Performance of Parallel Programs

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

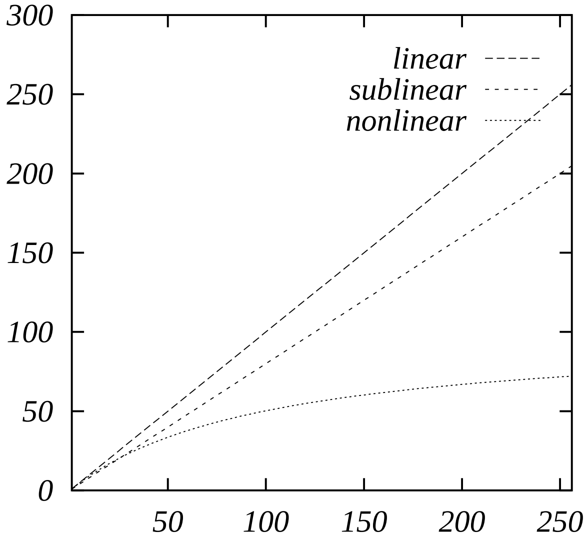
Speedup and Efficiency

- (Absolute) Speedup: $S_n = \frac{T_s}{T_p(n)}$.
  - T_s ... time of sequential program.
  - $T_p(n)$ ... time of parallel program with n processors.
  - $0 < S_n \leq n$ (always?)
  - Criterium for performance of parallel program.
- (Absolute) Efficiency: $E_n = \frac{S_n}{n}$.
  - $0 < E_n \leq 1$ (always?)
  - Criterium for expenses of parallel program.
- *Relative* speedup and efficiency use $T_p(1)$ instead of T_s.
  - $T_p(1) \geq T_s$ (why?)
  - Relative speedup and efficiency are larger than their absolute counterparts.

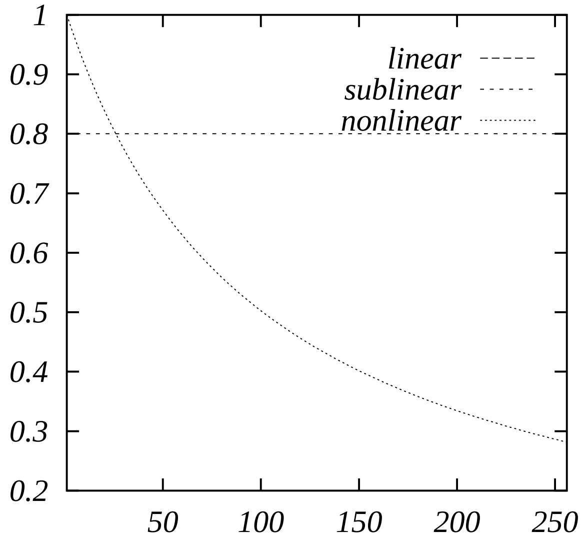
Observations depend on (size of) input data.

Speedup and Efficency Diagrams

Speedup

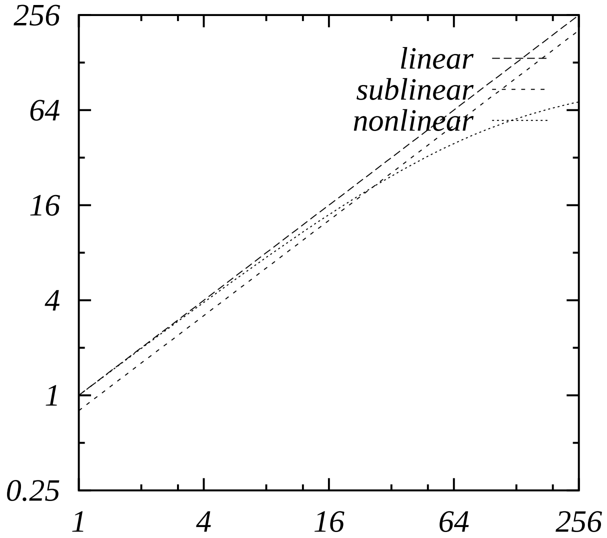


Efficiency

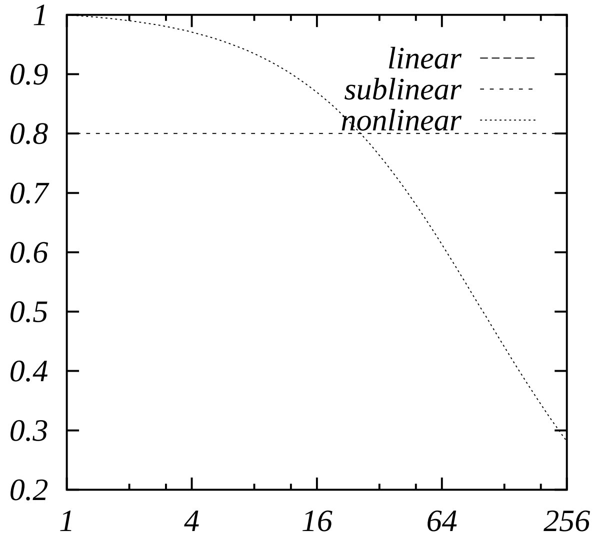


Logarithmic Scales

Speedup

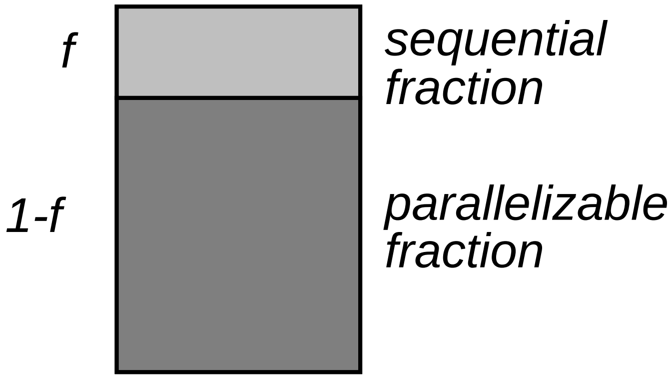


Efficiency



Amdahl's Law

Sequential Program



- Speedup $S_n \leq \frac{1}{f + \frac{1-f}{n}}$
- Limit $S_n \leq \frac{1}{f}$
- Example $f = 0.01 \Rightarrow S_n < 100!$

Speedup is limited by the sequential fraction of a program!

Superlinear Speedup

Question: Can speedup be larger than the number of processors?

$S_n > n, E_n > 1?$

Answer: In principle, no.

Every parallel algorithm solving a problem in time T_p with n processors can be in principle simulated by a sequential algorithm in $T_s = nT_p$ time on a single processor.

However, simulation may require some execution overhead.

Speedup Anomalies

Still sometimes superlinear speedups can be observed!

- Memory/cache effects
  - More processors typically also provide more memory/cache.
  - Total computation time decreases due to more page/cache hits.
- Search anomalies
  - Parallel search algorithms.
  - Decomposition of search range and/or multiple search strategies.
  - One task may be "lucky" to find result early.

Both "advantages" can "in principle" be also achieved on uniprocessors.

Scalability

- Scalable algorithm

  Large efficiency also with larger number of processors.

- Scalability analysis

  Investigate performance of parallel algorithm with

  – growing processor number,

  – growing problem size,

  – various communication costs.

- Various workload models

Fixed Workload Model

Amdahl's Law revisited:

- Assumption: problem size fixed.
  - Sequential and parallelizable fraction.
  - Total time $T = T_s + T_p$.
- Goal: minimize computation time.

  $$S_n \leq \frac{T_s+T_p}{T_s+\frac{T_p}{n}} \leq \frac{T_s+T_p}{T_s} = \frac{1}{\frac{T_s}{T_s+T_p}} = 1/f.$$
- Applies when given problem is to be solved as quickly as possible.
  - Financial market predictions.
  - Being faster yields a competitive advantage.

For not perfectly scalable algorithms, efficiency eventually drops to zero!

Fixed Time Model

Gustavson's Law

- Assumption: available time is constant.
- Goal: solve largest problem in fixed time.
- Strategy: scale workload with processor number.
  - $T = T_s + nT_p$
  - $S_n = \frac{T_s+nT_p}{T_s+n\frac{T_p}{n}} = \frac{T_s+nT_p}{T_s+T_p} = \frac{fT+n(1-f)T}{fT+(1-f)T} = f + n(1-f)$
- Speedup grows linearly with n!
- Applies where a "better" solution is appreciated.
  - Refined simulation model.
  - More accurate predictions.

Efficiency remains constant.

Fixed Memory Model

Sun & Ni

- Assumption: available memory is constant.
- Goal: solve largest problem in fixed memory.
- Strategy: scale problem size with available memory.
  - $T = T_s + cnT_p, c > 1$
  - $S_n = \frac{T_s + cnT_p}{T_s} + \frac{cnT_p}{n} = \frac{T_s + cnT_p}{T_s + cT_p} = \frac{f + cn(1-f)}{f + c(1-f)} \approx n$
- Applies when memory requirements grow slower than computation requirements.

Efficiency is maximized.

The Isoefficiency Concept

Komon & Rao

- Efficiency $E_n = \frac{w(s)}{w(s)+h(s,n)}$
  - s ... problem size,
  - $w(s)$ ... workload,
  - $h(s,n)$ ... communication overhead.
- As processor number n grows, communication overhead $h(s,n)$ increases and efficiency E_n decreases.
- For growing s, $w(s)$ usually increases much faster than $h(s,n)$.

An increase of the workload $w(s)$ may outweigh the increase of the overhead $h(s,n)$ for growing processor number n.

The Isoefficiency Concept

- Question: For growing n, how fast must s grow such that efficiency remains constant?
  - $E_n = \frac{1}{1+\frac{h(s,n)}{w(s)}}$
  - $\Rightarrow w(s, n)$ should grow in proportion to $h(s, n)$.
- Constant efficiency E
- Workload $w(s) = \frac{E}{1-E} h(s, n) = Ch(s, n)$
- Isoefficiency function $f_E(n) = Ch(s, n)$

If workload $w(s)$ grows as fast as $f_E(n)$, constant efficiency can be maintained.

Scalability of Matrix Multiplication

- n processors, $s \times s$ matrix.
- Workload $w(s) = O(s^3)$.
- Overhead $h(s,n) = O(n \log n + s^2\sqrt{n})$
- $w(s)$ must asymptotically grow at least as fast as $h(s,n)$.
  1. $w(s) = \Omega(h(s,n))$.
  2. $\Rightarrow s^3 = \Omega(n \log n + s^2\sqrt{n})$.
  3. $\Rightarrow s^3 = \Omega(n \log n) \wedge s^3 = \Omega(s^2\sqrt{n})$.
  4. $s^3 = \Omega(s^2\sqrt{n}) \Leftrightarrow s = \Omega(\sqrt{n})$.
  5. $s = \Omega(\sqrt{n}) \Rightarrow s^3 = \Omega(n\sqrt{n}) \Rightarrow s^3 = \Omega(n \log n)$.
  6. $\Rightarrow w(s) = \Omega(n\sqrt{n})$.
- Isoefficiency $f_E(n) = O(n\sqrt{n})$
- Matrix size $s = O(\sqrt{n})$

Matrix size s must grow with at least $\sqrt{n}$!

More Performance Parameters

- Redundancy $R(n)$
  - Additional workload in parallel program.
  - $R(n) = \frac{W_p(n)}{W_s}$
  - $1 \leq R(n) \leq n$.
- System utilization $U(n)$
  - Percentage of processors kept busy.
  - $U(n) = R(n)E(n) = \frac{W_p(n)}{nT_p(n)}$
  - $\frac{1}{n} \leq E(n) \leq U(n) \leq 1$.
  - $\frac{1}{n} \leq R(n) \leq \frac{1}{E(n)} \leq n$.
- Quality of Parallelism $Q(n)$
  - Summary of overall performance.
  - $Q(n) = \frac{S(n)E(n)}{R(n)} = \frac{T_s^3}{nT_p^2(n)W_p(n)}$
  - $0 < Q(n) \leq S(n)$

Parallel Execution Time

Three components

1. Computation Time T_{comp}

Time spent performing actual computation; may depend on number of tasks or processors (replicated computation, memory and cache effects).

2. Communication Time T_{msg}

- Time spent in sending and receiving messages
- $T_{\text{msg}} = t_s + t_w L$
- startup cost, cost/word, message length.

3. Idle Time T_{idle}

- Processor idle due to lack of computation or lack of data,
- Load balancing,
- Overlapping computation with communication.

Execution Profiles

Determine ratio of

1. Computation time,
2. Message startup time,
3. Data transfer costs,
4. Idle time

as a function of the number of processors.

Guideline for redesign of algorithm!

Experimental Studies

Parallel programming is an experimental discipline!

1. Design experiment
   - Identify data you wish to obtain.
   - Measure data for different problem sizes and/or processor numbers;
   - Be sure that you measure what you intend to measure.
2. Obtain and validate experimental data
   - Repeat experiments to verify reproducability of results.
   - Variation by nondeterministic algorithms, inaccurate timers, startup costs, interference from other programs, contention, ...
3. Fit data to analytic models.

   For instance, measure communication time and use scaled least-square fitting to determine startup and data transfer costs.