21st Summer School of PARALLEL COMPUTING

July 2 - 13, 2012 (Italian)
September 3 - 14, 2012 (English)

Performance Evaluation

Giovanni Erbacci - g.erbacci@cineca.it
Supercomputing, Applications & Innovation Department - CINECA

CINECA

Outline

Performance Evaluation

Definition of parallel system
- size of the problem,
- serial execution time
- parallel execution time

Performance indices
- speed-up
- efficiency

Art of Performance evaluation

→Performance evaluation of HPC Systems
- Benchmarks
- Performance models

→Performance evaluation of Aplication Codes

<u>What metrics to adopt</u>

Achieving Efficiency

- Parallel computers allow
  - faster solutions to problems
  - larger problems to be addressed

- An efficient parallel program
  - maximises the amount of work each processor does, and
  - minimises the amount of communication between processes

- How the problem is *decomposed is critical*
  - different ways exist depending on problem

Performance Measures

Scientist:

- size of the problem,
- accuracy of the solution, etc. ...
- Number of operations per unit time (flop / s),

Computational scientist:

- execution time,
- speed-up,
- efficiency.

Quantifying Performance

- Serial computing concerned with complexity
  - how execution time varies with problem size *N*
  - adding two arrays (or *vectors) is O(N)*
  - matrix times vector is *O(N2), matrix-matrix isO(N3)*

- Look for clever algorithms
  - naïve sort is *O(N2)*
  - divide-and-conquer approaches are*O(N log (N))*

- Parallel computing *also concerned with scaling*
  - how time varies with number of processors *P*
  - different algorithms can have different scaling behaviour
  - but always remember that we are interested in minimum time!

Parallel System

A parallel system is the implementation of a parallel algorithm on a specific parallel architecture.

"How to scale the parallel system?"

The size of the problem - **W** - is the number of basic operations required by the fastest known sequential algorithm to solve the problem itself, which is equivalent to the concept of computational complexity.

Observations

Sequential algorithm
- computational complexity
- execution time

f (W) (function of the amount of data supplied as input)

Parallel algorithm
- computational complexity
- execution time

f (W, p, arch)

p = number of processors used

arch = architecture on which the algorithm is implemented, (topology of the interconnection network)

Execution Time

Serial execution time - ***Ts*** - is the time that elapses between the beginning and the end of the program on a single processor.

Parallel execution time - ***Tp*** - is the time that elapses between the beginning of the parallel execution and the time when the last processor finishes his execution.

I/O Problem

Speed-up

$$\text{Speedup} = \frac{\text{Sequential execution time of the best sequential algorithm known}}{\text{Execution time on P processors}}$$

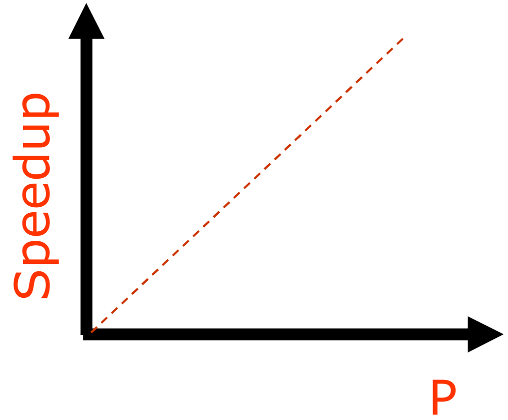
- A more honest measure of performance
- Avoids picking an easily parallelizable algorithm with poor sequential execution time

Speed-up

speed-up is defined as:

$$S(W,p) = T_s(W) \ / \ T_p(W,p)$$

- What does it represent?
- In which interval varies?



21st Summer School of PARALLEL COMPUTING

What Speedup you get?

Linear Speed-up :

- Speed-up = N, with N processors

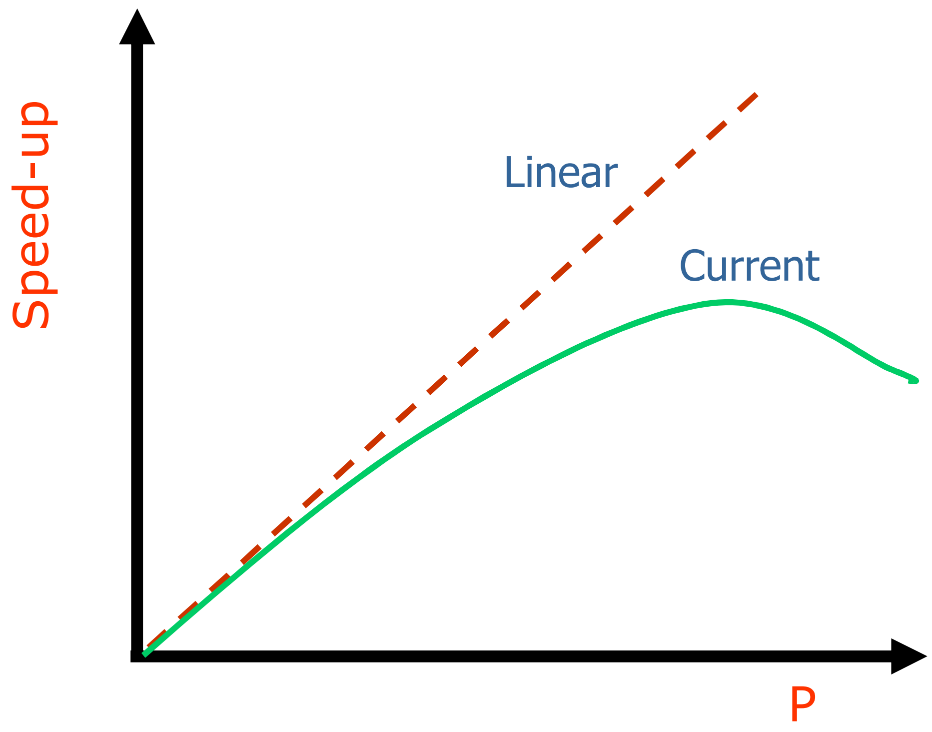
Sub-linear Speed-up :

- More normal, due to the overhead of initialization, synchronization, communication, etc..

Speed-up in flexion:

- Decreases as the number of processorsgrows.

super-linear Speed-up



Super Linear Speed-up

The speed-up is superlinear. when **s> p**,

This behavior is due to the fact that the program uses the memory hierarchies (due HW),

- Subparts fit into cache / memory of each node
- Whole problem does not fit in cache / memory of a single node

or there is a better code optimization with regard to the scheduling of instructions (due SW) or not detrminism eg. in search problems. One thread finds near-optimal solution very quickly => leads to drastic pruning of search space

Efficiency

Efficiency is defined as

$$E\,(W, p) = S\,(W, p) \,/\, p$$

is the ratio between the sequential execution time and the execution time on p processors, multiplied by p

What amount is?
What does it represent?
In whichi nterval varies?

Scalability of Parallel computers

There is no precise definition of **scalability**

An Architecture is scalable if it continues to have the same performance per processor when the number of processors and the size of the computational problem to be solved increse.

Scalable MPP systems are designed in such a way that larger versions of the same machine (systems with a greater number of nodes) can be constructed or extended from the same design.

A program scales for a number of processors P, if moving from p-1 to p processors an improvement in terms of speed-up is observed .

Improving load balance / algorithm increases the turn-over to a higher numbers of PEs

better scaling = ability to utilise larger computers

Sources of overhead

The efficiency of real parallel systems often (unless trivial parallel algorithms, embarrassingly parallel) is not maximum because in a parallel system appears sources of overhead such as:

- extra computations for the parallel algorithm compared to the best sequential algorithm,

- need for interprocessor communications,

- workload imbalance and more.

Granularity

How long does it take to communicate? Relevant network metrics:

- Bandwidth: number of bits per second that can be transmitted through the network
- Latency: time to make a message transfer through the network

Message-passing parallel programs can minimize communication delays by partitioning the program into processes and considering the granularity of the process on the machine.

$$granularity = \frac{t_{computation}}{t_{communication}}$$

21st Summer School of PARALLEL COMPUTING

Serial and parallel fraction

The **serial fraction** of a program, f_s , is the ratio between the time spent in the code inherently sequential and $T_s(W)$.

We define **parallel fraction** of a program, f_p, the ratio between the time spent in the code parallelizable and $T_s(W)$.

Obviously $f_s = (1 - f_p)$.

Sequential vs. Parallel

Sequential execution time: t seconds

Start-up overhead of parallel execution: t_st seconds (depends on architecture)

Parallel execution time (ideal): t/p + t_st

If t/p + t_st > t, no gain!

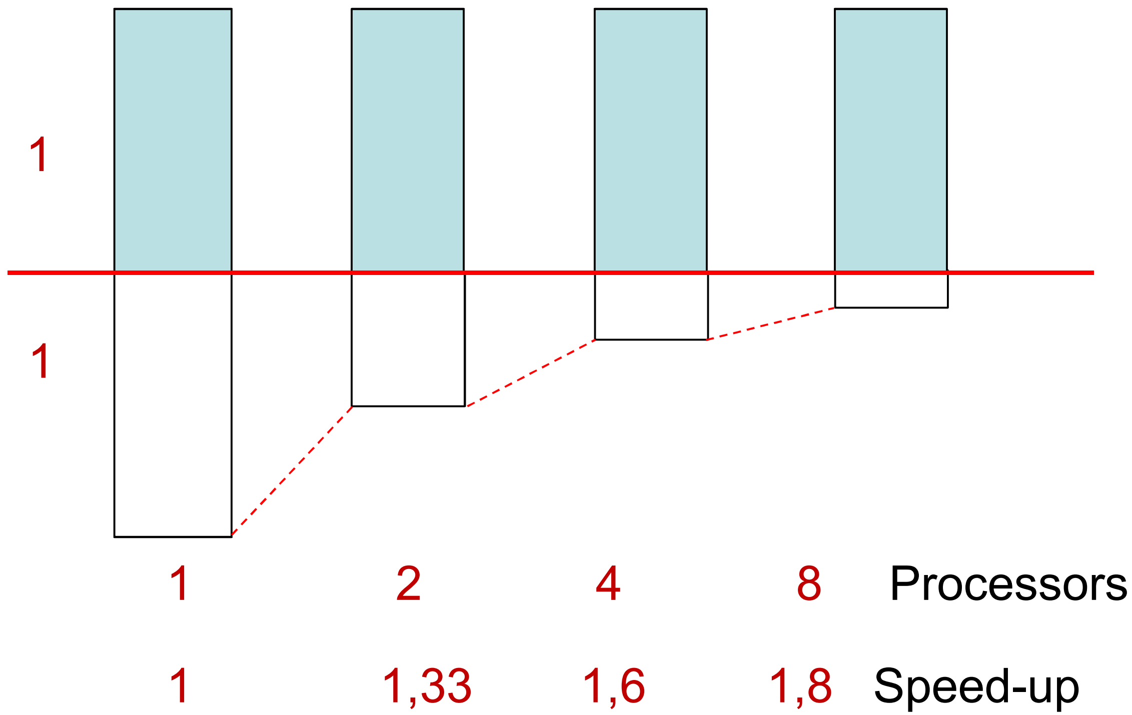
21st Summer School of PARALLEL COMPUTING

The Serial Component

- Amdahl's law

  "the performance improvement to be gained by parallelisation is limited by the proportion of the code which is serial"

  Gene Amdahl, 1967



Amdahl's Law

- Assume a fraction f_s is completely serial
  - time is sum of serial and potentially parallel
- Parallel time
  - parallel part 100% efficient

$$T_p(W,p)=T_s(W)\, f_s + (T_s(W)\; f_p / p)$$

- Parallel speedup $S(W,p) = T_s(W) \;/\; T_p(W,p) =$

$$= T_s(W) /(f_s \times T_s(W) + (1- f_s) \times T_s(W) /p)$$

$$= p \;/\; (1 +(p-1) \times f_s)$$

$$S(W,p) \rightarrow 1 / f_s \;\; per\; p \rightarrow \infty$$

  - for $f_s = 0$, *S= P as expected (ie E= 100%)*
  - otherwise, speedup limited by 1/ f_s for any *P*
  - Eg. if 5% of the code is sequential, the speed-up will never exceed 20 even with an infinite number of processors.
  - *the performance improvement to be gained by parallelisation is limited by the proportion of the code which is serial*
  - impossible to effectively utilise large parallel machines?

Amdahl Law confutation

Sometimes even a limited speed-up can be a very important milestone for certain applications.

In addition, applications can scale as the number of processors increases:

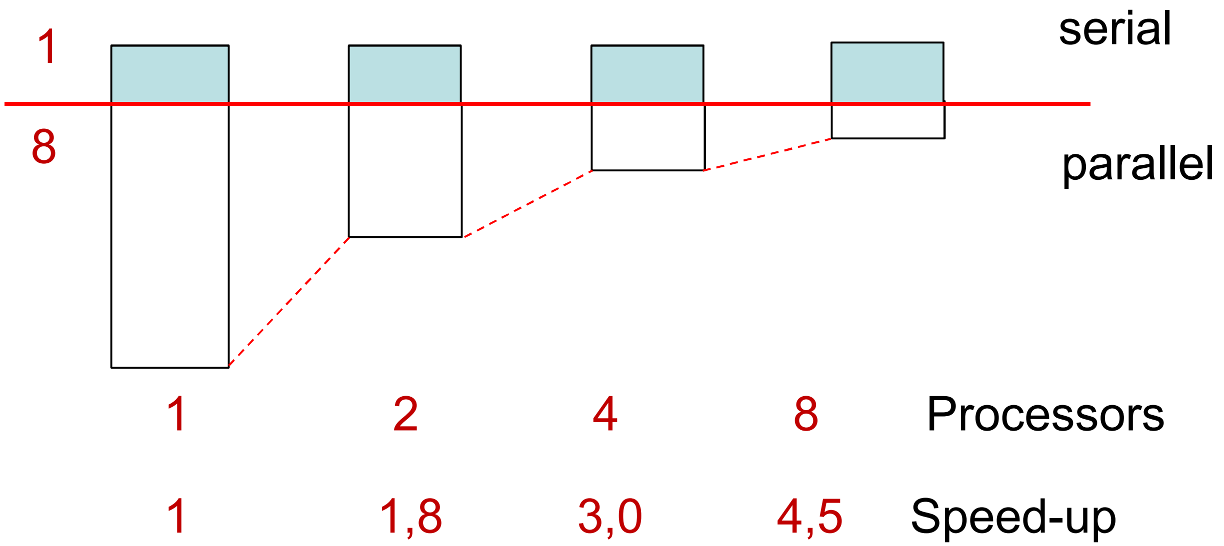
– a system with a larger number of processors in general allows to solve the biggest problems in a reasonable time

– instead of assuming fixed the **size of the problem** we assume that **the parallel execution** time is fixed

– Gustafson's Law.

Gustafson's Law

- Need larger problems for larger numbers of CPUs
- to maintain constant efficiency we need to scale the problem size with the number of CPUs



Performance Models

Fixed-size model: to find the best parallel system by fixing **W** and varying **p**.

Fixed-time model: identify on the curve of the execution time, the pairs ***(W, p)*** keeping fixed $T_p(W,p)$.

Fixed-memory model: we always work with all available memory.

21st Summer School of PARALLEL COMPUTING

References

Ian Foster, *Designing and Building Parallel Programs.*

http://www-unix.mcs.anl.gov/dbpp/

Geoffrey C. Fox, Roy D. Williams, Paul C. Messina. *Parallel Computing Works.*

http://www.netlib.org/utk/papers/mpi-book/mpi-book.html

Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, Jack Dongarra. *MPI: The Complete Reference.*

http://www.netlib.org/utk/papers/mpi-book/mpi-book.html

Message Passing Interface (MPI)

http://www-unix.mcs.anl.gov/mpi/

contiene informazioni sull'implementazione dello standard MPI. Inoltre fornisce link a papers, libri e tutorials.

Beginner's guide to MPI.

http://www.jics.utk.edu/MPI/MPIguide/MPIguide.html

Fornisce una breve referenza per le funzioni MPI più usate.

E' un buon punto di partenza , ma non fornisce molti dettagli.

Books

A. Tanenbaum, *Structured Computer Organization*, Prentice Hall, Inc. (4th ed.), 1999, (Cap 8: Architettura dei computer paralleli)

B. Wilkinson, M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstation and Parallel Computers.* Prentice-Hall, 1999.

W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface* (2nd ed.), The MIT Press, 1996.

R.Chandra, L. Dagum, D. Kohr, D. Maydan, J. Mc Donald, R. Menon, *Parallel Programming in OpenMP*, Academic Press, 2000.

J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torkson, A. White, *Sourcebook of Parallel Computing*, Morgan Kaufmann Publ., 2003.

I. Foster, C.Kesselman, *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publ., 1998.

Books / 1

K. Dowd, C. R. Severance, *High Performance Computing*, O'Reilly, (2nd ed.), 1998.

M. J. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, 1995.

Vipin Kumar, George Karypis, Ananth Grama, Anshul Gupta. *Introduction to Parall Computing: Design and Analysis of Algorithms*.

Joseph Jaja. *An Introduction to Parallel Algorithms*.

L. Ridgway Scott, Terry Clark, Babak Bagheri Scientific paralell computing, Princeton University press, 2005

Archives

Bibliographies on Parallel Processing.
http://liinwww.ira.uka.de/bibliography/Parallel/index.html

Internet Parallel Computing Archive.
http://wotug.ukc.ac.uk/parallel/index.html

Web-pages with links to parallel-processing resources

Parallel Processing Information.
http://www.jics.utk.edu/ParallelInfo/

Nan Schaller's Parallel Computing Page.
http://www.cs.rit.edu/~ncs/parallel.html
An extensive collection of links to parallel processing standards, languages, research groups, courses, and companies. Some of the links are broken, quite old