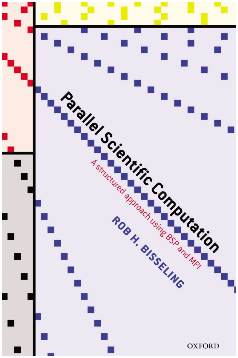
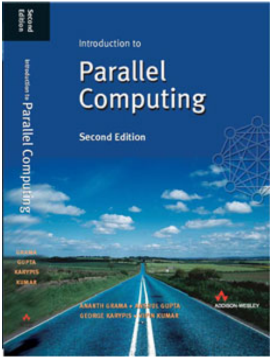
Performance metrics for parallelism

Albert-Jan Yzelman

8th of November, 2013

Sources





- Rob H. Bisseling; *Parallel Scientific Computing*, Oxford Press.
- Grama, Gupta, Karypis, Kumar; *Parallel Computing*, Addison Wesley.

Measuring performance

Definition (Parallel overhead)

- let T_{seq} be the time taken by a sequential algorithm;
- let T_p be the time taken by a parallelisation of that algorithm, using p processes.

Then, the parallel overhead T_{o} is given by

$$T_{\text{o}} = pT_p - T_s.$$

(Effort is proportional to the number of workers multiplied with the duration of their work, that is, equal to pT_p.)

Best case: $T_o = 0$, such that $T_p = T_{\text{seq}}/p$.

Measuring performance

Definition (Speedup)

Let T_{seq}, p, and T_p be as before. Then, the speedup S is given by

$$S(p) = T_{\text{seq}} / T_p.$$

Measuring performance

Definition (Speedup)

Let T_{seq}, p, and T_p be as before. Then, the speedup S is given by

$$S(p) = T_{\text{seq}} / T_p.$$

- Target: $S = p$ (no overhead; $T_o = 0$).
- Best case: $S > p$ (**superlinear speedup**).
- Worst case: $S < 1$ (slowdown).

Measuring performance

What is T_{seq}?

- Many sequential algorithms solving the same problem.

Measuring performance

What is T_{seq}?

- Many sequential algorithms solving the same problem.
- When determining the speedup S,
  compare against the best sequential algorithm
  (that is available on your architecture).

Measuring performance

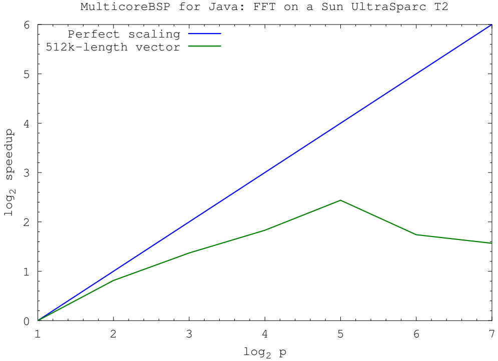
What is T_{seq}?

- Many sequential algorithms solving the same problem.
- When determining the speedup S,
  compare against the best sequential algorithm
  (that is available on your architecture).
- When determining the overhead T_o,
  compare against the most similar algorithm
  (maybe even take $T_{\text{seq}} = T_1$).

Measuring performance

Definition (strong scaling)

$$S(p) = T_{seq}/T_p = \Omega(p) \quad (\text{i.e., } \mathit{lim\ sup}_{p\to\infty}|S(p)/p| > 0)$$

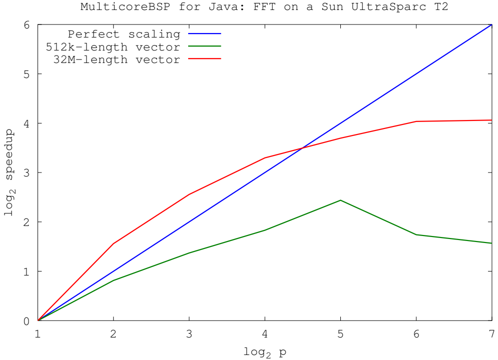


Question: is it reasonable to expect strong scalability for (good) parallel algorithms?

Measuring performance

Definition (strong scaling)

$$S(p) = T_{seq}/T_p = \Omega(p) \quad \text{(i.e., } \mathit{lim\ sup}_{p\to\infty} |S(p)/p| > 0\text{)}$$

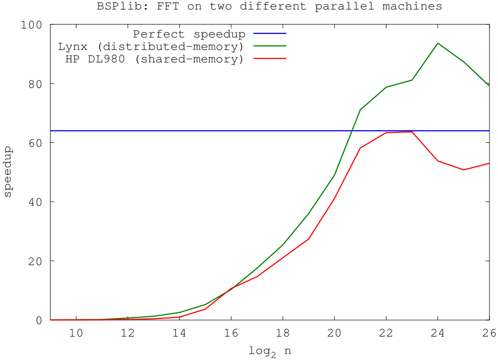


Answer: not as $p \to \infty$. You cannot efficiently clean a table with 50 people, or paint a wall with 500 painters.

Measuring performance

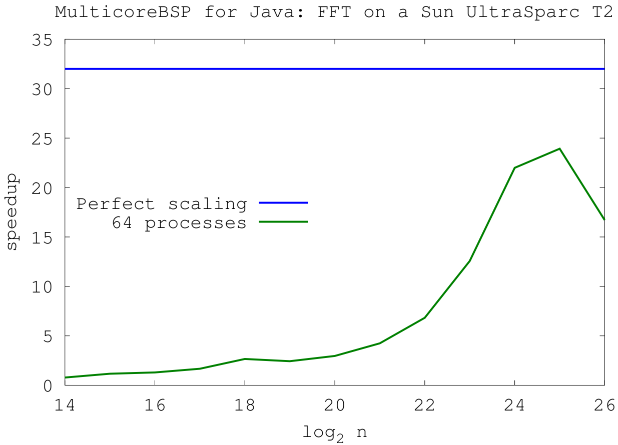
Definition (weak scaling)

$$S(n) = T_{\text{seq}}(n)/T_p(n) = \Omega(1), \text{ with } p \textbf{ fixed.}$$



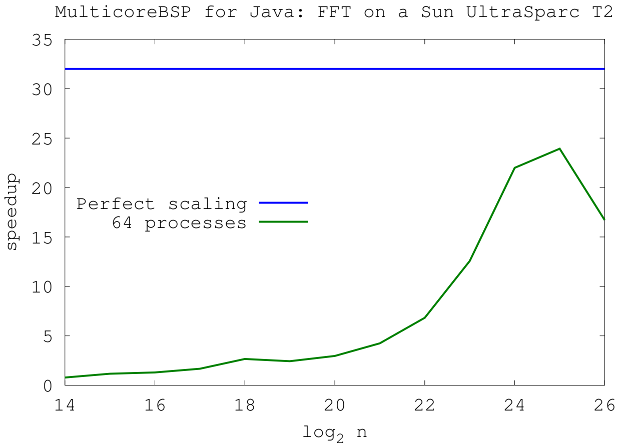
For large enough problems, we do expect to make maximum use of our parallel computer.

Measuring performance: example



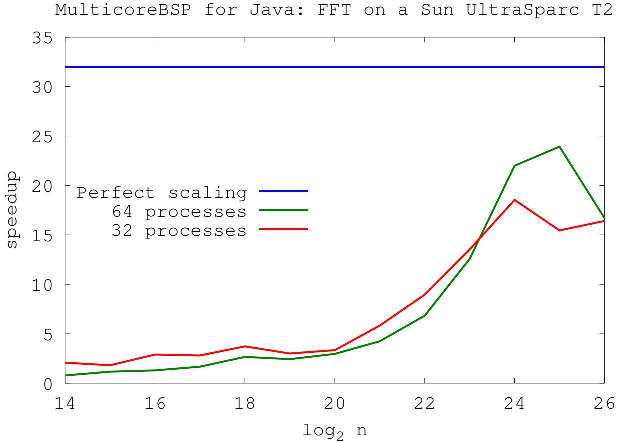
- This processor advertises 64 processors,
-

Measuring performance: example



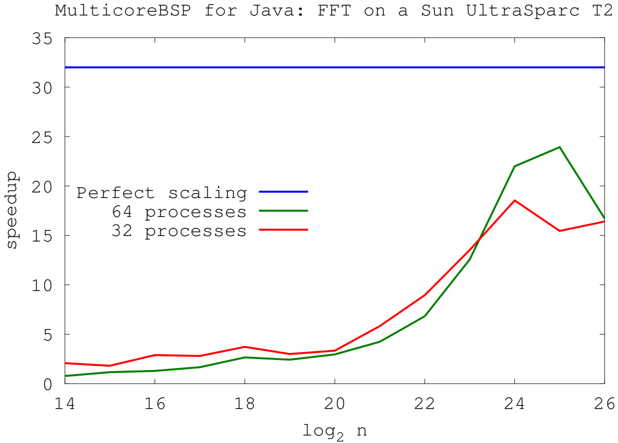
- This processor advertises 64 processors, but only has 32 FPUs.
- We oversubscribed!

Measuring performance



- This processor advertises 64 processors, but only has 32 FPUs.
- Be careful with oversubscription! (**Including hyperthreading**!)

Measuring performance



- This processor advertises 64 processors, but only has 32 FPUs.
- **Q**: would you say this algorithm scales on the Sun Ultrasparc T2?

Measuring performance

A: if the speedup stabilises around 16x, then

yes,

since the relative efficiency is stable.

Definition (Parallel efficiency)

Let T_{seq}, p, T_p, and S as before. The parallel efficiency E equals

$$E = \frac{T_{\text{seq}}}{p} / T_p = T_{\text{seq}} / pT_p = S/p.$$

Measuring performance

- If there is no overhead ($T_o = pT_p - T_{\text{seq}} = 0$), the efficiency $E = 1$; decreasing the overhead increases the efficiency.

Weak scalability: what happens if the problem size increases?

Measuring performance

- If there is no overhead ($T_o = pT_p - T_{\text{seq}} = 0$), the efficiency $E = 1$; decreasing the overhead increases the efficiency.

Weak scalability: what happens if the problem size increases?

But what is a sensible definition of the 'problem size'?

Consider the following applications:

- inner-product calculation;
- binary search;
- sorting (quicksort).

Measuring performance

Problem	Size	Run-time
Inner-product	$\Theta(n)$ bytes	$\Theta(n)$ flops
Binary search	$\Theta(n)$ bytes	$\Theta(\log_2 n)$ comparisons
Sorting	$\Theta(n)$ bytes	$\Theta(n \log_2 n)$ swaps
FFT	$\Theta(n)$ bytes	$\Theta(n \log_2 n)$ flops

Hence the problem size is best identified by T_{seq}.

Question:

- How should the ratio T_o / T_{seq} behave as $T_{\text{seq}} \to \infty$, for the algorithm to scale in a weak sense?

Measuring performance

If $T_o / T_{\text{seq}} = c$, with $c \in \mathbb{R}_{\geq 0}$ constant, then

$$\frac{pT_p - T_{\text{seq}}}{T_{\text{seq}}} = pS^{-1} - 1 = c, \text{ so}$$

$$S = \frac{p}{c+1}, \text{ which is constant when } p \text{ is fixed.}$$

Note that here, $E = S/p = \frac{1}{c+1}$

Question:

- How should the ratio T_o / T_{seq} behave as $p \to \infty$, for the algorithm to scale in a strong sense?

Measuring performance

If $T_o / T_{\text{seq}} = c$, with $c \in \mathbb{R}_{\geq 0}$ constant, then

$$\frac{pT_p - T_{\text{seq}}}{T_{\text{seq}}} = pS^{-1} - 1 = c, \text{ so}$$

$$S = \frac{p}{c+1}.$$

Note that here, $E = S/p = \frac{1}{c+1}$ which is still constant!

Answer:

- Exactly the same! Both strong and weak scalability are **iso-efficiency** constraints (E remains constant).

Measuring performance

Definition (iso-efficiency)

Let E be as before. Suppose $T_o = f(T_{\text{seq}}, p)$ is a known function. Then the iso-efficiency relation is given by

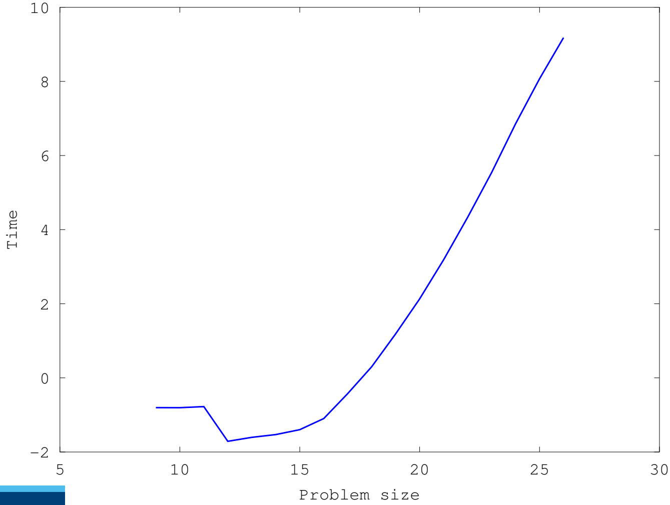
$$T_{\text{seq}} = \frac{1}{\frac{1}{E} - 1} f(T_{\text{seq}}, p).$$

This follows from the definition of E:

$$\begin{aligned} E^{-1} &= pT_p / T_{\text{seq}} + 1 - \frac{T_{\text{seq}}}{T_{\text{seq}}} \\ &= 1 + \frac{pT_p - T_{\text{seq}}}{T_{\text{seq}}} \\ &= 1 + \frac{T_o}{T_{\text{seq}}}, \qquad \text{so } T_{\text{seq}}(E^{-1} - 1) = T_o. \end{aligned}$$

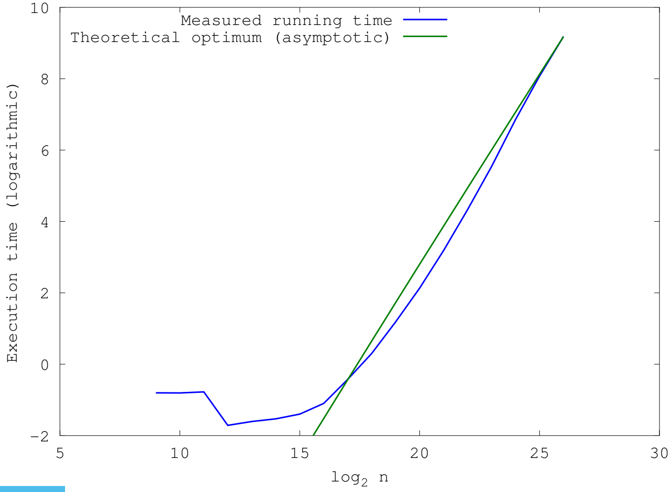
Questions

- Does this scale?



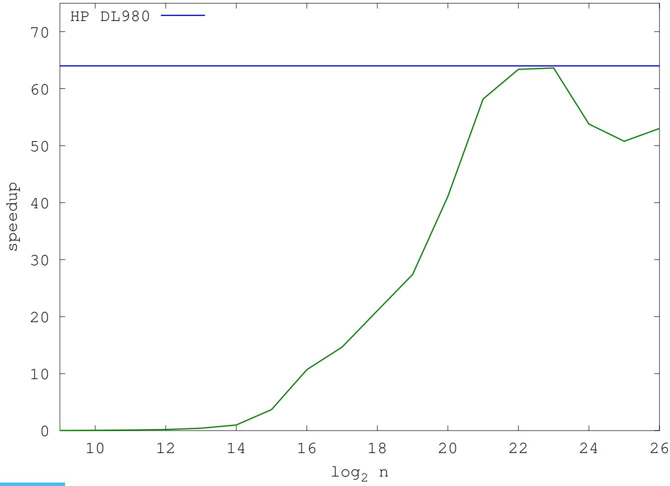
Questions

- Does this scale? **Yes**! (It's again an FFT)



Questions

- Better use speedups when investigating scalability.



In summary...

A BSP algorithm is **scalable** when

$$T = \mathcal{O}(T_{\text{seq}}/p + p).$$

This considers scalability of the speedup and includes parallel overhead.

In summary...

A BSP algorithm is **scalable** when

$$T = \mathcal{O}(T_{\text{seq}}/p + p).$$

This considers scalability of the speedup and includes parallel overhead. It does not include **memory scalability**:

$$M = \mathcal{O}(M_{\text{seq}}/p + p),$$

where M is the memory taken by one BSP process and M_{seq} the memory requirement of the best sequential algorithm.

Question:

does this definition of a scalable algorithm make sense?

In summary...

It does indeed make sense:

$$\begin{aligned} E^{-1} &= \frac{pT_p}{T_s} \\ &= 1 + \frac{pT_p - T_s}{T_s} \\ &= 1 + \frac{T_o}{T_s}. \end{aligned}$$

If $T_o = \Theta(p)$, then

$$E = \frac{1}{1 + p/T_s} = \frac{T_s}{T_s + p}.$$

Note $\lim_{p\to\infty} E = 0$ (strong scalability, Amdahl's Law) and $\lim_{T_s\to\infty} = 1$ (weak scalability). For iso-efficiency, the ratio T_o and p has to change appropriately.

Practical session

Coming Thursday, see `http://people.cs.kuleuven.be/~albert-jan.yzelman/education/parco13/`

Subjects:

1. Amdahl's Law
2. Parallel Sorting
3. Parallel Grid computations

Insights from the practical session

This intuition leads to a definition of how 'parallel' certain algorithms are:

Definition (Parallelism)

Consider a parallel algorithm that runs in T_p time. Let T_{seq} the time taken by the best sequential algorithm that solves the same problem. Then the **parallelism** is given by

$$\frac{T_{\text{seq}}}{T_\infty} = \lim_{p \to \infty} \frac{T_{\text{seq}}}{T_p}.$$

This kind of analysis is fundamental for fine-grained parallelisation schemes.

- Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. 1995. Cilk: an efficient multithreaded runtime system. SIGPLAN Not. 30, 8 (August 1995), pp. 207-216.