Measuring Performance

Speed-up, Amdahl’s Law,
Gustafson’s Law, efficiency, benchmarks
Why Measure Performance?

- Performance tells you how you are doing and whether things can be improved appreciably
- When measuring performance, it is important to understand exactly what you are measuring and how you are measuring it.
Two notions of “performance”

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td>286,700</td>
</tr>
<tr>
<td>BAD/Sud Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

Which has higher performance?

° Time to do the task (Execution Time)
  – execution time, response time, latency

° Tasks per day, hour, week, sec, ns. .. (Performance)
  – performance, throughput, bandwidth

Response time and throughput often are in opposition…
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.
Poor Performance Metrics

• Marketing metrics for computer performance included MIPS and MFLOPS
  • MIPS : millions of instructions per second
    – Advantage : Easy to understand and measure
    – Disadvantages : May not reflect actual performance, since simple instructions do better.
  • MFLOPS : millions of floating point operations per second
    – Advantage : Easy to understand and measure
    – Disadvantages : Same as MIPS, only measures floating point
Speed-Up

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

Speedup provides a measure of application performance with respect to a given program platform.
What is “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** = \( S(n) = \frac{\text{Execution time on Single CPU}}{\text{Execution on n parallel processors}} \)

- We need more information to evaluate speedup:
  - What problem size? Worst case time? Average case time?
  - What do we count as work?
    - Parallel computation, communication, overhead?
  - What serial algorithm and what machine should we use for the numerator?
    - Can the algorithms used for the numerator and the denominator be different?
Common Definitions of Speed-Up

- **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 parallel algorithm with one processor
  $$S(n) = \frac{T(1)}{T(n)}$$
- Serial algorithm is fastest known serial algorithm for running on a serial processor
  $$S(n) = \frac{T_s}{T(n)}$$
- Serial algorithm is fastest known serial algorithm running on a one processor of the parallel machine (Gustafson)
  $$S(n) = \frac{T'(1)}{T(n)}$$
Can speedup be superlinear?

- **Zen answer:** both **YES** and **NO**, depending on the definition
- **Speedup CANNOT 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: \( S(n) = \frac{T(1)}{T(n)} \)
  - Suppose a parallel algorithm A solves an instance I of a problem in \( t \) time units
  - Then A can solve the same problem in \( nt \) units of time on M through time slicing
  - The best serial time for I will be no bigger than \( nt \)
  - Hence speedup cannot be greater than \( n \).

\[
S(n) = \frac{T(1)}{T(n)} \leq \frac{nt}{t} = n
\]
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 main memories of parallel processors vs. needing secondary storage on serial processor to retain all data
  - “work” may be counted differently in serial and parallel algorithms
Bounds on Speedup

• What is the maximum speedup possible for a parallel program?
• \( f = \) fraction of program (algorithm) that is serial and \textit{cannot be parallelized}
  – Data setup
  – Reading/writing to a single disk file

• \textbf{Amdahl’s law} provides limit on speedup in terms of serial portion and parallelizable portion of algorithm.

\[
T_s = fT_s + (1 - f)T_s
\]

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

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

\[
\lim_{n \to \infty} = \frac{1}{f}
\]
Example of Amdahl’s Law

• Suppose that a calculation has a 4% serial portion, what is the limit of speedup on 16 processors?
  – $16/(1 + (16 - 1) \times 0.04) = 10$
  – What is the maximum speedup?
  • $1/0.04 = 25$
Variants of Speedup: Efficiency

- Efficiency: \( E(n) = \frac{S(n)}{n} \times 100\% \)
- Efficiency measures the fraction of time that processors are being used on the computation.
  - A program with linear speedup is 100% efficient.
- Using efficiency:
  - A program attains 89% efficiency with a serial fraction of 2%. Approximately how many processors are being used according to Amdahl’s law?
Limitations of Speedup

• Conventional notions of speedup don't always provide a reasonable measure of performance

• Questionable assumptions:
  – "work" in conventional definitions of speedup is defined by operation count
    • communication more expensive than computation on current high-performance computers
  – best serial algorithm defines the least work necessary
    • for some languages on some machines, serial algorithm may do more work -- (loop operations vs. data parallel for example)
  – good performance for many users involves fast time on a sufficiently large problem; faster time on a smaller problem (better speedup) is less interesting
  – traditional speedup measures assume a "flat memory approximation", i.e. all memory accesses take the same amount of time
“Flat Memory Approximation”

• “Flat memory Approximation” – all accesses to memory take the same amount of time
  – in practice, accesses to information in cache, main memory and peripheral memory take very different amounts of time.
Limitations of Speedup

• Gustafson challenged Amdahl's assumption that the proportion of a program given to serial computations \((f)\) and the proportion of a program given to parallel computations remains the same over all problem sizes.
  – For example, if the serial part is a loop initialization and the can be executed in parallel over the size of the input list, then the serial initialization becomes a smaller proportion of the overall calculation as the problem size grows larger.

• Gustafson defined two “more relevant” notions of speedup
  – Scaled speedup
  – Fixed-time speedup
  – (usual version he called fixed-size speedup)
Gustafson’s Law

- Fix execution time on a **single processor**
  - \( s + p = \text{serial part} + \text{parallelizable part} \)
  - \( s = \frac{1}{s} \) (normalized serial time)
  - (s = same as \( f \) previously)
  - Assume problem fits in memory of serial computer
  - **Fixed-size speedup**

\[
S_{fixed\_size} = \frac{s + p}{s + \frac{p}{n}} = \frac{1}{1 - s}
\]

- Fix execution time on a **parallel computer (multiple processors)**
  - \( s + p = \text{serial part} + \text{parallelizable part} \)
  - \( s + np = \text{serial time on a single processor} \)
  - Assume problem fits in memory of parallel computer
  - **Scaled Speedup**

\[
S_{scaled} = \frac{s + np}{s + p} = n + (1 - n)s
\]
Scaled Speedup

• Scaling implies that problem size can increase with number of processors
  – Gustafson’s law gives measure of how much

• Scaled Speedup derived by fixing the parallel execution time (Amdahl fixed the problem size → fixes serial execution time)
  – Amdahl’s law may be too conservative for high-performance computing.

• Interesting consequence of scaled speedup: no bound to speedup as n→ infinity, speedup can easily become superlinear!

• In practice, unbounded scalability is unrealistic as quality of answer will reach a point where no further increase in problem size may be justified.
Using Gustafson’s Law

• Given a scaled speedup of 20 on 32 processors, what is the serial fraction from Amdahl’s law? What is the serial fraction from Gustafson’s Law?

\[
S_{scaled} = \frac{s + np}{s + p} = n + (1 - n)s
\]
Fixed Time Speedup

- Also due to Gustafson
- In original paper, (http://www.scl.ameslab.gov/Publications/FixedTime/FixedTime.html) Gustafson uses scaled speedup when the memory requirements scale linearly with the number of processors
- Gustafson uses fixed-time speedup when the work scales linearly with the number of processors, rather than the storage
- Both measures allow the problem size to scale whereas fixed-size speedup (conventional speedup measure) assumes that the problem size is fixed.
Fixed Time Speedup

Let

\[ T_p'(1, X) = \text{complexity of the best serial algorithm for a size X problem on one processor of the parallel machine.} \]

- Curve-fitting of experimental data can be used to define the algebraic coefficients for a functional expression for \( C(N) \)

\[ T_p(m, X) = \text{complexity of the parallel algorithm run on m processors for problem size X} \]

\( N_0 \) = 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

\[ S_{scaled} = \frac{T_p'(1, mN_0)}{T_P(m, mN_0)} \]

\[ S_{fixed\_time} = \frac{T_p'(1, N_m)}{T_P(m, N_m)} \approx \frac{T_p'(1, N_m)}{T_P'(1, N_0)} \]
Using programs to Measure Machine Performance

• Speedup measures performance of an individual program on a particular machine
  – Speedup cannot be used to
    • Compare different algorithms on the same computer
    • Compare the same algorithm on different computers

• Benchmarks are representative programs which can be used to compare performance of machines
Benchmarks used for Parallel Machines

- The Perfect Club
- The Livermore Loops
- The NAS Parallel Benchmarks
- The SPEC Benchmarks
- The “PACKS” (Linpack, LAPACK, ScaLAPACK, etc.)
- ParkBENCH
- SLALOM, HINT
The Perfect Club

- Developed at University of Illinois around 1987
- Set of real applications donated by interested parties organized into a standardized set of benchmarks
  - Originally 13 codes, ~1000 lines of Fortran
  - Full-scale scientific apps rather than kernels or compact apps
  - Floating point-intensive codes usually executed on (vector) supercomputers
- Applications characterized in terms of their algorithmic behavior, allowing users to get meaningful predictions of the performance they could expect for their own applications
- Some codes incorporated into the SPEC benchmarks
The Livermore Loops

• Set of 24 Fortran DO loops extracted from operational code at LLNL
  – Originated the use of MFLOP/s for performance
  – Performance statistics reported: arithmetic, harmonic, geometric means, …

• http://www.netlib.org/benchmark/livermore
NAS Parallel Benchmarks (NPB)

- Benchmarks from CFD (computational fluid dynamics) codes
  - Fortran and C versions available.
  - NPB are kernels and compact pseudo-applications, not full applications
- Algorithmic definition of each program and sequential implementation of each algorithm
- Application can be supported or implemented efficiently in a machine–dependent way
- Users write a set of tuned parallel applications
- Suite gives manufacturers a chance to demonstrate what their machines can do
- NPB are widely used
SPEC Benchmarks

• **SPEC = Standard Performance Evaluation Corporation**
  - non-profit, industry-sponsored organization
    - Goal is to ensure that the marketplace has a fair and useful metric to differentiate candidate systems
  - SPEC benchmarks are standardized suite of source codes based on existing applications that have already been ported to a variety of architectures
    - E.g. SPECseis is sample code used to locate oil and gas deposits from echo data, SPECchem is GAMESS for drug design and bonding analysis
    - Serial and parallel versions, MP, SM being developed
  - Benchmarker takes the source code, compiles it for target system and can tune system for best results.
  - Focus groups (open system high performance, graphics performance, etc.) develop benchmarks and make them available via a website.

• **SPEC benchmark suites include**
  - CINT92 (CPU intensive integer benchmarks)
  - CFP92 (CPU intensive floating point benchmarks)
  - SDM (UNIX Software Development Workloads)
  - SFS (System level file server (NFS) workload)

LinPack

- Linear Algebra routines available in both C and Fortran
- Benchmarks solve a system of equations using Gaussian Elimination
  - MFLOPS reported
- Core of Linpack is subroutine ("saxpy" in the single-precision version, "daxpy" in the double-precision version) doing the inner loop for frequent matrix operations: $y(i) = y(i) + a \times x(i)$
- Standard version operates on 100x100 matrices; there are also versions for sizes 300x300 and 1000x1000, with different optimization rules.
- Optimizations:
  - Linpack is easily vectorizable on many systems.
  - Easy to exploit a multiply-add operation
  - Some compilers have "daxpy recognizers" to substitute hand-optimized code!
- Originator: Jack Dongarra, Univ. of Tennessee,
- netlib@ornl.gov: source, results
- netlib.att.com:/netlib/benchmark/linpack*: source
Dongarra’s Top 500 List

- Overview of the market of high-performance systems
- Twice a year list of the 500 most powerful installed systems
- Linpack benchmark used to rank systems
- Annual report analyzes developments in HW, SW and the market
- http://www.netlib.org/benchmark/top500.html
Other “PACKS”

- **LAPACK**
  - Subroutine library for solving the most common problems in numerical linear algebra
  - Designed to run efficiently on shared memory vector and parallel processors

- **ScaLAPACK**
  - Software library for performing dense and band linear algebra computations on a distributed memory MP MIMD computers and networks of workstations supporting PVM and/or MPI

- **PARKBENCH**
  - Stands for Parallel Kernels and Benchmarks
  - Suite contains sequential codes, communication codes, linear algebra kernels, NPB kernels, NASA compact application codes, parallel spectral transform shallow water model code
  - [http://www.netlib.org/parkbench/](http://www.netlib.org/parkbench/)
SLALOM

- Developed by Gustafson, Rover, Elbert, Carter
- Benchmark computes equilibrium radiation transfer on a closed interior
- Scales in the number of finite elements into which the surface is decomposed
- Benchmark scales automatically to run in one minute on any computer
  - Versions in several languages and for several architecture types (vector, serial, SM, MP, SIMD, MIMD)
  - Memory requirements adjust automatically
- Can be used to compare machines as disparate as laptops and supercomputers
- http://www.scl.ameslab.gov/Publications/FixedTime/FixedTime.html#6.3
HINT

- Developed by Gustafson and Snell
- **HINT** = Hierarchical Integration benchmark
- Work measure is **QUIPS** – quality improvement per second – to measure the amount of work a computer can perform over time
- **HINT** based on successive refinement-style optimization of an integration curve
  - QUIPS computed as a step functions of time whenever an improvement to answer quality is computed
  - Task adjusts to precision available and has unlimited (algorithmic) scalability
- **HINT** a successor of SLALOM
  - Provides a more precise way of measuring answer quality
  - Reduces the complexity of SLALOM
  - Less difficult to optimize than SLALOM
  - Addresses technical problems of SLALOM wrt memory usage
Limitations and Pitfalls of Benchmarks

- Benchmarks cannot address questions you did not ask
- Specific application benchmarks will not tell you about the performance of other applications without proper analysis
- General benchmarks will not tell you all the details about the performance of your specific application
- One should understand the benchmark itself to understand what it tells us
Benefits of Benchmarks

- Popular benchmarks keep vendors attuned to applications
- Benchmarks can give useful information about the performance of systems on particular kinds of programs
- Benchmarks help in exposing performance bottlenecks of systems at the technical and applications level