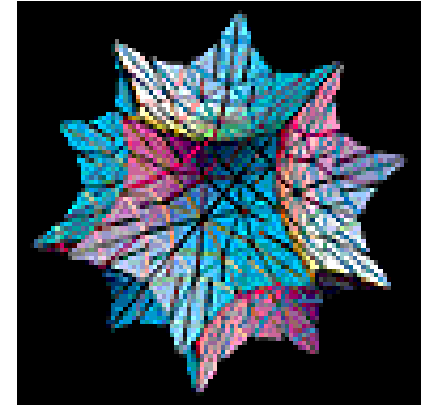
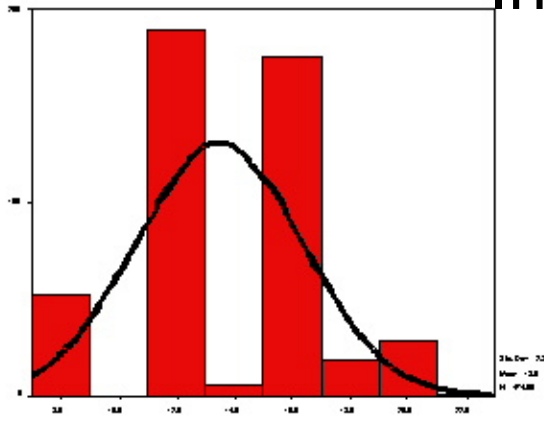


# Introduction to Parallel Computing

February 1, 2006



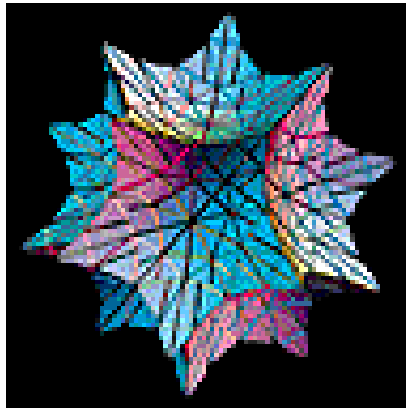
Presented by the

TTC Research Computing Support Group

Kathy Gerber, Ed Hall, Katherine Holcomb, Nancy Kechner, Tim F. Jost Tolson

- Implementing Parallel Codes by Katherine Holcomb– Wednesday, February 15 at 3:30 PM

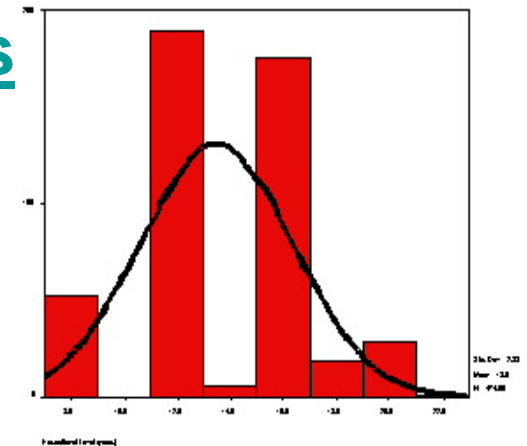
# ITC Research Computing Support Introduction to Parallel Computing



By: Katherine Holcomb  
Research Computing Support Center  
Phone: 243-8800 Fax: 243-6604

E-Mail: [Res-Consult@Virginia.EDU](mailto:Res-Consult@Virginia.EDU)

<http://www.itc.Virginia.edu/researchers>



# Why Compute in Parallel?

Most large scientific problems  
exceed the capabilities of a single  
processor

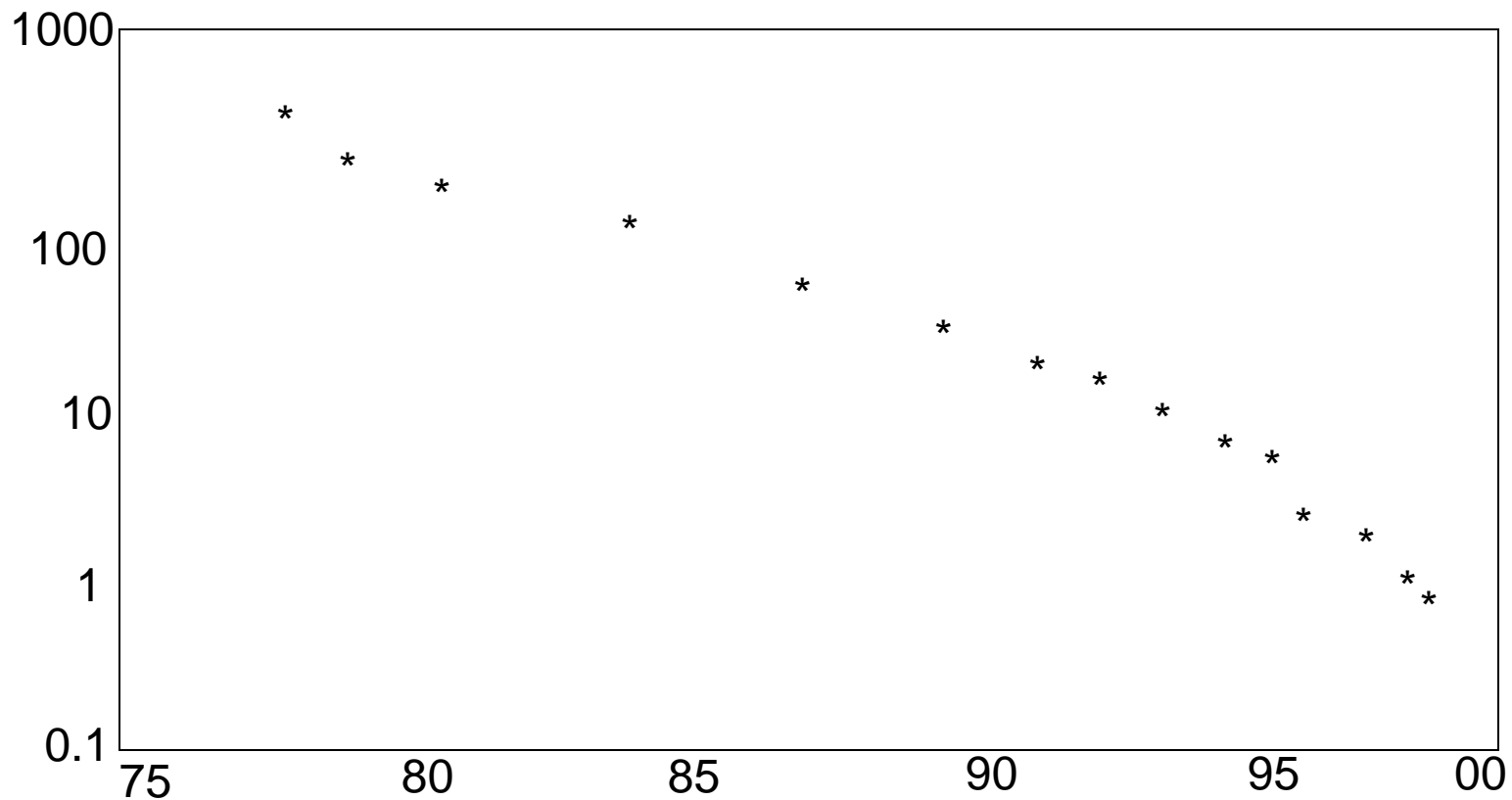
**CPU clock cycle times continue to fall roughly according to Moore's Law – halving approximately every 18-24 months.**

**BUT**

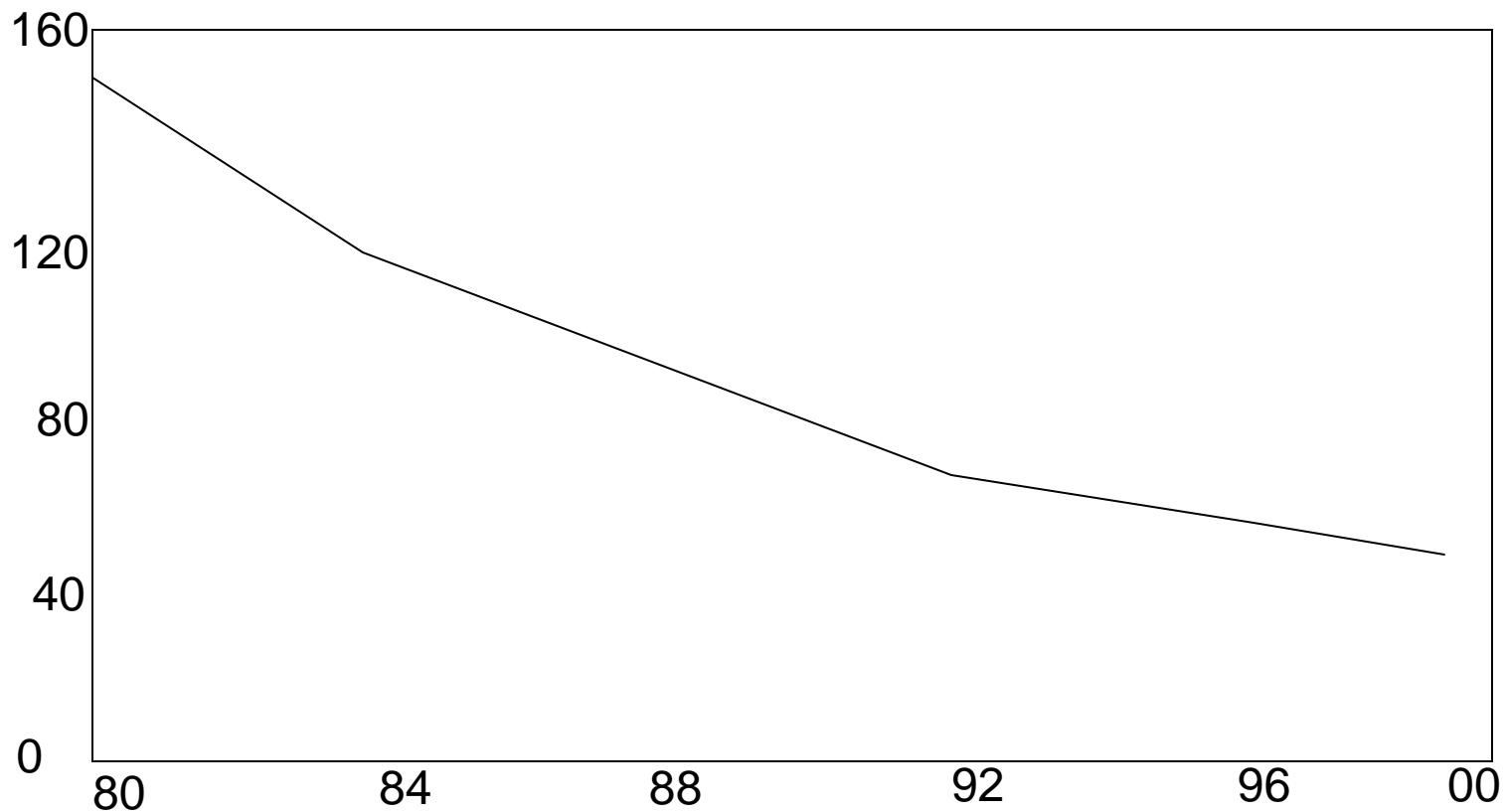
**Decrease in memory latency (the time required to access the first bit of information) is not keeping up.**

**Other factors: disk capacities have increased enormously, but they are mechanical devices and their latencies decrease fairly slowly.**

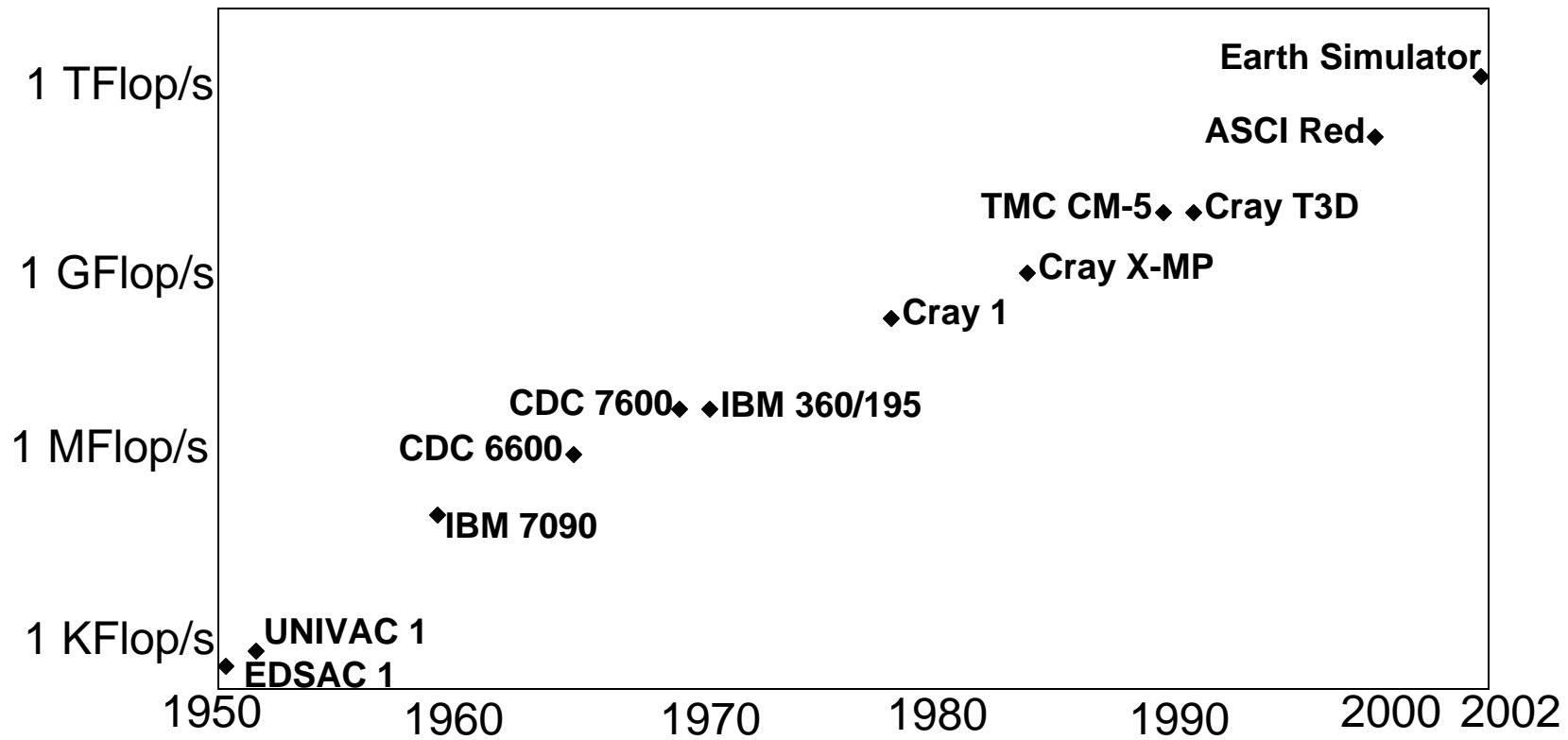
**Network speeds (internal and external) have also increased fairly slowly, and this speed is ultimately limited by light travel times across the network.**



**Clock rate in nanoseconds versus time in years**  
**Note the logarithmic scale on the vertical axis**



**DRAM latency in nanoseconds versus time in years  
Here the scale on the vertical axis is linear**



## Peak performance of supercomputers versus time

Flop/s = floating-point operations/second

**Peak performance for supercomputers has followed Moore's Law quite well, but only about half the improvement is due to increases in single-processor speed, while the rest is due to an increasing number of processors.**

**In many cases, parallelism has been the only means by which large computational runs can be performed.**

# Scalability

Parallelizing a code does not always result in a speedup; sometimes it actually slows the code down! This can be due to a poor choice of algorithm or to poor coding.

Define the speedup to be

$$\frac{T(1)}{T(N)}$$

where  $N$  = number of processors,  $T(1)$  = time for serial run.

The best possible speedup is linear, i.e. it is proportional to the number of processors:  $T(N) = T(1)/N$ .

A code that continues to speed up reasonably close to linearly as the number of processors increases is said to be **scalable**. Many codes scale up to some number of processors but adding more processors then brings no improvement. Very few, if any, codes are indefinitely scalable.

# Amdahl's Law

It is for practical purposes impossible to parallelize all parts of a code. Let the fraction of the code that is perfectly parallel be  $p$ , so the time for the parallel part is  $T_p = pT(1)/N$ ; the time for the serial part is then  $T_s = (1-p)T(1)$ . **Amdahl's Law** says that

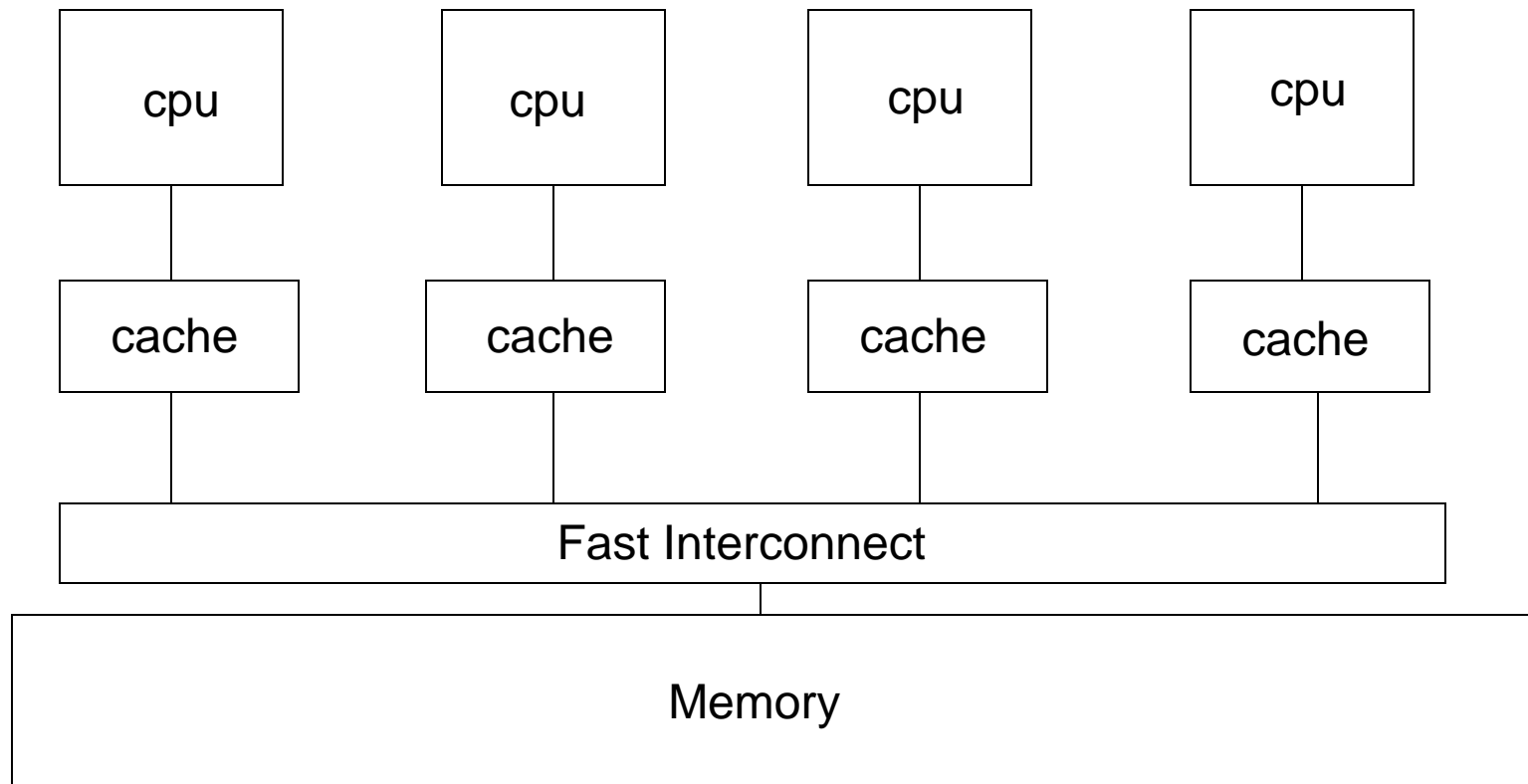
$$\text{Speedup} = \frac{1}{(1-p) + p/N}$$

which is bounded by  $1/(1-p) = T(1)/T_s$  as  $N \rightarrow \infty$ .

One way to reduce  $T_s/(T_s+T_p)$  is to increase the problem size so that the parallel parts dominate.

# Parallel Architectures

**SMP** – Symmetric Multiprocessing  
Uniform Memory Access (UMA)

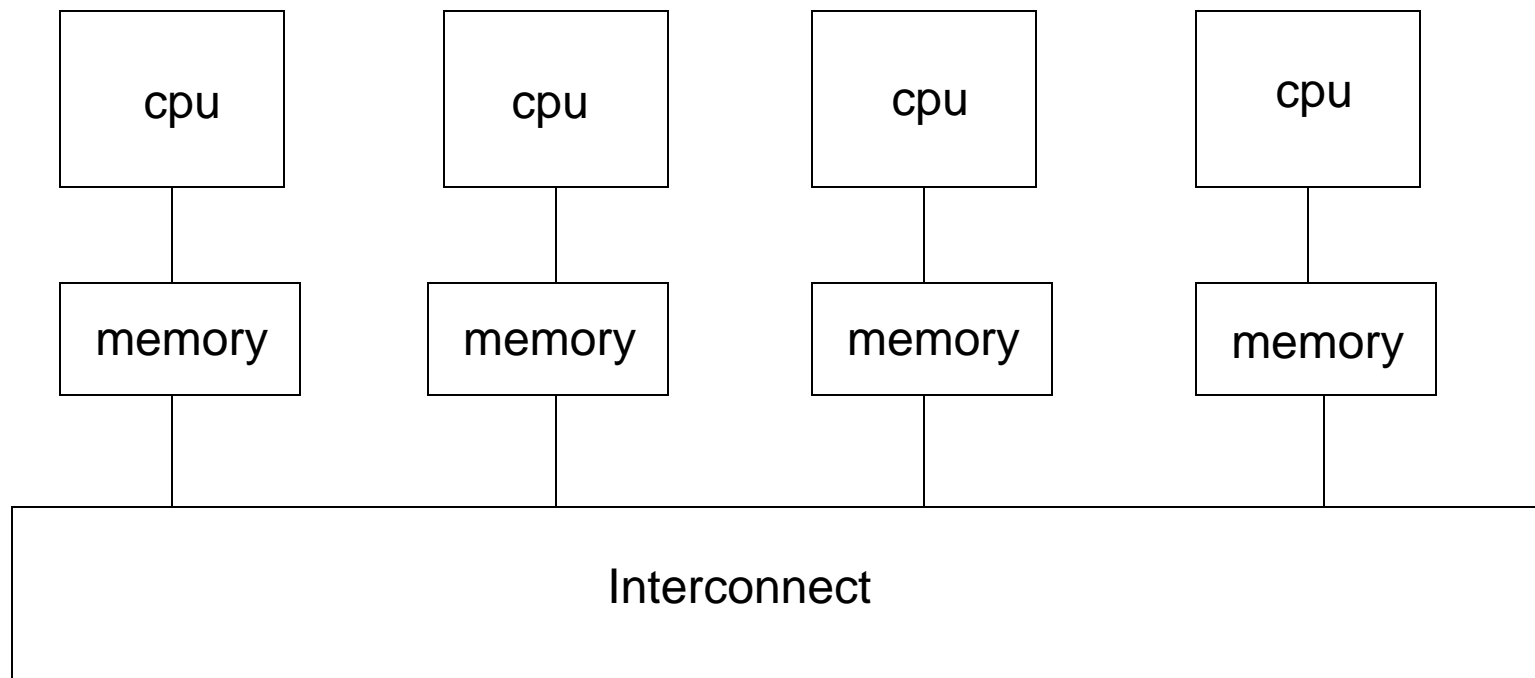


# Parallel Architectures

## MPP – Massively Parallel Processor

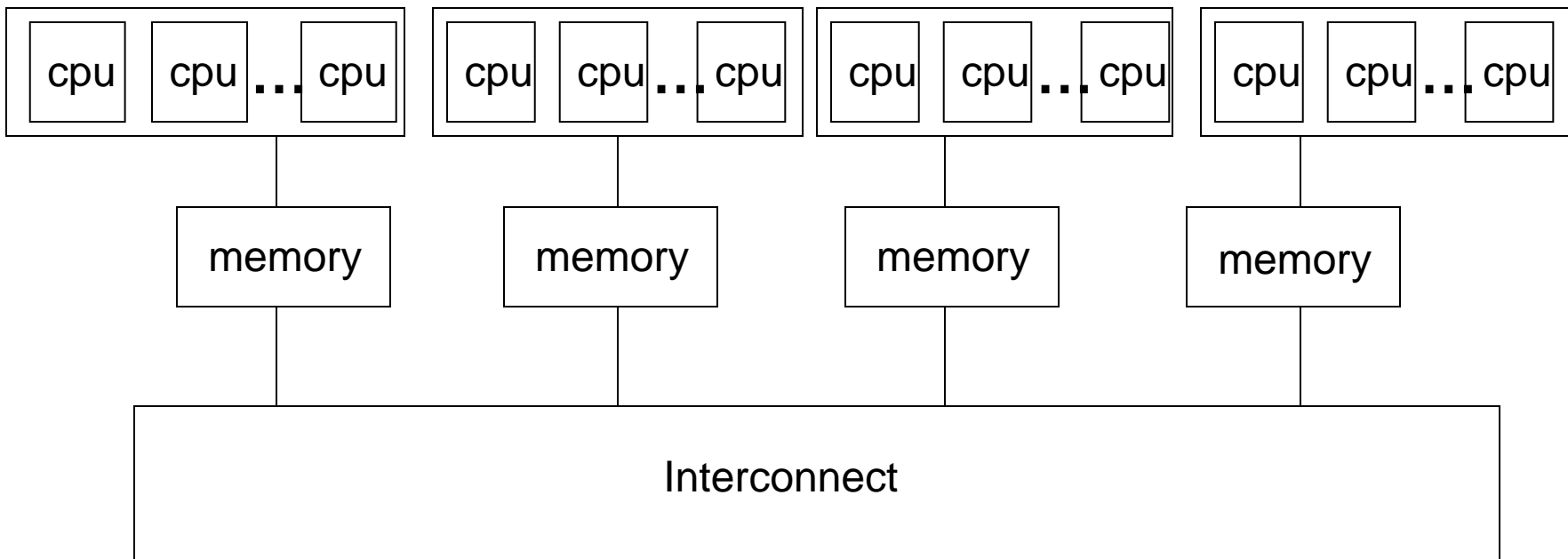
NonUniform Memory Access (NUMA): each processor can see other processors' memory

Distributed Memory: memory inaccessible to other processors



# Parallel Architectures

“Constellation”



# Computing Models

There are two dominant types of problem decomposition

- Task parallelism (MIMD)
  - divide tasks among the processors.
  - sometimes can be “embarrassingly parallel” with very little interprocess communication required.
- Data parallelism (SIMD)
  - divide data among the processors.
  - Each process executes same instructions on different data

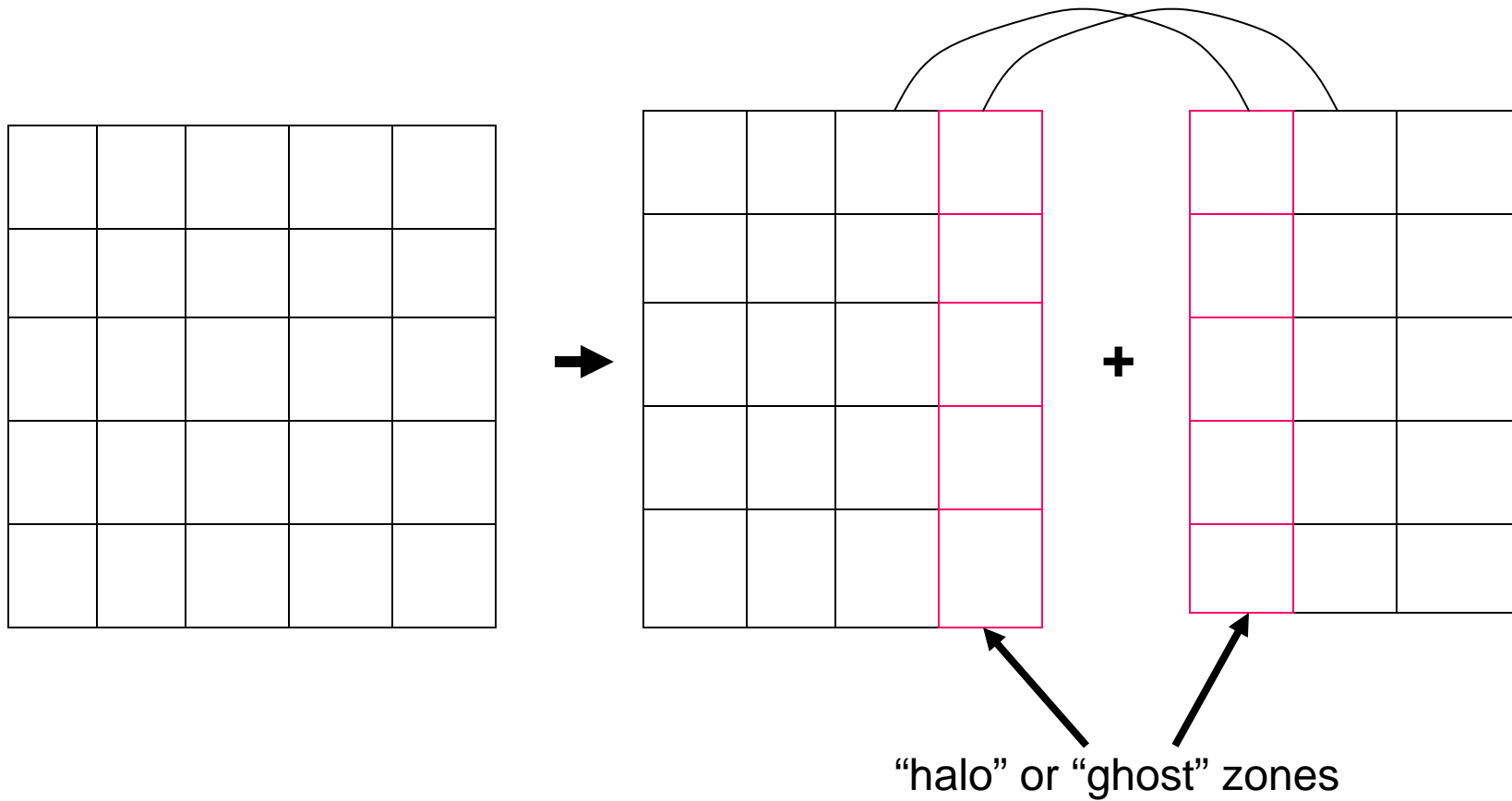
# Programming Models

- Threads/OpenMP
  - For SMP (SIMD)
  - Easy to use but can be hard to get a speedup
- Message Passing
  - For MPP or Distributed Memory Clusters
  - Corresponds to MIMD computing model
  - Harder to use but generally gives best results
  - Can be used with SMP systems with right libraries
- Hybrid
  - For “constellations”
  - Rarely results in any benefit over message passing

# Example: Distributed Data

- Many scientific and engineering discretizations involve grids or meshes.
- Sometimes this would map best to the SIMD (data parallel) model, but for MPP systems such as clusters, the problem must be forced into the MIMD (task parallel) model.
- **Domain decomposition** is the most common way to handle this type of problem.

# Block Data Decomposition

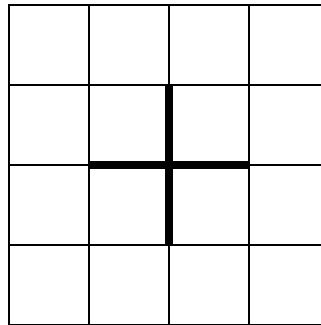


# Example: Jacobi Iteration

Laplace Equation:  $\nabla^2 T = 0$

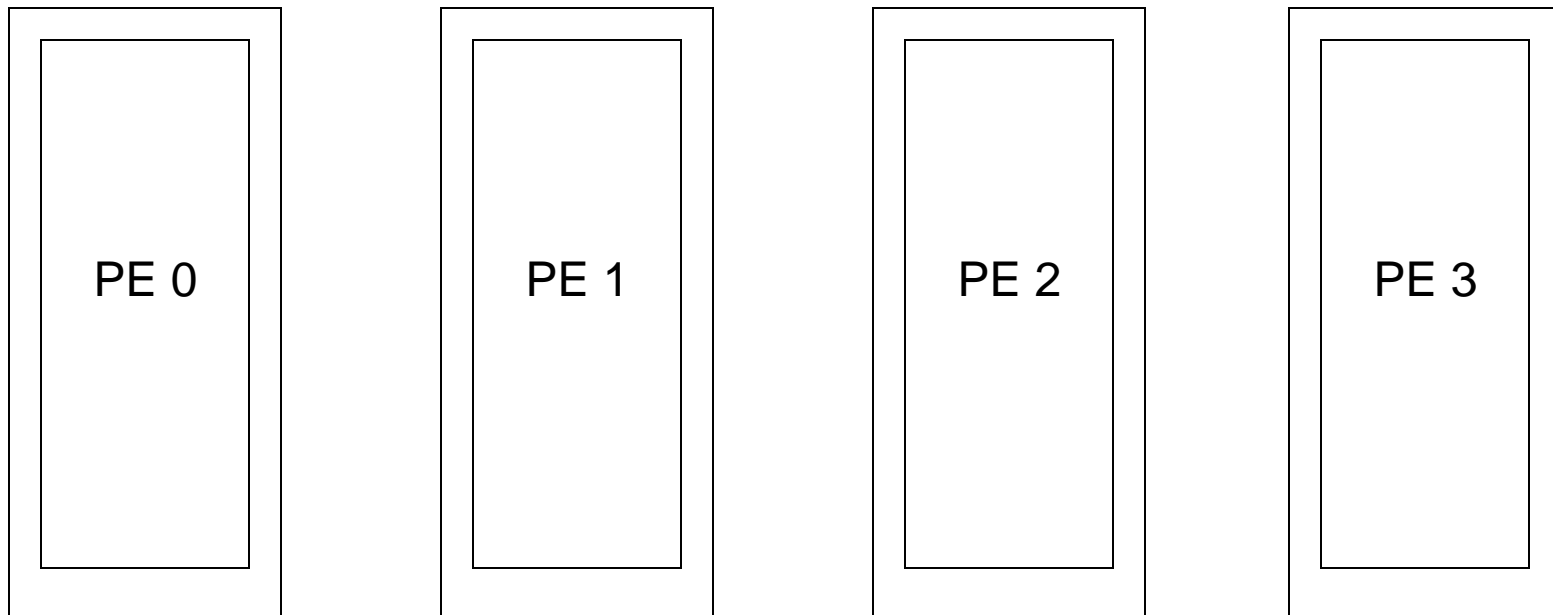
$$T_n = 0.25 * (T_{n-1}(i-1,j) + T_{n-1}(i+1,j) + T_{n-1}(i,j-1) + T_{n-1}(i,j+1))$$

This leads to a five-point stencil



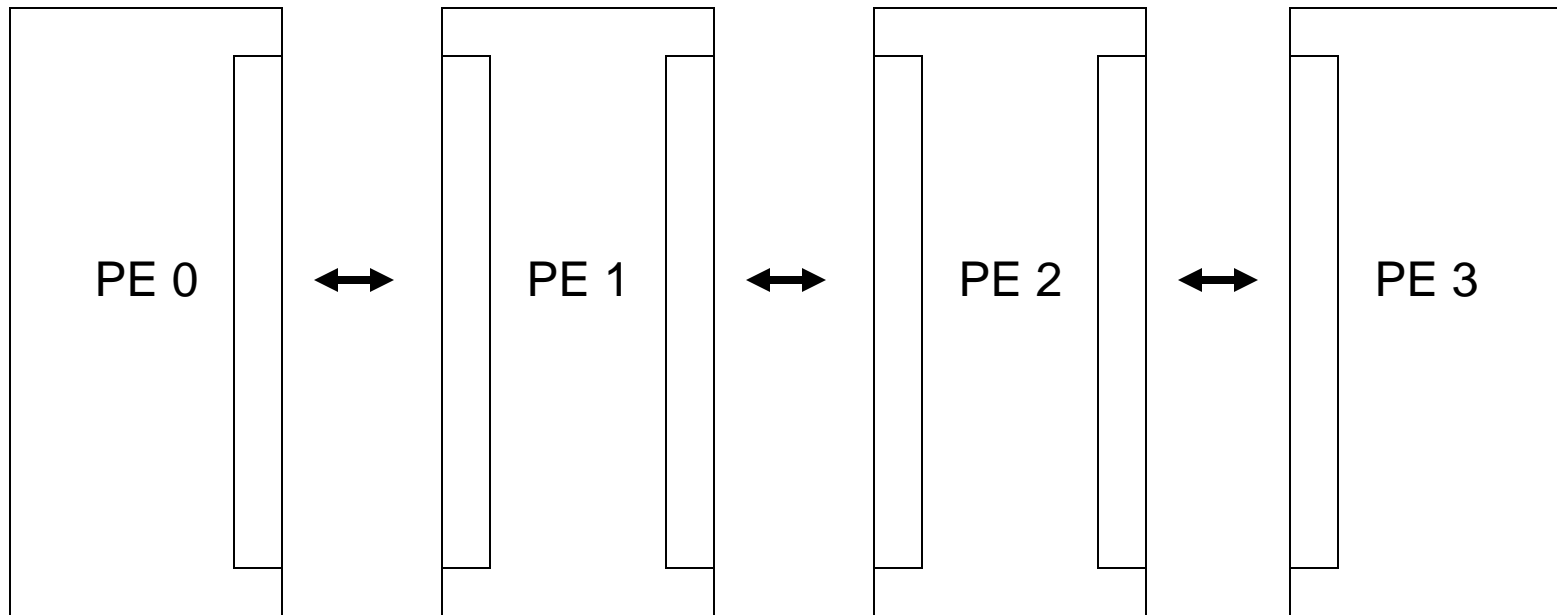
# Sample Parallelization Strategy

Break the grid into groups of columns for Fortran, rows for C.



The outer rectangles accommodate boundary and “ghost” zones.

# Exchange Edge Data at Each Iteration



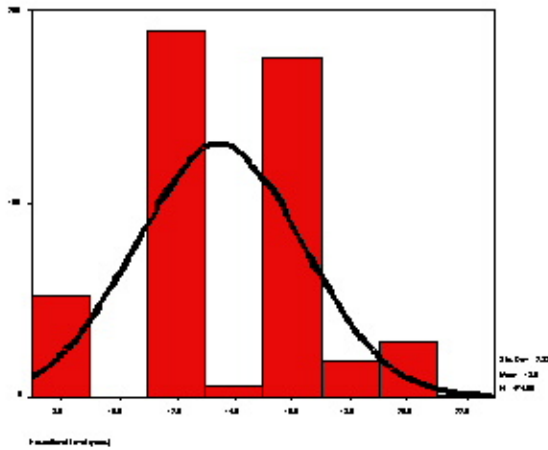
# General Advice for Parallelization

- Start with a **correct, legible** serial code. Rewrite if necessary.
- Determine whether the serial algorithm can be parallelized. Consider alternatives that may be more scalable.
- Test and debug on a small number of processes (2 to 4).

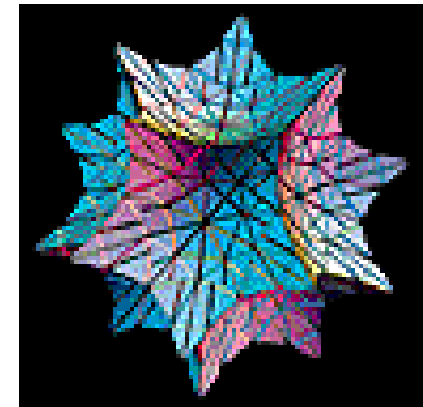
# Summary

- Simultaneous use of multiple compute resources.
- Parallelism can be coarse- or fine-grained.
- Saves wall-clock time, solves bigger problems
- Make sure serial program optimized before parallelizing it.

[www.llnl.gov/computing/tutorials/workshops/workshop/parallel\\_comp/](http://www.llnl.gov/computing/tutorials/workshops/workshop/parallel_comp/)



## Upcoming Talk



Implementing Parallel Codes  
Wednesday, February 15 at 3:30 PM  
Wilson 244

Tutorial is online at [www.itc.virginia.edu/research/linux-clusters/hands-on](http://www.itc.virginia.edu/research/linux-clusters/hands-on)

Talks are online at [www.itc.virginia.edu/research/talks](http://www.itc.virginia.edu/research/talks)