

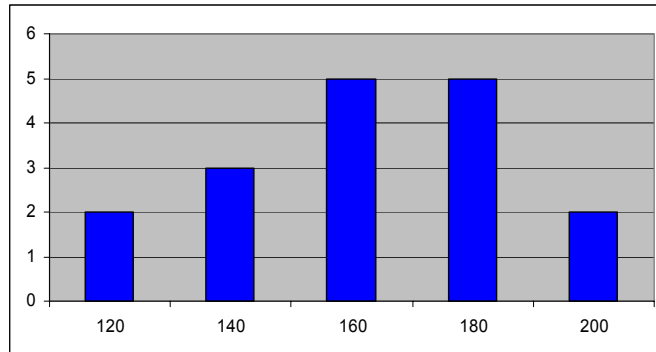
Understanding and Measuring Speedup

- Last Time
 - » Midterm Exam
- Today
 - » Midterm Summary
 - » Definitions of Speedup
 - » Measuring Speedup
- Reminders/Announcements
 - » New Homework #3 will be out soon (tomorrow?)
 - » Midterm Exams will be returned today (END of class)
 - » Graded Homework #2's will be returned next week...

Midterm: In Perspective

- In general, the class did very well.
- #2: Many didn't articulate the fundamental reasons that applications are increasingly parallel – large scale data, analysis complexity, large human activity
- Generally well on aspects of parallel Java programming: threads, synchronized, examples
- #9: Most didn't get the Pagerank question
 - » Two simple ways to calculate – random walk, equations for each node – iteration was the hard way

Midterm Scores



- Average and Median = 154
- Standard Deviation = 26
- High Score = 198

Why Measure Performance?

- Tells you how you are doing
- Limits tell you whether things can be improved appreciably
- Important: Understand exactly what you are measuring and how you are measuring it.

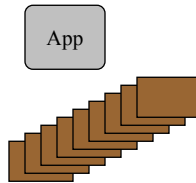
Common Resource Performance Measures

- MFLOPS = million floating point operations per seconds
 - » GFLOPS, TFLOPS ...
- MBYTES = million bytes per second
 - » GBytes, TBytes ...
- MIPS = Million instructions per second
- These metrics provide one measure of resource performance. They do not however indicate how fast YOUR program will run.

Performance Improvement

- Relative Performance – a la CSE 141
- What is being compared?
 - » Machine A vs. Machine B
 - » Program A vs. Program B
- System A is X times faster than System B

Comparing Performance for Parallel Programs



- Parallel Program vs. Sequential Program
- Same Machine? Try to keep the processors equal (1 of them vs. N of them)
- These comparisons are known as “speedup”.

Speedup

- Speedup = $S(n) = \frac{\text{(Execution time on Single CPU)}}{\text{(Execution on N parallel processors)}}$

$$= \frac{T_s}{T_p}$$

- » Speedup measures of application performance on
 - a given application implementation and
 - platform (system software and hardware)

Preview: What is a Good Speedup?

- Hopefully, $S(n) > 1$
- Linear speedup:
 - » $S(n) = n$
 - » Parallel program considered perfectly scalable
- Superlinear speedup:
 - » $S(n) > n$
 - » Can this happen?

Defining Speed-Up

- Speedup = $S(n) = \frac{\text{(Execution time on Single CPU)}}{\text{(Execution on } n \text{ parallel processors)}}$
- Speedup depends on many attributes:
 - » What problem size?
 - » Worst case? Average case? Best case?
 - » What do we count as work?
 - Parallel computation, communication, overhead?
 - » What sequential algorithm and what machine for the numerator?
 - Can the algorithms used for the numerator and the denominator be different?

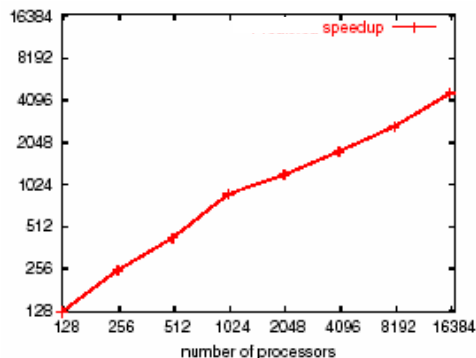
Common Definitions of Speedup

- Speedup = $S(n) = \frac{\text{(Execution time on Single CPU)}}{\text{(Execution on n parallel processors)}}$
- Let M be a parallel machine with p processors
- Let $T(X)$ be the time it takes to solve a problem on M with X processors
- Common definitions of Speedup:
 - » Serial machine is one processor of parallel machine and serial algorithm is interleaved version of parallel algorithm $S(n) = \frac{T(1)}{T(n)}$
 - » Serial algorithm is fastest known serial algorithm for running on a serial processor (W+A) $S(n) = \frac{T_s}{T(n)}$
 - » Serial algorithm is fastest known serial algorithm running on a one processor of the parallel machine (Gustafson) $S(n) = \frac{T'(1)}{T(n)}$

CSE 160 Chien, Spring 2005

Lecture #10, Slide 11

Typical Speedup Graph



- X-axis is the number of processors; Y-axis is the speedup
- Graph is for a particular program
- Ideal is a straight line, with unit slope (that is, 1)

CSE 160 Chien, Spring 2005

Lecture #10, Slide 12

Can speedup be superlinear?

- Speedup CAN be superlinear:
 - » Let M be a parallel machine with n processors
 - » Let T(X) be the time it takes to solve a problem on M with X processors
 - » Speedup definition: $S(n) = \frac{T_s}{T(n)}$
 - » Serial version of the algorithm may involve more overhead than the parallel version of the algorithm
 - E.g. A=B+C on a SIMD machine with A,B,C matrices vs. loop overhead on a serial machine
 - » Hardware characteristics may favor parallel algorithm
 - E.g. if all data can be decomposed in caches or main memories of parallel processors vs. needing secondary storage on serial processor to retain all data

Bounds on Speedup (Amdahl)

- What is the maximum speedup possible for a parallel program?
 - » Let f = serial fraction that cannot be parallelized
- Amdahl's law – bounds the speedup in terms of serial portion and parallelizable portion of algorithm.

$$T_s = fT_s + (1 - f)T_s$$

$$T_p = fT_s + \frac{(1 - f)T_s}{n}$$

$$S(n) = \frac{T_s}{fT_s + \frac{(1 - f)T_s}{n}} = \frac{n}{nf + 1 - f} = \frac{1}{\frac{(n - 1)f + 1}{n}}$$

$$\lim_{n \rightarrow \infty} = \frac{1}{f}$$

Example of Amdahl's Law

- Suppose that a calculation has a 4% serial portion, what is the limit of speedup on 64 processors?
- What is the maximum speedup?

Speedup Variants: Parallel Efficiency

- Efficiency: $E(n) = S(n)/n * 100\%$
- Efficiency measures the fraction of ideal speedup that is being achieved
 - » A program with linear speedup is 100% efficient.
- Using efficiency:
 - » A program attains 89% parallel efficiency on 64 processors, what is the speedup?

Pitfalls: Cheating Speedup

- Not using the best sequential algorithm or running time – makes you look good
 - » Using the parallel version (lots of overhead built-in)
 - » Using an algorithm which doesn't make optimal use of the caches

Beyond Amdahl's Law

- Gustafson challenged Amdahl's assumption that serial fraction (f) remains constant for all problem sizes (and for larger machines \rightarrow larger problems)
 - » Example: if serial part grows as N and the parallel part grows as N^2 , then as problem size grows, the serial fraction (f) decreases
 - » $N = 100$, $N^2=10,000$, $f = 100/10,100 = 1\%$
 - » $N = 1000$, $N^2=1,000,000$, $f = 0.1\%$
 - » $N = 10,000$, $N^2=100,000,000$, $f = 0.01\%$
- According to Amdahl what speedups would be possible?

Gustafson's Speed Limits

- Gustafson defined two “more relevant” notions of speedup
 - » Scaled speedup
 - » Fixed-time speedup
 - » And renamed Amdahl's version as “fixed-size” speedup

Gustafson's Law

- Fix execution time on a single processor
 - » $s + p = \text{serial part} + \text{parallel part} = 1$ (normalized serial time)
 - » ($s =$ same as f previously)
 - » Assume problem fits in memory of serial computer
- Fixed-size speedup (Amdahl's Law)
- Fix execution time on a parallel computer
 - » $s + p = \text{serial part} + \text{parallel part} = 1$ (normalized parallel time)
 - » $s + np = \text{serial time on a single processor}$
 - » Assume problem fits in memory of parallel computer
- Scaled Speedup (Gustafson's Law)

$$\begin{aligned} S_{\text{fixed_size}} &= \frac{s + p}{s + \frac{p}{n}} \\ &= \frac{1}{s + \frac{1-s}{n}} \end{aligned}$$

$$\begin{aligned} S_{\text{scaled}} &= \frac{s + np}{s + p} \\ &= n + (1 - n)s \end{aligned}$$

Scaled Speedup

- Scaling: problem size can increase with number of processors
 - » Memory, Compute Power Increase, so does problem ambition! (at some point problems may not be meaningful)
 - » Gustafson's law gives measure of how much
- Scaled Speedup fixes the parallel execution time
 - » Amdahl fixed the problem size → fixes serial execution time
 - » Too conservative for large-scale systems
- Interesting consequence: no bound to speedup as $n \rightarrow$ infinity, speedup has no real bound...

Using Gustafson's Law

- Given a scaled speedup of 80 on 128 processors, what is the serial fraction from Amdahl's law? What is the serial fraction from Gustafson's Law?

$$\begin{aligned} S_{scaled} &= \frac{s + np}{s + p} \\ &= n + (1 - n)s \end{aligned}$$

Fixed Time Speedup

- Gustafson also!
 - » Use scaled speedup when the memory requirements scale linearly with the number of processors
- Idea: Use fixed-time speedup when the work scales linearly with the number of processors, rather than the memory
 - » A different kind of scaleup – allows problem size to increase (and perhaps also serial fraction to decrease)

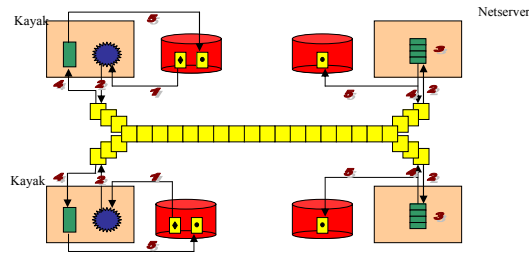
Fixed Time Speedup

- Let
 - $T_p'(1, X)$ = complexity of the best serial algorithm for a size X problem on one processor of the parallel machine.
 - $T_p(m, X)$ = complexity of the parallel algorithm run on m processors for problem size X
 - N_0 = the size of the largest problem that conveniently fits into primary memory of one processor
 - N_m = maximum value of N satisfying $T_p(m, N) \leq T_p'(1, N_0)$
may be non-monotonic due to architectural features
 - mN_0 = size of the problem that conveniently fits into primary memory of a parallel machine with m processors

$$S_{scaled} = \frac{T_p'(1, mN_0)}{T_p(m, mN_0)}$$

$$S_{fixed_time} = \frac{T_p'(1, N_m)}{T_p(m, N_m)} \approx \frac{T_p'(1, N_m)}{T_p'(1, N_0)}$$

Example: MinuteSort



- Minute Sort (all the records you can sort in a Minute!)
 - » Fixed Time Scaling
 - » 340Million, 32GB, 2004
 - » ~120M, 12GB, 2000
- See Gray's Sort Benchmark Page
<http://research.microsoft.com/barc/SortBenchmark/>

Fixed Work Benchmark

- Work (and data) scales up with # of processors
- Measure time to complete an iteration --- it goes up with # of Nodes!
- Similar to Fixed Time Model

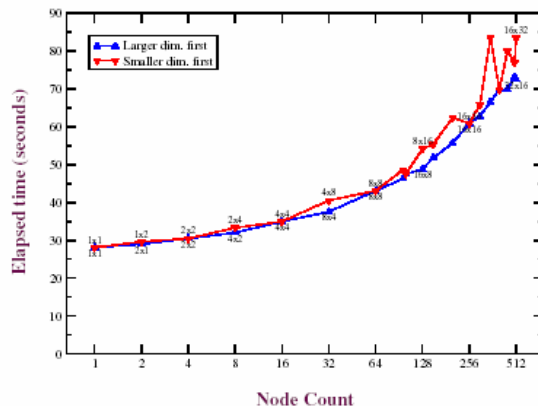


Figure 14. Sweep3D, 50x50x50 cells,MMI=3, MK=10 blocking.

Summary

- Midterm Redux
- Speedup
 - » Amdahl's Law and Gustafson's Revision
 - » Speedup vs. Absolute Efficiency
- Next Time
 - » Benchmarks
 - » Some Application and Machine Examples