

Phys 7411: Performance Metrics

Karen Tomko

Karen.Tomko@uc.edu

Spring 2009

Today's topics

- Speedup
- Ahmdal's Law
- Parallel Efficiency
- Serial Fraction
- Scaled Speedup
- Other Metrics

Reading:

- Chapter 5 of *Introduction to Parallel Computing* by Grama et. al.
- Chapter 8 of *Parallel Programming in C with MPI and OpenMP* by Quinn

Why analyze performance?

- To answer questions such as:
 - How much faster can I make my code run?
 - How many processors should I use?
 - How big of a problem can I solve?
- To answer the questions, we must be able to quantify the performance with some set of metrics , for example:
 - Speedup
 - Parallel efficiency

Modeling Parallel Performance

- The execution time of a parallel program is the time that elapses from when the first processor(core) starts executing on the problem to when the last processor completes
- Execution time on processor j:

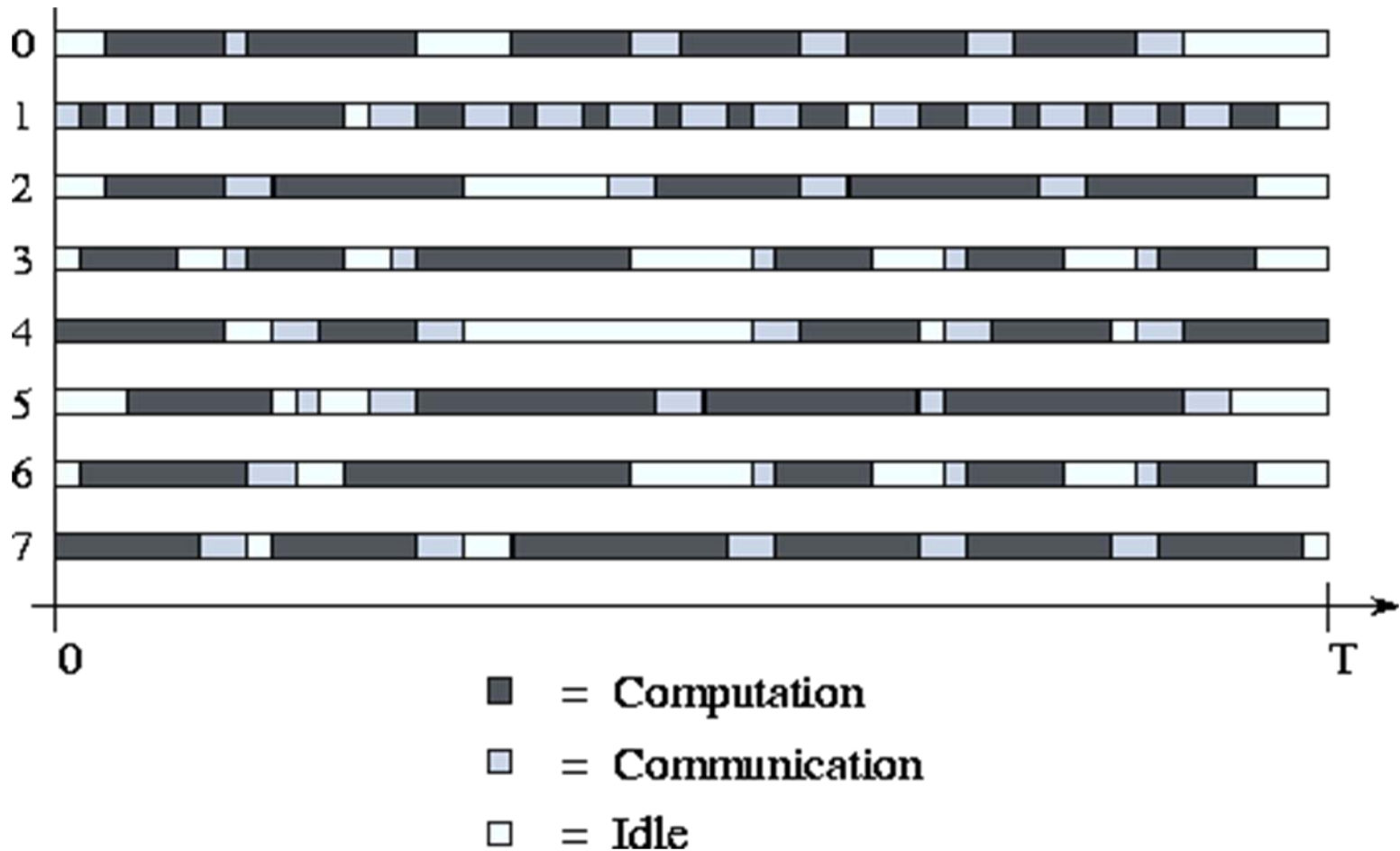
$$t_j = t_{\text{comp}} + t_{\text{comm}} + t_{\text{idle}}$$

where t_{comp} is time for computation (doing the math)

t_{comm} is time spend transferring data between processors

t_{idle} is time spent waiting for other processors

Diagram of a parallel execution



From Ian Foster's Web book.

Speedup

- *Speedup* is the ratio between execution time on a single processor (or core) and execution time on multiple processors.

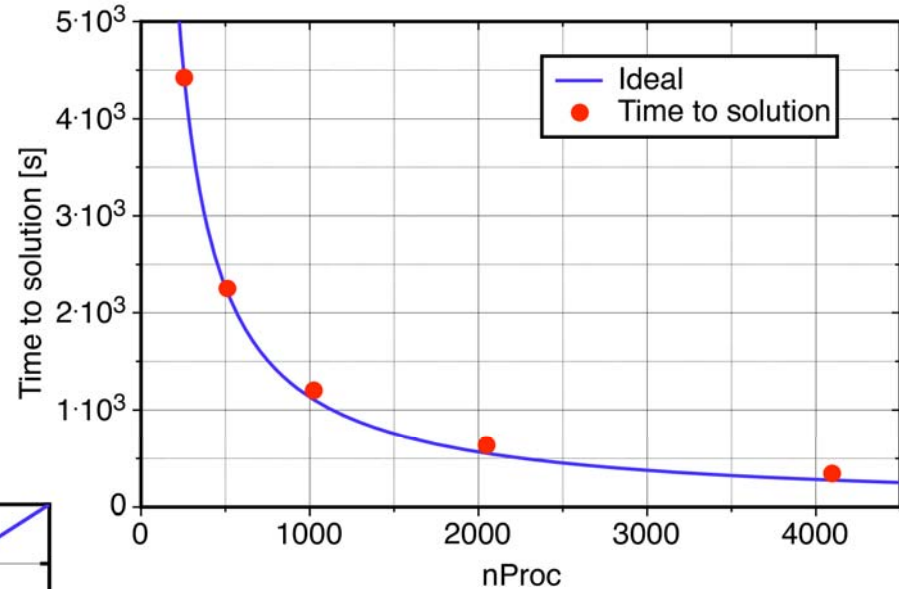
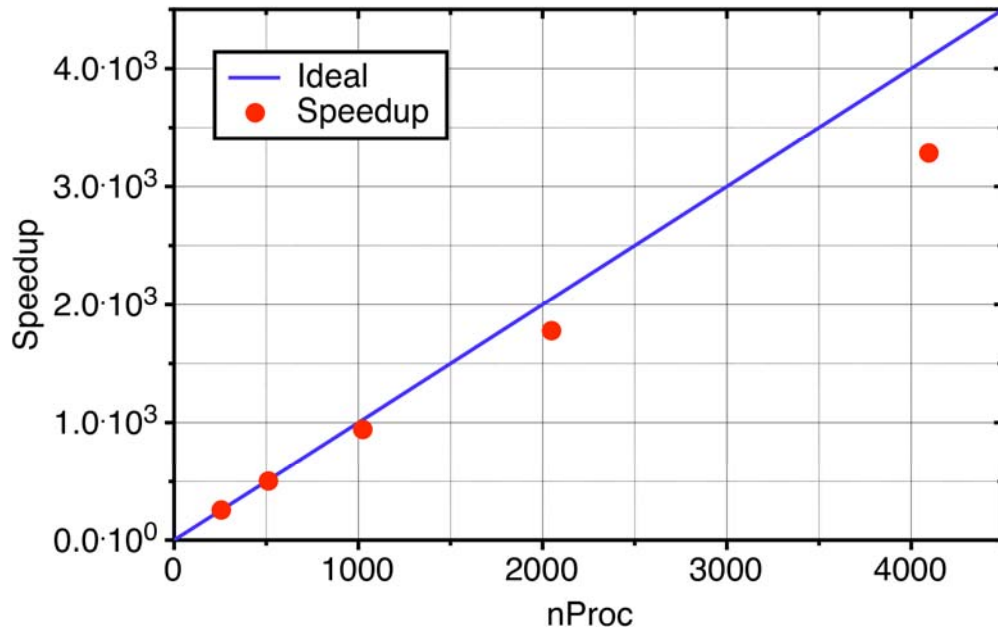
$$S(n) = \frac{T(1)}{T(n)}$$

where $T(1)$ is the time to execute on 1 core and $T(n)$ is the time on n cores

- Perfect or *ideal speedup* when $S(n)=n$
- Speedup up which is greater than n is referred to as *superlinear* speedup, and can occur in practice, due to memory effects.

Typical Performance Curve

Wallclock Execution time and Speedup for 2D HF-QMC DCA Solver on the Cray XT4 system, NCCS, ORNL



Graphs courtesy of Th. Maier, ORNL, Sept 2007.

Amdahl's Law

- Observation: *Every algorithm has a sequential component that will eventually limit the speedup that can be achieved on a parallel computer.*
- More Formally: if the sequential component of an algorithm accounts for $1/s$ of the program's execution time, then the maximum possible speedup that can be achieved on a parallel computer is s .

$$S(n) = \frac{T_s + T_p}{T_s + \frac{T_p}{n}} \leq \frac{T_1}{T_s}$$

- Example:
 - 80% of time spend in one set of loops, parallelize this section of the code and run on 32 processors, assuming linear speedup the time will be
$$t_{32} = 20\% t_1 + (80\% t_1)/32 = .2 t_1 + .8 t_1 /32 = .225 t_1$$
Speedup = $t_1/.225 t_1 = 1/.225 = 4.44$
 - What if it is run on 1000 processors, still with linear speedup
$$t_{1000} = 20\% t_1 + (80\% t_1)/1000 = .2 t_1 + .8 t_1 /1000 = .2008 t_1$$
Speedup = $t_1/.2008 t_1 = 1/.2008 = 4.98$

Parallel Efficiency

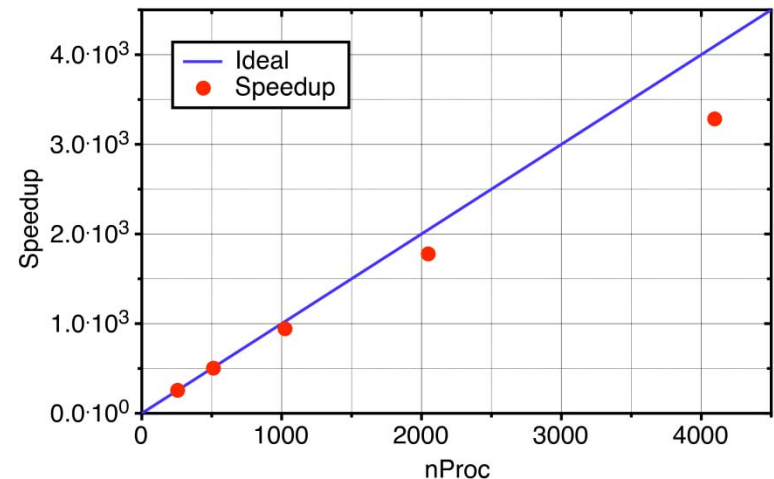
- Goal:
 - Characterize the effectiveness with which an algorithm uses the computational resources of a parallel computer in a way that is independent of problem size.

- Parallel Efficiency equation

$$E = \frac{S}{n} = \frac{T(1)}{nT(n)}$$

- For example, for the HF-QMC solver

- $S \cong 3300$ for $n = 4096$
- $E = 3300/4096 = 80.6\%$



Determining Serial Fraction, f

Consider: $T(1) = T_s + T_p$

$$T(n) = T_s + \frac{T_p}{n}$$

$$T(n) = T_s + \frac{T(1) - T_s}{n} \quad \text{and} \quad f = \frac{T_s}{T(1)}$$

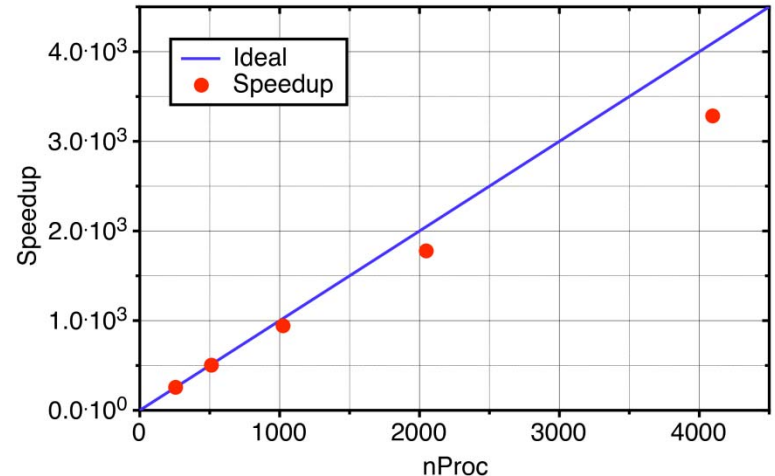
$$T(n) = f \times T(1) + \frac{T(1) - f \times T(1)}{n}$$

$$\frac{T(n)}{T(1)} = \frac{1}{S} = f + \frac{1-f}{n}$$

$$f = \frac{1/S - 1/n}{1 - 1/n}$$

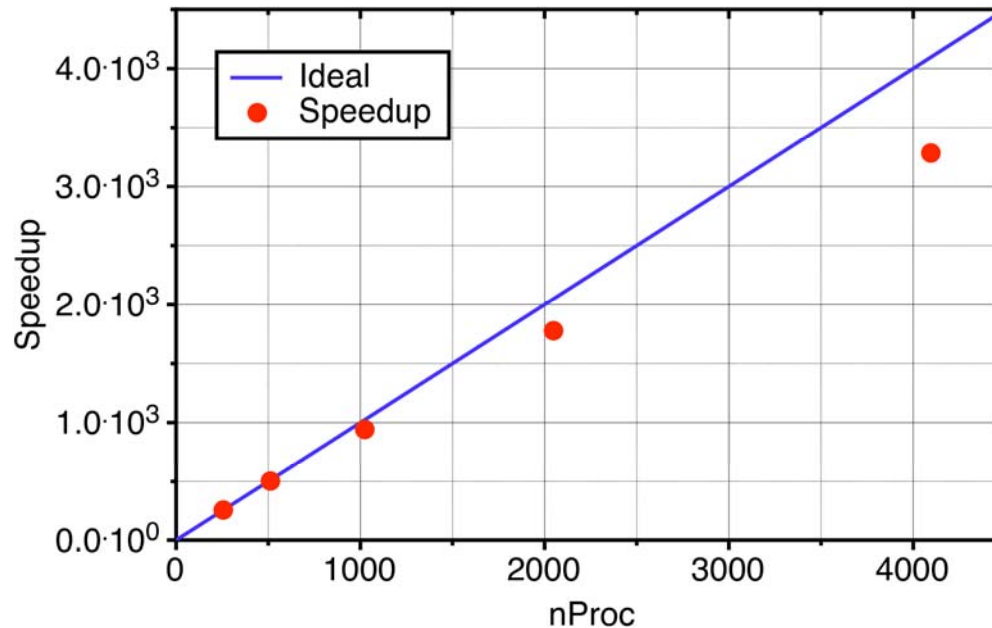
Serial Fraction (continued)

- For a problem of fixed size,
 - if the serial fraction does not increase with n , then parallelism is limited by T_s
 - If the serial fraction increases with n , then parallel overheads are increasing with n and limiting performance (global communication, load imbalance)
 - For the HF QMC example
 - $S \cong 3300, n = 4096, f = .00007$
 - $S \cong 1800, n = 2048, f = .00006$



Scaled Speedup

- Does it make sense to run this problem on 8000 cores? What about 40,000 cores?



Scaled Speedup

- On a bigger system, we will solve bigger problems
- Increase problems size linearly with number of processors
- Best case when then the time for the bigger problem of size, nN , on n processors will be the same as the execution time for the original problem size, N , on 1 processor.

Isoefficiency

- Some Observations:
 - With fixed problem size Efficiency decreases with increasing numbers of processors, n .
 - Efficiency increases with increasing problem size, N .
- Isoefficiency Function:
 - Scaled problem Size
 - the amount of computation performed is increased as more processors are used, keeping E constant.
 - This function of $N = F(P)$ is called an algorithm's *isoefficiency function*

Other Metrics

- GFLOP/s
 - Giga FLoating point Operations / Second
 - Percentage of peak
- Memory Usage
 - As a function of problem size parameters
 - For example: 6 arrays, double complex, $N \times N \times N$
- Memory Bandwidth
 - Is the problem compute or memory bound?
- Cost or Processor-time product
 - $C = n T(n)$
 - used to calculate resource usage (RUs or SUs)

Communication Costs

- Latency
 - Delay
 - Time that it takes to send a byte of data from one processor to another
- Bandwidth
 - Rate or capacity
 - Bytes / second
- Typical Point-to-Point communication latency model

$$T_{p2p} = T_{startup} + \text{Bandwidth} \times \text{Size}$$

Cost for Global Communications

- Often quantified in terms of number of point-to-point messages required
- Consider a Reduction (e.g. global maximum) on 8 processors

$\log_2 n$
message
exchanges

