PERFORMANCE ANALYSIS

Course "Parallel Computing"

Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)

Wolfgang.Schreiner@risc.jku.at

http://www.risc.jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Evaluating Parallel Programs

We achieved a speedup of 10.8 on $p = 12$ processors with problem size $n = 100$.

- Multiple programs may satisfy this observation:
  - Program 1:
    $T = n + n^2/p$.
  - Program 2:
    $T = (n + n^2)/p + 100$
  - Program 3:
    $T = (n + n^2)/p + 0.6p^2$

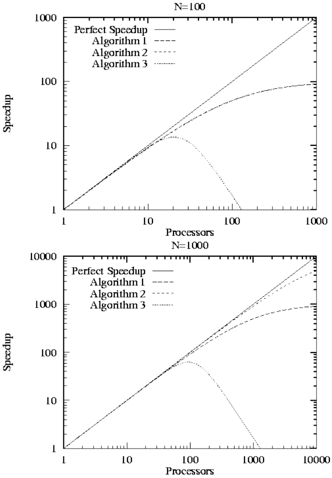


Figure 3.1, Ian Foster: DBPP

We have to evaluate programs on varying parameters.

Speedup and Efficiency

- (Absolute) speedup S_p and efficiency E_p:

$$S_p = \frac{T}{T_p} \qquad\qquad E_p = \frac{S_p}{p} = \frac{T}{p \cdot T_p}$$

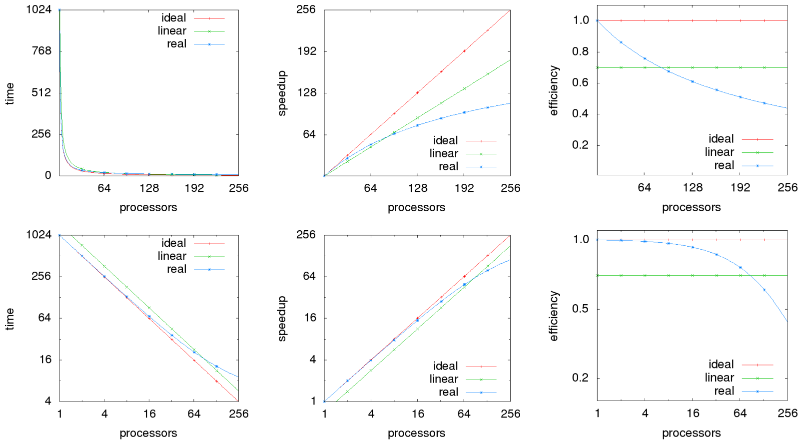
  - T: execution time of sequential program.
  - T_p: execution time of parallel program with p processors.
- Relative speedup $\overline{S}_p$ and efficiency $\overline{E}_p$:

$$\overline{S}_p = \frac{T_1}{T_p} \qquad\qquad \overline{E}_p = \frac{\overline{S}_p}{p} = \frac{T_1}{p \cdot T_p}$$

  - Use for comparison the parallel program with 1 processor.
  - Measures “scalability” rather than “performance”.
- Typical ranges: $S_p \leq \overline{S}_p \leq p$ and $E_p \leq \overline{E}_p \leq 1$.
  - If $\overline{S}_p > p$, we have a “superlinear speedup”.
  - If $S_p > \overline{S_p}$, then $T > T_1$.

Speedup denotes the “performance” of parallelism, efficiency relates this performance to the invested “costs”.

Diagrams



Logarithmic scales may yield additional insights.

Superlinear Speedups

Can the speedup be larger than the number of processors?

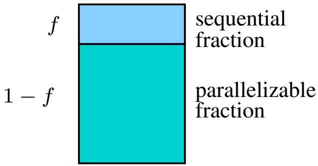
- Simple theoretical argument: “no”.
  - We can simulate the execution of a parallel program with p processors on a single processor in time $p \cdot T_p$. Thus $T \leq p \cdot T_p$ and $S_p = T/T_p \leq p$.
- However, practical observation: “yes”.
  - Cache effects: a system with p processors has typically also p times as much cache which yields more cache hits.
  - Search anomalies: if the computation involves a “search”, one processor may be lucky to find the result early.
- These advantages can be “practically” not achieved on a single processor system.

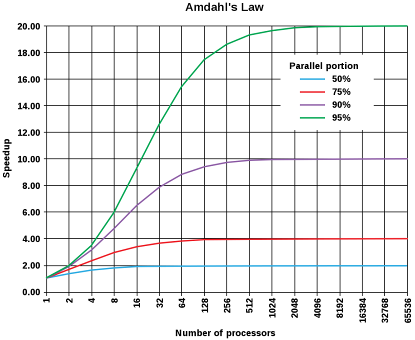
However, often super-linear speedups indicate program errors.

Amdahl's Law

Assume that a workload contains a sequential fraction f.

- Amdahl's law: $S_p \leq \frac{1}{f+\frac{1-f}{p}} \leq \frac{1}{f}$
  - Speedup has an upper limit determined by f.





Amdahl's law, en.wikipedia.org

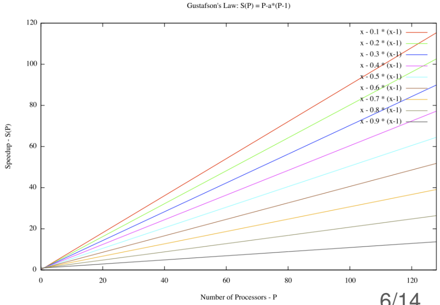
Speedup is limited by the sequential fraction of a workload.

Gustafson's Law

Assume workload can be scaled as much as time permits.

- Amdahl: $S_p \leq \frac{1}{f + \frac{1-f}{p}}$
  - Fixed work load $T = f \cdot T + (1 - f) \cdot T$
  - $S_p \leq \frac{T}{f \cdot T + \frac{(1-f) \cdot T}{p}} = \frac{1}{f + \frac{1-f}{p}}$
- Gustafson: $S_p \leq f + p \cdot (1 - f)$
  - Scalable work load $T_p = f \cdot T + p \cdot (1 - f) \cdot T$
  - $S_p \leq \frac{f \cdot T + p \cdot (1-f) \cdot T}{f \cdot T + \frac{p \cdot (1-f) \cdot T}{p}} = \frac{f \cdot T + p \cdot (1-f) \cdot T}{T} = f + p \cdot (1 - f)$

If the parallelizable workload grows linearly with the numer of processors, the speedup grows correspondingly such that the efficiency remains constant.



Gustafson's law, en.wikipedia.org

Scalability Analysis

We have to scale the workload to keep the efficiency constant.

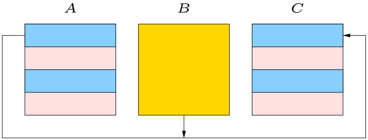
- Assume $T_{p,n} = \frac{T_n + P_{p,n}}{p}$.
  - $T_{p,n}$: the parallel time with p processors for problem size n.
  - T_n: the basic work performed by the sequential program.
  - $P_{p,n}$: the extra work performed by the parallel program.
- Then $E_{p,n} = \frac{T_n}{p \cdot T_{p,n}} = \frac{T_n}{T_n + P_{p,n}}$.
  - $E_{p,n}$: the efficiency with p processors for problem size n.
  - Thus $T_n = \frac{E_{p,n}}{1 - E_{p,n}} \cdot P_{p,n}$; for achieving constant efficiency E, we have to ensure $T_n = \frac{E}{1-E} \cdot P_{p,n} = K_E \cdot P_{p,n}$.
- Isoefficiency function: $I_p^E = K_E \cdot P_{p,n_p}$
  - n_p: a function that maps processor number p to problem size n_p such that $T_{n_p} = K_E \cdot P_{p,n_p}$.
  - I_p^E describes how much the basic work load has to grow for growing processor number p to keep efficiency E.

The less I_p^E grows, the more scalable the program is.

Example: Matrix Multiplication

Multiplication of two square matrices A, B of dimension n.

- Row-oriented parallelization.
  - A is scattered, B is broadcast, C is gathered.
- $T_n = n^3$ and $P_{p,n} = 3pn^2$
  - $T_{p,n} = \frac{n^3}{p} + 3n^2$
  - $P_{p,n} = T_{p,n} \cdot p - T_n = 3pn^2$
- $T_{n_p} = K_E \cdot P_{p,n_p}$
  - ${n_p}^3 = K_E \cdot 3pn_p^2$
  - $n_p = K_E \cdot 3p$
- $I_p^E = K_E \cdot P_{p,n_p}$
  - $I_p^E = K_E \cdot 3p \cdot (K_E \cdot 3p)^2 = (K_E)^2 \cdot 27p^3$



The matrix dimension n must grow with $\Omega(p)$, the basic work load thus grows with $\Omega(p^3)$.

Example: Matrix Multiplication

Often only asymptotic estimations are possible/needed.

- $T_n = \Theta(n^3)$ and $P_{p,n} = \Theta(p \log p + n^2 \sqrt{p})$
  - Fox-Otto-Hey algorithm on $\sqrt{p} \times \sqrt{p}$ torus.
- $T_{n_p} = \Omega(P_{p,n_p})$
  - ${n_p}^3 = \Omega(p \log p + n_p^2 \sqrt{p})$
  - ${n_p}^3 = \Omega(n_p^2 \sqrt{p}) \Rightarrow n_p = \Omega(\sqrt{p})$
  - ${n_p}^3 = \Omega(\sqrt{p}^3) = \Omega(p\sqrt{p}) = \Omega(p \log p)$ ✓
  - $n_p = \Omega(\sqrt{p})$
- $I_p^E = \Omega(P_{p,n_p})$
  - $I_p^E = \Omega(p \log p + p\sqrt{p}) = \Omega(p\sqrt{p})$

The matrix dimension n must grow with $\Omega(\sqrt{p})$, the basic work load thus grows with $\Omega(p\sqrt{p})$.

Modeling Program Performance

$$T = \frac{1}{p}(T_{\mathrm{comp}} + T_{\mathrm{comm}} + T_{\mathrm{idle}})$$

- T_{comp}: computation time.
- T_{comm}: communication time.
- T_{idle}: idle time.

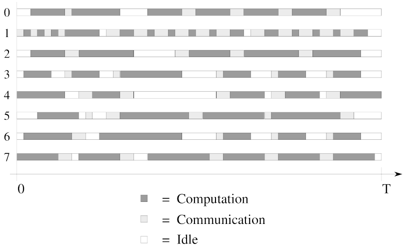


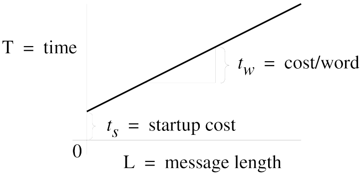
Figure 3.2, Ian Foster: DBPP

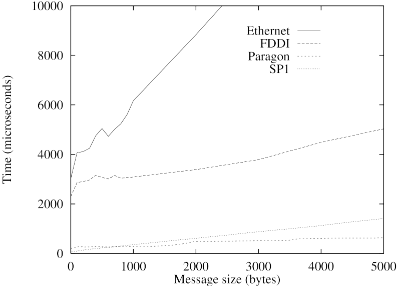
The parallel program overhead mainly stems from communicating and idling.

Communication Time

$$T_L = t_s + t_w \cdot L$$

- T_L: the time for senting a message of size L.
- t_s: the fixed message startup time.
- t_w: the transfer time per word of the message.





Figures 3.3 and 3.4, Ian Foster: DBPP

Typically $t_s \gg t_w$, thus it is better to send a single big message rather than many small messages.

Idle Time

- Apply load-balancing techniques.
- Overlap computation and communication.
  - Have multiple threads per processor.
  - Let process interleave computation and communication.

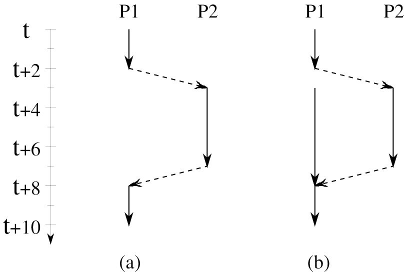


Figure 3.5, Ian Foster: DBPP

Structure the program to minimize idling.

Execution Profiles

Poor performance may have multiple reasons.

- Replicated computation.
- Idle times due to load imbalances.
- Number of messages transmitted.
- Size of messages transmitted.

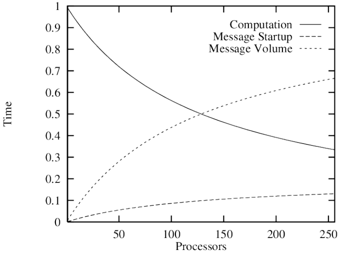


Figure 3.8, Ian Foster: DBPP

Modeling/measuring execution profiles may help to improve the design of a program.

Experimental Studies

- Design experiment.
  - Identify data to be obtained.
  - Determine parameter ranges.
  - Ensure adequacy of measurements.
- Perform experiment.
  - Repeat runs to verify reproducability.
  - Drop outliers, average the others.
- Fit observed data $o(i)$ to model $m(i)$:
  - Least square fitting: minimize

$$\sum_i (o(i) - m(i))^2$$

  - Scaled least square fitting: minimize

$$\sum_i (\frac{o(i) - m(i)}{o(i)})^2$$

(giving more weight to smaller values).

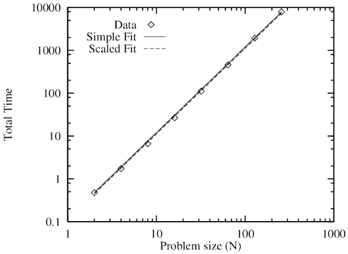


Figure 3.9, Ian Foster: DBPP