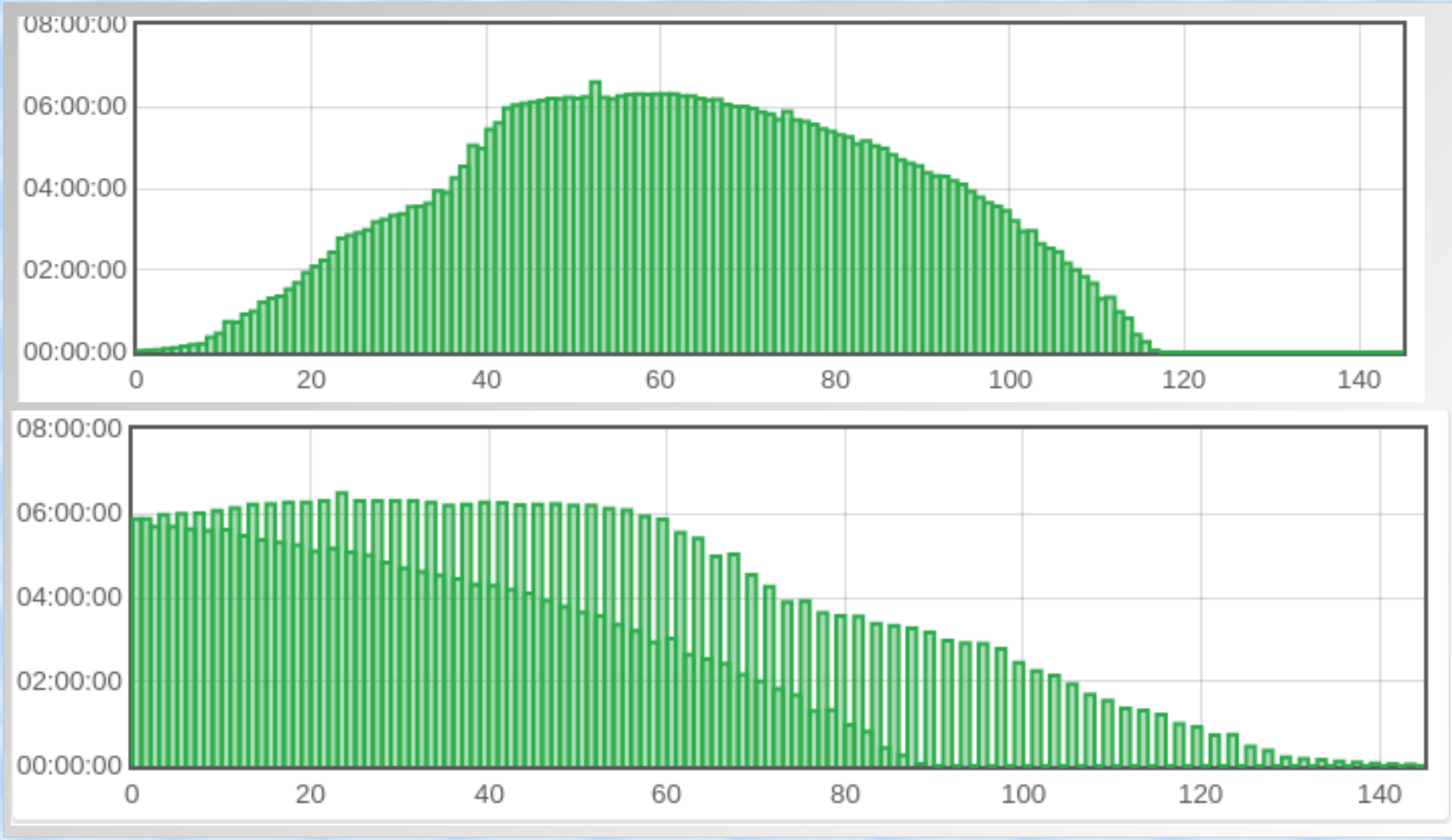
- FSL = Software package for fMRI, MRI, and DTI brain imaging data analysis
- Uses shell script “drivers” to perform complex tasks through calling individual excutables
- Parallelized through processing chunks of data
- Need to adjust HPC cluster support for our parallel computing optimized machine

  => adapt scripts to be run inside batch job

  => make use of custom dispatch tool

  => 50% speedup through replacing math library

# Workflow Optimization Example



# Introduction to High-Performance Computing


## Dr. Axel Kohlmeyer

Assistant Dean for High-Performance Computing
Associate Director, ICMS
Associate Director, TMI
College of Science and Technology
Temple University, Philadelphia
**axel.kohlmeyer@temple.edu**

External Scientific Associate
International Centre for Theoretical Physics, Trieste, Italy
**akohlmey@ictp.it**