Performance Analysis and Performance Metrics

Jarmo Rantakokko

Performance Metrics

How do we evaluate and compare different parallel algorithms and implementations?

- How much does the performance depend on the:
  - On the algorithm?
  - On the implementation?
  - On the compiler?
  - On the MPI library?
  - On computer platform?

- Can we in advance (a priori) predict the performance?
- Can we do a posteriori analysis of the observed performance and how?
- How do we present our results in a report?

UPPSALA UNIVERSITET

Traditional performance metrics:

- Execution Time (T=1.24s for 1440x1440 MxM)
- Execution Speed (Gflop/s, Gflop/$, Gflop/W)

⇒ Too hardware dependent (cpu, mem, size) and do not say anything about parallelism!

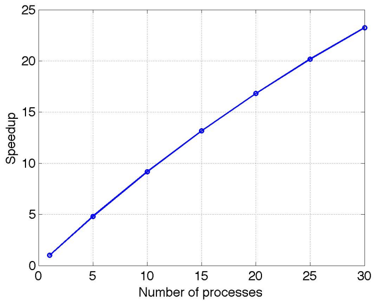
New metrics:

- Speedup [How much faster]
- Sizeup [How much bigger]
- Parallel efficiency [How efficient]
- Isoefficiency function [How scalable]

UPPSALA UNIVERSITET

Speedup

How much faster can we run on P processors than on one single processor? Define speedup $S_P=T_1/T_P$



Problem: Algorithm 1: S_P=30 on 32 processors
Algorithm 2: S_P=15 on 32 processors

⇒ Is Algorithm 1 better? (Compare Enumsort vs Quicksort)

Different interpretations of speedup due to:

1. Algorithm
   - Compare with the best serial algorithm (code)
     Absolut speedup ($S_P \leq P$)
   - Compare with the same code one processor
     Relative speedup (possible with $S_P > P$, how?)

2. Problem size
   - Constant total problem size,
     Fixed size speedup, $S_P = T_1(w)/T_P(w)$ w-work
   - Constant size (work) per processor
     Scaled speedup, $S'_P = P\, T_1(w)/T_P(pw)$
     (Solve a P-times larger problem in parallel ⇔ solve P original problems on one processor and compare speedup)

Note: $S'_P \geq S_P$, why? (Explained by Amdahl's law)

Amdahl's law

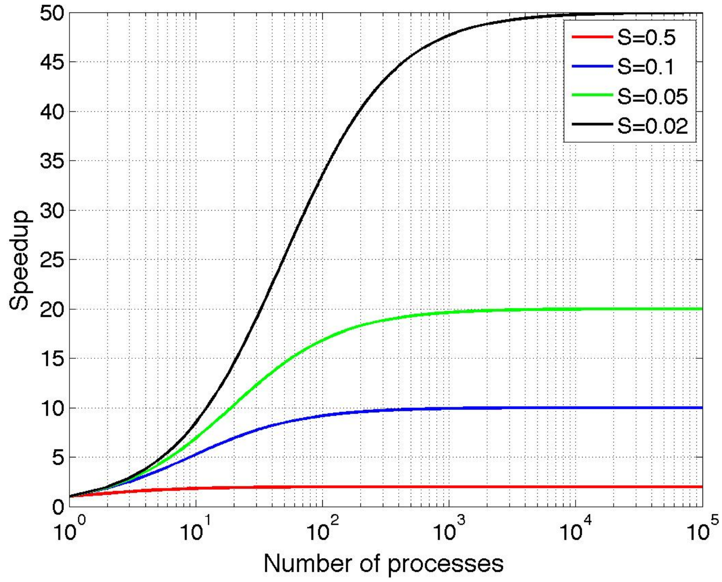
Assume that a fraction S of a program can not be parallelized (serial part, e.g., I/O) and the other fraction 1-S is perfectly parallel.

$$T_1 = (S+(1-S))\, W$$
$$T_P = (S+(1-S)/P)\, W$$

$$\Rightarrow S_P = T_1/T_P = 1 / (S+(1-S)/P) \rightarrow 1/S \text{ as } P \rightarrow \infty$$

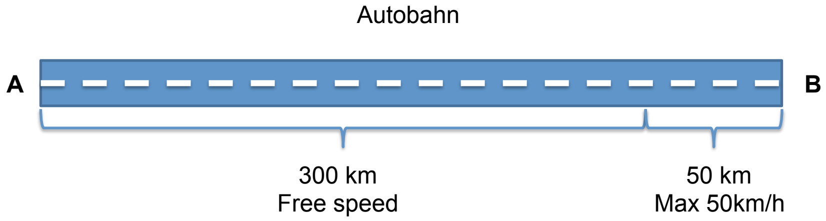
Amdahl's law: Speedup is bounded $S_P \leq 1/S$

UPPSALA UNIVERSITET



UPPSALA UNIVERSITET

Example:



Volkswagen: T= 300/100 + 50/50 = 4h
Mercedes: T= 300/200 + 50/50 = 2.5h
Porsche T= 300/300 + 50/50 = 2h
Fighter Jet T= 300/2000+50/50 = 1.15h

No matter how much power you have the "serial part" will ultimately limit your performance.

Note 1: Amdahl's law is a rough simplification, does not consider communication, synchronization and different levels of parallelism but it gives always an upper estimate of what we possibly can achieve.

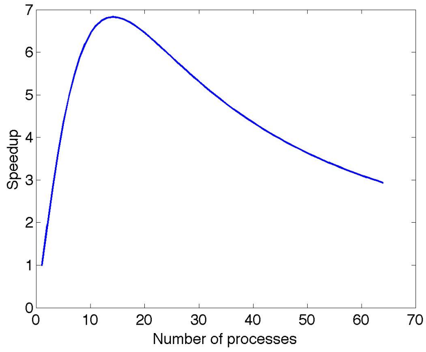


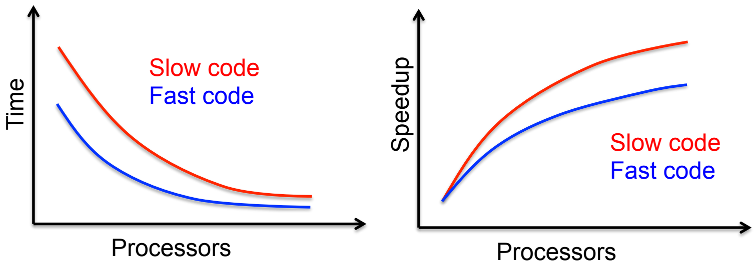
Figure: An application where the communication part is dominating and the parallel overhead increases with P.

Note 2: We want to use (we can use) parallel computers to solve bigger problems. For real applications the parallel work grows faster than the serial work as we increase the problem size.

$\Rightarrow$ The serial fraction S decreases!
Increasing the problem size gives higher speedup, explains why $S'_P \geq S_P$.
[*Gustafson-Barsis' law:* $S'_P = P - S(P-1)$]

Example; *Matrix-Matrix multiplication*
communication $\propto n^2$ while computation $\propto n^3$
(Computations are perfectly parallel, communication is a "serial" part that we can not speedup up by increasing the number of processors)
The serial fraction S decreases as we increase n!

Note 3: Speedup favors slow processors, non-optimized code and bad compilers.



When optimizing the code it is usually the parallel fraction (computational part) that is improved, not the serial fraction (I/O etc)

$\Rightarrow$ The serial fraction S becomes larger for the fast code and speedup is penalized.

(What happens if you cache optimize your MxM in assignment 1?)

Sizeup

How much bigger problem can we solve on P processors than on one processor on the same time?

Find w_P such that $T_P(w_P) = T_1(w_1)$ constant runtime

$\Rightarrow$ Define sizeup $Sz_P = w_P/w_1$

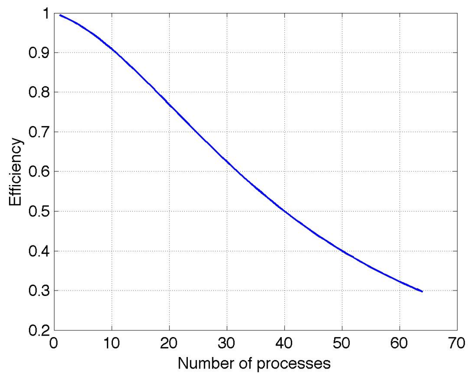
Note 1: Sizeup is less dependent of processor speed and level of optimization than speedup. (Runtime is not an explicit part of the metric.)

Note 2: Sizeup reflects the second goal of parallel computing, i.e., we want to solve larger problems in the same time. E.g., in weather forecasting we use the extra computational power for a more accurate solution (with more grid points).

Parallel Efficiency

How efficiently are we using the parallel computer?

Define parallel efficiency: $E_P = S_P/P$ $(0<E_P\leq 1)$
(For ideal speedup $S_P=P$ we get full efficiency)



Isoefficiency function

How scalable is our algorithm? (Theoretical metric)

It has been observed that efficiency increases with the problem size and decreases with increasing the number of processors. The idea is to keep $E_p(w)$ constant, while increasing w and p simultaneously.

Def: "How much should we increase the problem size when we add more processors to keep the efficiency E_P constant"
$\Rightarrow$ w=f(p) *isoefficiency function*

If f(p) is linear (or constant) the algorithm is highly scalable
If f(p) is exponential the algorithm is poorly scalable

Model the runtime theoretically and compute w=f(p) from $E_P=T_1(w)/pT_P(w)$ for a constant E_P. The isoefficiency function is used to compare the growth rates of two algorithms theoretically.

UPPSALA UNIVERSITET

Note:
Very difficult to model/predict parallel runtimes with theoretical models. Processors run asynchronously, unpredictable cache utilization, disturbances from other processes and users. (Can still compare algorithms theoretically and derive upper limits, e.g., roofline model)

⇒ Large variations in runtime from run to run!

When measuring runtime, make many runs and take the *best time*. Gives least disturbances and more consistent timing (speedup) with less spikes (outliers).

[*Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers*, David H. Bailey, Supercomputing Review, Aug. 1991, pg. 54—55.]

UPPSALA UNIVERSITET

Parallel Overhead

Why don't we get optimal speedup (S_P=P, E_P=1) when we run on a parallel computer?

1. Interprocessor communication
2. Load imbalance, runtime depends on the most heavily loaded processor
3. Serial parts or parts with limited parallelism
4. Extra computations compared to the best serial algorithm (e.g., recursive loops)
5. Synchronization of processors
6. Startup/rescheduling time of processes
7. Time sharing of shared resources (processors, memory, network bandwidth) with other users and processes