Parallel Performance Concepts

Beth Richardson
NCSA Consultant and Trainer
bethr@ncsa.uiuc.edu

NCSA
University of Illinois at Urbana-Champaign

Outline

Speedup and Efficiency
Amdahl's Law
Limitations on Speedup
Superlinear Speedup
Price Performance

NCSA
University of Illinois at Urbana-Champaign

Speedup

We don’t describe performance of a parallel˝code in terms of Megaflops

The two most commonly used terms for describing the performance of a parallel code are

- speedup and efficiency

They tell us

- how much performance gain is achieved by parallelizing a code
- or, the benefit of solving a problem in parallel

University of Illinois at Urbana-Champaign

Speedup Formula

Speedup of your parallel code is defined as:

$$S_P = \frac{\text{time for a single processor run}}{\text{time using p processors}}$$

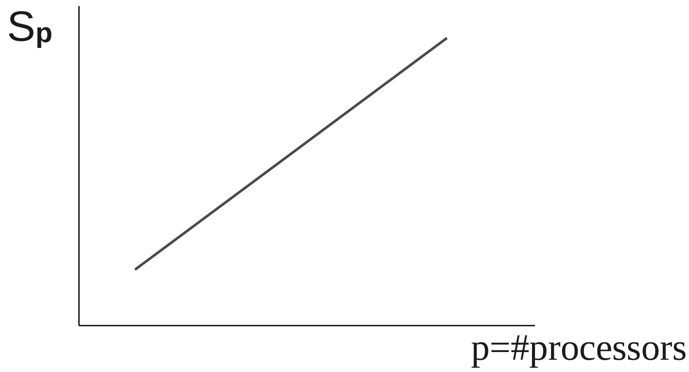
Perfect Speedup

- If it takes one processor an amount of time t to do a task then p processors can do the task in time t/p. This is only achieved in special situations.

University of Illinois at Urbana-Champaign

Perfect Speedup

Perfect or Linear Speedup



Speedup Formula (Cont)

Range of S_P

- **Typically S_P < p Speedup is between 0 and p**

The formula is good measure to see how your code scales as you add more processors.

The scalability of a parallel program is defined as:

- **its ability to achieve performance proportional to the number of processors**

We expect to see S>1, and the larger the better.

Efficiency

Closely related to speedup is efficiency

$E_P = S_P / p$

$E_P <= 1$ Efficiency is between 0 and 1

When $E_P = 1$ this corresponds to perfect speedup of $S_P = p$

Can think of efficiency as the average speedup per processor.

Processors can't devote 100% of their time to computations. So efficiency measures the fraction of time the processors are usefully employed.

University of Illinois at Urbana-Champaign

Speedup Example

For our example, we would like to add together 2 vectors of 1000 elements each

$x_i + y_i$ i=1,1000

- where suppose p=7 is the number of processors
- and suppose the time for 1 addition is t

Single processor time

- 1000t
- this is the numerator of the Speedup formula

University of Illinois at Urbana-Champaign

Speedup Example (Cont.)

Parallel time using p processors

- 7 processors can do 142 adds each in parallel in time 142t, for a total of 994 adds.
- then 6 processors idle while one of the processors does an extra 6 adds, for an extra 6t.
- the total parallel time is 148t
- this is the denominator of the speedup formula

Speedup Example (Cont)

Thus speedup and efficiency are:

$$S_P = \frac{1000t}{148t} = 6.7 < p$$

$$E_P = \frac{6.7}{7} < 1$$

The less than perfect speedup is due to load imbalance.

Amdahl's Law

A speedup formula is attributed to Amdahl.

$$S_P = \frac{1}{f + (1-f)/p}$$

f is the fraction of operations done sequentially with one processor, where $0 <= f <=1$
(1-f) is the fraction of operations done in perfect parallelism
Note when f=0, S_P =p or perfect parallelism.

Amdahl's Law Concepts

This law shows that speedup is limited by the fact that not ALL parts of our program can be run in parallel

Even if we use an infinite number of processors on our code, its speedup is still limited by 1/f.

The sequential fraction has a strong effect on speedup

This explains the need for large problem sizes

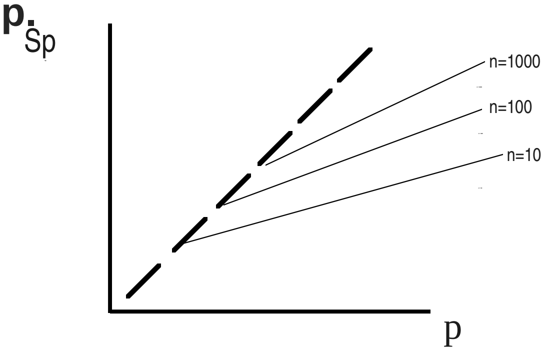
As the problem size increases the opportunity for parallelism grows, and the sequential fraction shrinks

Limitations on Speedup

- The code is I/O bound
- The algorithm is not suitable for a parallel computer
- Network saturation
- Too much memory contention
- The problem size is too small
- The problem size is fixed
- There is load imbalance
- There is too much sequential code
- There is too much parallel overhead

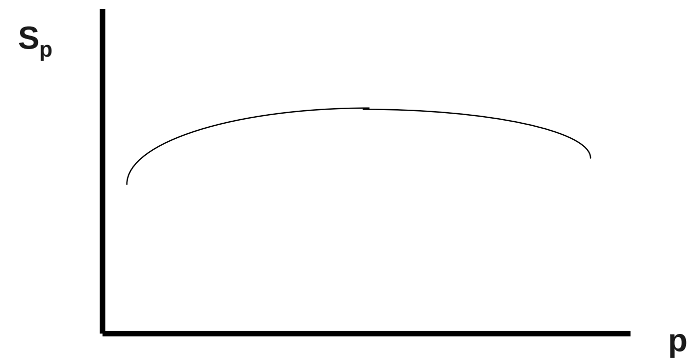
Problem Size is too Small

- Speedup is almost always an increasing function of problem size.
- If there's not enough work to be done by the available processors, the code will show limited speedup.



Problem Size is Fixed

When the problem size is fixed, we can reach a point of negative returns when using additional processors.



NCSA
University of Illinois at Urbana-Champaign

Load Imbalance

Load balancing is defined as:

- the assignment of subtasks to processors so as to keep each processor busy doing useful work as long as possible.

For optimal performance, the workload must be balanced among the processors.

When processors have different work loads some of the processors idle while others are still working.

NCSA
University of Illinois at Urbana-Champaign

Excessive Sequential Code

As shown by Amdahl's law.

If you think this is a problem, you can calculate the sequential fraction of code, or conversely the parallel fraction

- run and time your code with 1 processor and with p processors
- divide the 2 timings and calculate speedup
- Substitute this value into the Amdahl's Law˝formula

Too Much Parallel Overhead

Synchronization

- Some or all˝processes need to synchronize at some point. When all processes are not ready, the ones ready early˝will be idle waiting. Causes a reduction in speedup.

Parallel Process Creation

- When the work done by the parallel processes is too small, the overhead time needed to create the processes limits the savings due to parallelism.

Too Much Parallel Overhead (Cont.)

If you think parallel overhead may be a problem you can calculate it.

Calculating Parallel Overhead

- Run and time your code on 1 processor
- Parallelize your code
- Run and time the parallelized code with 1 processor
- Subtract the 2 timings
- This gives a rough estimate of parallel overhead.

Superlinear Speedup

Defined as speedup that is greater than the number of processors used.

The parallel subtasks have a much smaller slice of the problem than the entire sequential code. They compute on a fraction of the entire dataset and can make better use of cache and registers.

Price Performance

Another parallel performance measure is price performance.

$$\text{price performance} = \frac{\text{performance of applications}}{\text{list price of system}}$$

You can buy more compute power per dollar by spending your money on microprocessor-based parallel computers, rather than pipelined vector computers.