Performance

Ned Nedialkov

Department of Computing and Software
McMaster University, Hamilton, Ontario
Canada

Outline

- Sources of overhead
- Total parallel overhead
- Speedup
- Efficiency
- Scalability
- Communication cost
- Timing

Sources of overhead

- Communication
- Idling — load imbalance, synchronization, presence of serial components
- Excess computation

  The fastest known serial algorithm may be difficult or impossible to parallelize

  We may have to use a poorer, but easier to parallelize algorithm

  The difference in computation performed by the parallel program and the best serial program is the excess computation overhead incurred by the parallel program

Total parallel overhead

Denote serial running time by T_s and parallel running time on p processors by T_p

(We assume solving problems of the same size)

Total overhead is defined as

$$T_o = pT_p - T_s$$

T_s is the time for the fastest serial algorithm solving the problem on 1 processor

pT_p is total time spent over all processors solving the problem

Speedup

Speedup is defined as

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

Usually, $0 < S \leq p$

If $S = p$, we have linear speedup

Theoretically, $S \leq p$

In practice, it may happen that $S > p$: superlinear speedup

Efficiency

Efficiency is a measure of processor utilization in a parallel program

That is, a measure of the fraction of time for which a processor is employed

$$E = \frac{S}{p}$$

If $E = 1$, linear speedup

If $E < 1/p$, slowdown (serial program is faster)

Scalability

Efficiency as p grows, problem size fixed

Efficiency can be written as

$$E = \frac{S}{p} = \frac{T_s}{pT_p} = \frac{1}{1 + \frac{T_o}{T_s}}$$

T_o is an increasing function of p

- every program must contain some serial component
- if the time for this component is t_s, $p - 1$ processors will be idle for $(p-1)t_s$ time
- hence T_o grows at least linearly
- because of communication, idling, and excess computation, it may grow superlinearly

For a given problem size, efficiency goes down as p increases

Efficiency as problem size grows, p fixed

T_o depends on problem size and p

In many cases, T_o grows sublinearly w.r.t. to program size

Efficiency increases as p is fixed and problem sized is increased

Scalable: the efficiency can be kept constant as the number of processing elements is increased, provided that the problem size is increased

Communication cost

Consider send/receive

Their execution can be divided into two phases

- startup
- communication

Their cost varies among systems

Denote the runtime of the startup phase by t_s, and the runtime of the communication phase by t_c

The cost of sending a message consisting of k units is

$$t_s + k \cdot t_c$$

Usually t_s is much larger than t_c

Their values vary among systems

Timing

One can use `double MPI_Wtime(void)` to time MPI code

This function returns elapsed time

Hence, one can measure real, or clock, time

Another approach is to measure system and user time; for example by calling the C `times` function