

# Scalability Performance

Lars Koesterke

IST, Lisbon, Portugal

May 25-26, 2009



THE UNIVERSITY OF TEXAS AT AUSTIN  
**Texas Advanced Computing Center**



# Overview

- What is scalability?
- Theoretical background
  - Amdahl's Law
  - Amdahl's Effect
  - Gustafson's Law
- Scaling: Strong vs. Weak
  - Examples
- Useful timers/monitors
  - Shell-level timing
  - MPI\_Wtime()
  - 3<sup>rd</sup>-party performance tools (PAPI, TAU)
- Analyzing code scalability

# What is scalability?

- How well a solution to some problem will work when the size of the problem increases.
  - Typically associated with algorithmic (time/space) complexity ( $O$ ,  $\Omega$ ,  $\Theta$ )
- How well a *parallel* solution to some problem will work when the *number of processing elements* (*PEs*) increases.
  - Strong scaling (speedup) or Weak scaling

# Theoretical Background

- Amdahl's Law
  - Theoretical performance of an application with a ***fixed amount of parallel work*** given a particular number of PEs
- Gustafson's Law
  - Theoretical performance of an application with a ***fixed amount of parallel work per PE*** given a particular number of PEs

# Amdahl's Law

- Effect of multiple PEs on run time of a problem with a *fixed amount of parallel work*

$$t_N = \left( f_s + f_p / N \right) t_s$$

- Speedup is the ratio of serial run time to parallel run time

$$S_N \leq \frac{t_s}{t_N} = \frac{1}{f_s + f_p / N}$$

# Amdahl's Effect

- What happens if you increase the size of the problem?
  - In many cases, increasing the problem size will increase the amount of parallel work, not serial
    - This will increase the fraction of parallel work (and decrease the fraction of serial work)
  - Result: Speedup appears to increase as the problem size increases

# Gustafson's Law

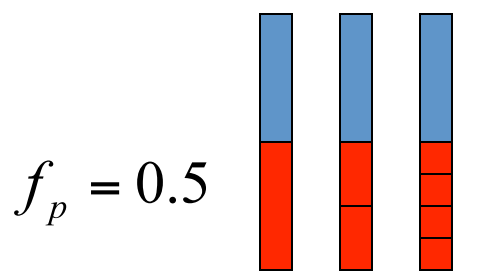
- Effect of multiple PEs on run time of a problem with *a fixed amount of parallel work per PE*

$$S_N \leq N + (1 - N)s$$

- $s$  is the fraction of parallel code where the parallel work per PE is fixed (not the same as  $f_p$  from Amdahl's)
- $N$  is the number of PEs

# Comparison of Amdahl and Gustafson

Amdahl – fixed work

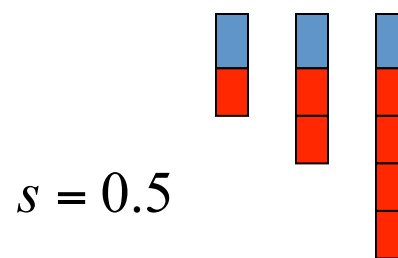


$$S \leq \frac{1}{f_s + f_p / N}$$

$$S_2 \leq \frac{1}{0.5 + 0.5 / 2} = 1.3$$

$$S_4 \leq \frac{1}{0.5 + 0.5 / 4} = 1.6$$

Gustafson – fixed work per PE



$$S \leq N + (1 - N) s$$

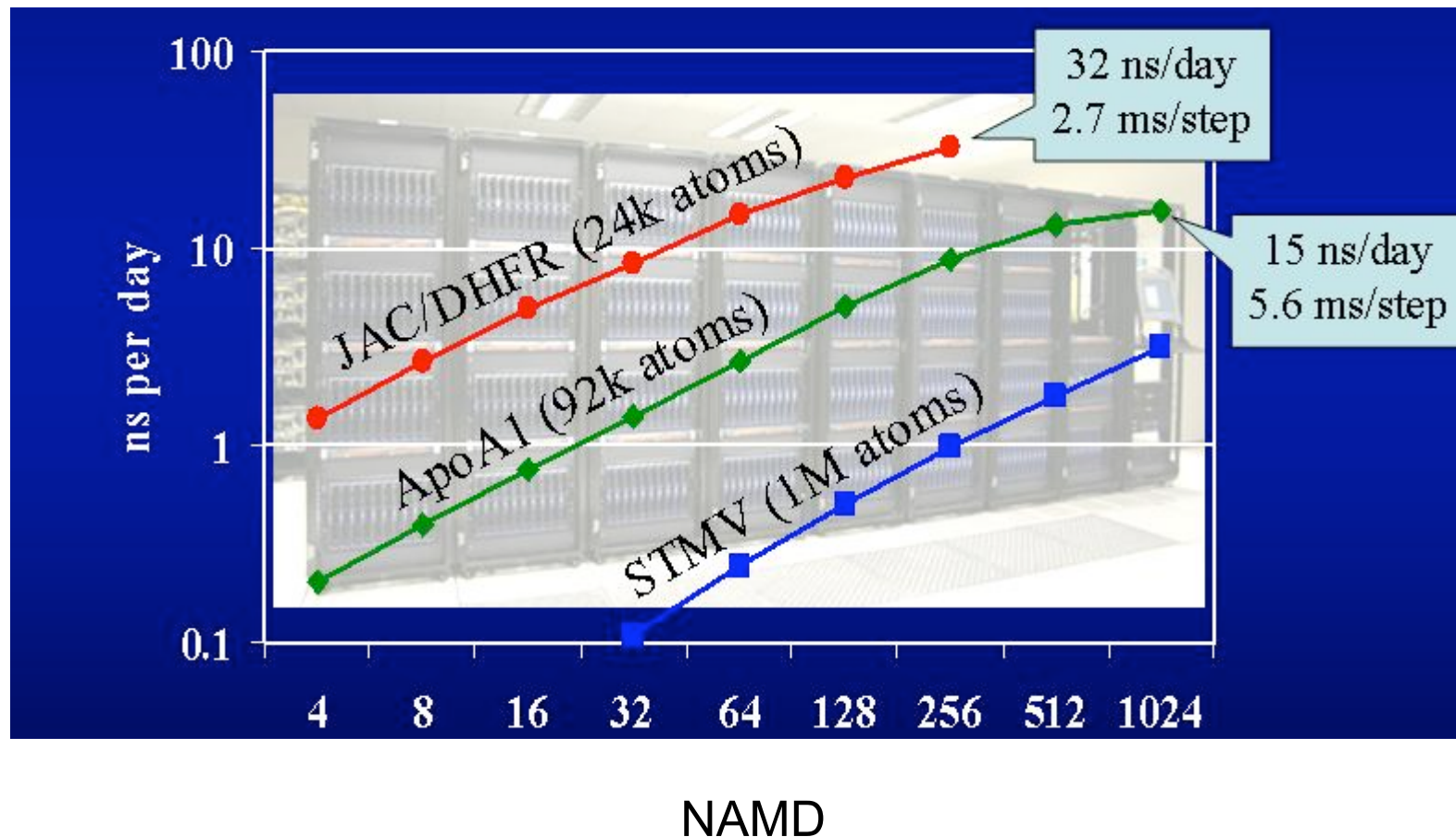
$$S_2 \leq 2 + (1 - 2) 0.5 = 1.5$$

$$S_4 \leq 4 + (1 - 4) 0.5 = 2.5$$

# Scaling: Strong vs. Weak

- We want to know how quickly we can complete analysis on a particular data set by increasing the PE count
  - Amdahl's Law
  - Known as “strong scaling”
- We want to know if we can analyze more data in approximately the same amount of time by increasing the PE count
  - Gustafson's Law
  - Known as “weak scaling”

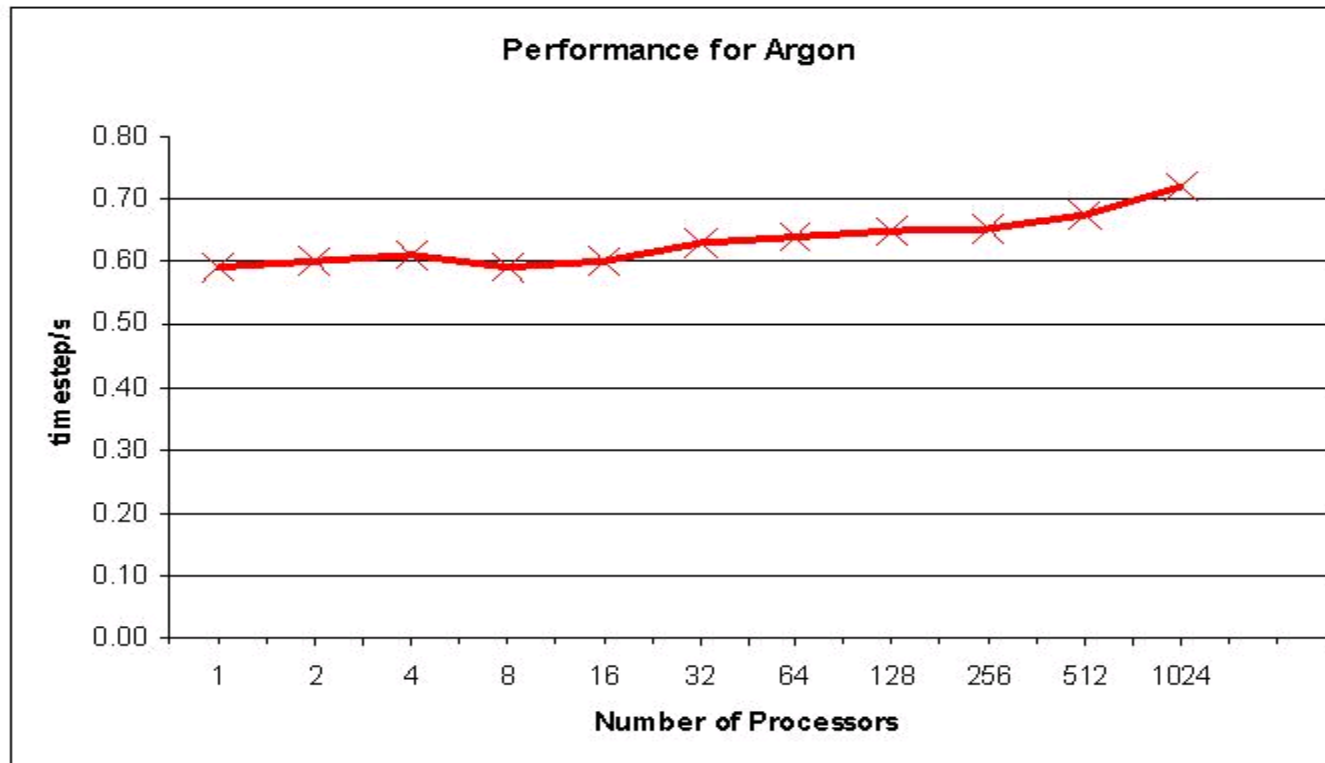
# Example of Strong Scaling



# Examples of Weak Scaling

- Berkeley Open Infrastructure for Network Computing (BOINC) (Fully distributed applications)
  - SETI@Home
  - No communication between independent machines/processes
  - Assuming all machines are equal, every process instance will require the same amount of time

# Examples of Weak Scaling - 2



DL\_POLY 3 (32,000 atoms per PE)

# Useful timers/monitors

- time/timex
  - Provides the amount of time consumed by an entire shell command
    - Extremely coarse
    - Includes startup/shutdown time
    - Usage on Linux: `/usr/bin/time -p <command>`
    - Usage on AIX: `/usr/bin/timex <command>`

# MPI\_Wtime()

- `int MPI_Wtime(void)`
  - Provides the number of seconds since an arbitrary point in the past
    - Because we cannot set the epoch for MPI\_Wtime, it must be used in pairs
    - Place one call to MPI\_Wtime() before the code segment to be timed, and another call after
    - Subtract before from after to get the time consumed in that particular code segment
    - Changeable attributes allow individual tasks or entire communicator groups to be timed

# 3<sup>rd</sup>-party performance tools

- PAPI – Performance Application Programming Interface (API)
  - Provides a common interface for performance monitors across varying platforms
- TAU – Tuning and Analysis Utilities
  - Provides both text-based and graphical code profiling at various levels of granularity (function, statement, etc.)
  - Specifically designed for parallel performance analysis

# Analyzing Code Scalability

- Although Amdahl and Gustafson provide theoretical upper bounds for codes, eventually real data are necessary for analysis
- Inconsistencies in performance – especially on shared systems – often appear in singular runs
- Best practice: Monitor codes several times and average the results to filter out periods of heavy I/O or network traffic due to other users

# Analyzing Code Scalability - 2

- Ideally, HPC codes would be able to scale to the theoretical limit
  - Never the case in reality
  - All codes eventually reach a real upper limit on speedup
  - At some point codes become “bound” to one or more limiting hardware factors
    - Memory, Network, Disk

# Conclusions

- Parallel applications have upper limits on performance
  - Amdahl's law provides a theoretical time limit for a fixed-size problem
    - Strong scaling
  - Gustafson's law provides a theoretical time limit for a problems with a fixed amount of data per PE
    - Weak scaling
- Timers and performance tools help developers determine the scalability of codes
- In reality codes are hindered not only by serial segments, but by the actions of other users or hardware limitations

# References

- <http://www.ks.uiuc.edu/Research/namd/>
- [http://www.cse.scitech.ac.uk/ccg/software/DL\\_POLY/](http://www.cse.scitech.ac.uk/ccg/software/DL_POLY/)