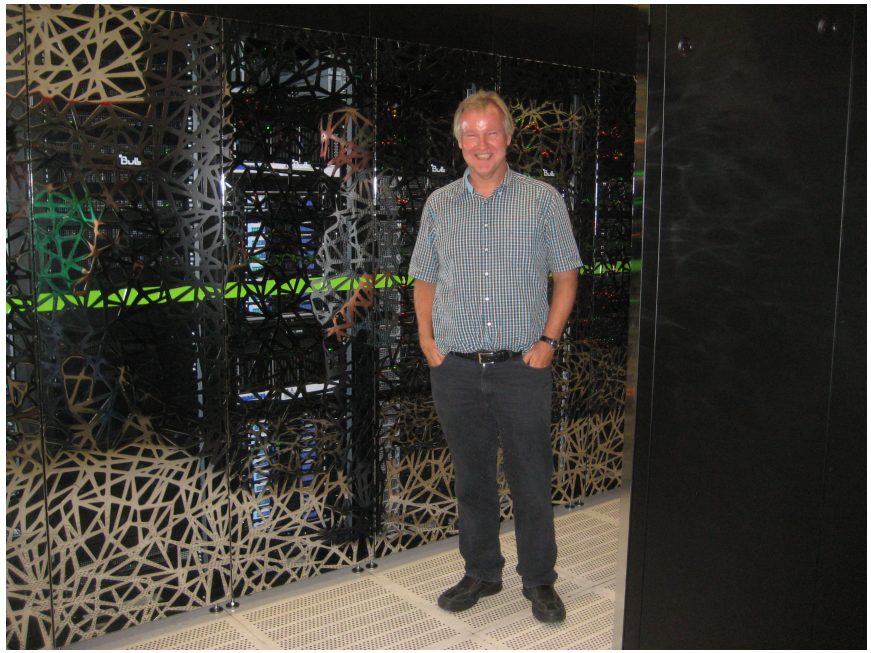
Experimental results for the FFT

Section 3.7 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

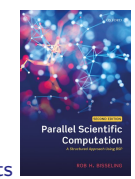
Utrecht University

The Cartesius supercomputer



Rob Bisseling dwarfed by Cartesius, October 2013

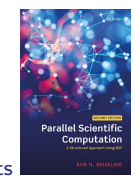
- The Dutch national supercomputer Cartesius was installed at SURFsara in Amsterdam, the Netherlands, in June 2013.
- YouTube movie of installation:
  `https://www.youtube.com/watch?v=01buSTm9p4Q`



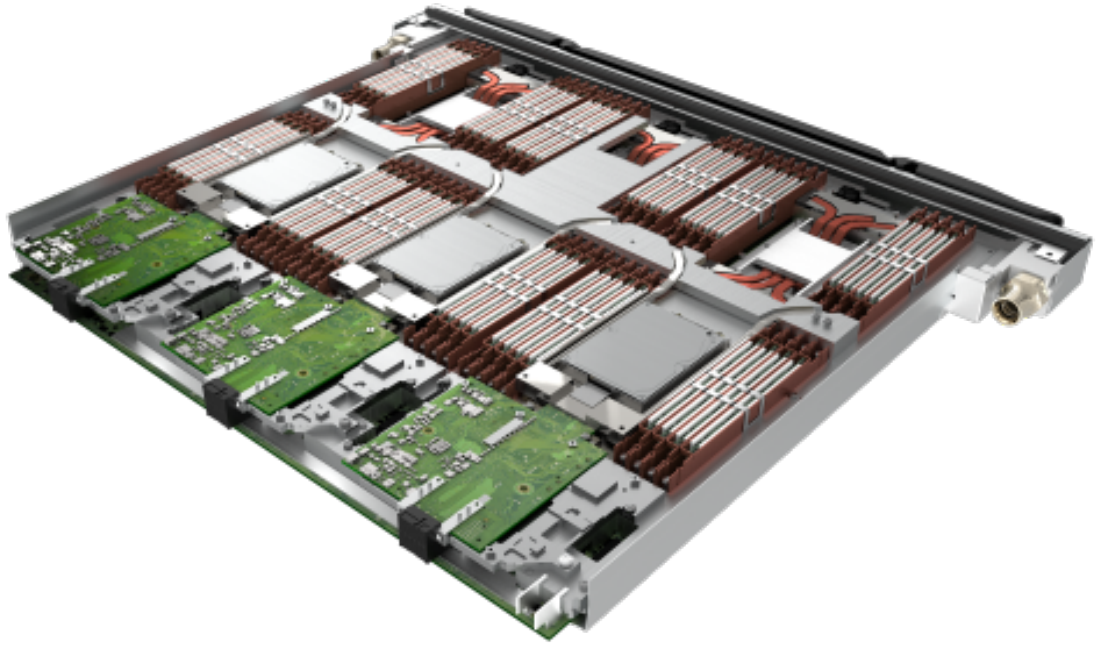
The BullSequana cell added to Cartesius in 2016

BullSequana cell. Source: `https://tweakers.net`

- A BullSequana cell with two cabinets, each containing up to 24 blades at the front and 24 at the back, and a cabinet in the middle with cables and switching devices for communication.
- YouTube movie of extension: `https://www.youtube.com/watch?v=p9imWg4pHeI`

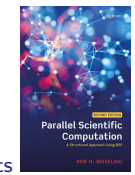


One blade of the BullSequana cell



BullSequana X1120 blade. Source: `https://atos.net`

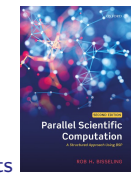
- One BullSequana blade contains three thin nodes.
- In total, Cartesius has 177 thin nodes of type Broadwell.
- Each thin node has two 16-core 2.6 GHz Intel Xeon E5-2697A v4 (Broadwell) CPUs and a memory of 64 GB.



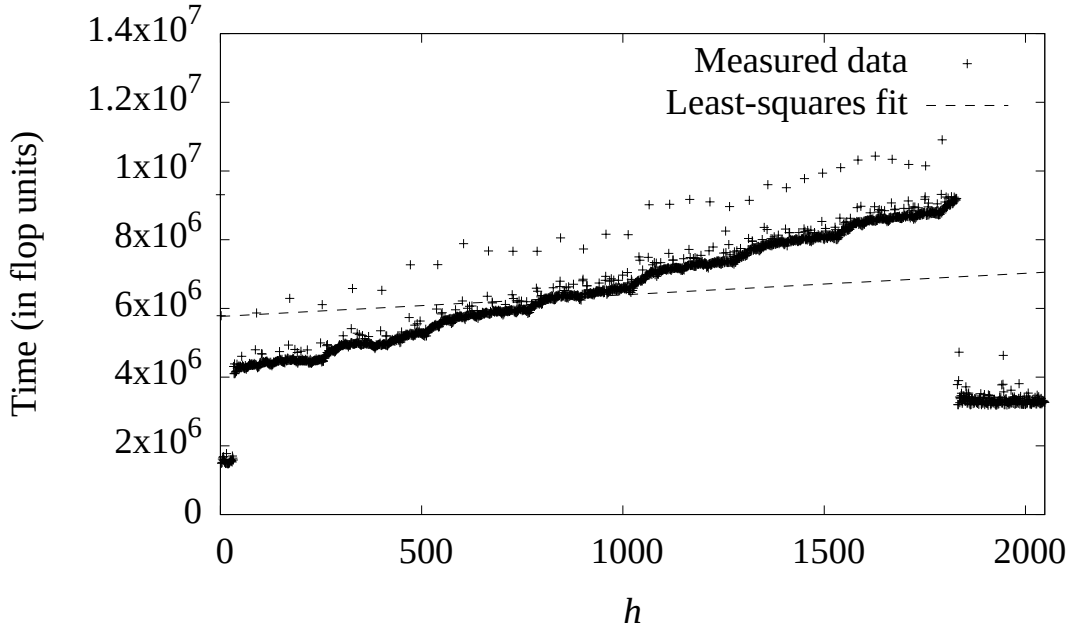
Benchmarked BSP parameters of a BullSequana cell

p	g	l	$T_{\mathrm{comm}}(0)$
1	315	25 999	16 270
2	352	63 146	47 270
4	333	122 716	96 964
8	328	216 380	146 945
16	350	330 632	221 014
32	450	469 605	312 097
64	500	2 437 872	1 733 434
128	1 800	2 646 273	2 647 049
256	756	5 045 305	4 310 730
512	1 114	4 098 700	3 078 612

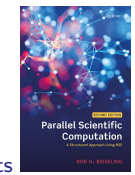
- $r = 5.912$ Gflop/s.
- $T_{\mathrm{comm}}(0)$ is the time of a 0-relation.
- BSPonMPI communication library v1.1 on top of Intel MPI v5.0.3.



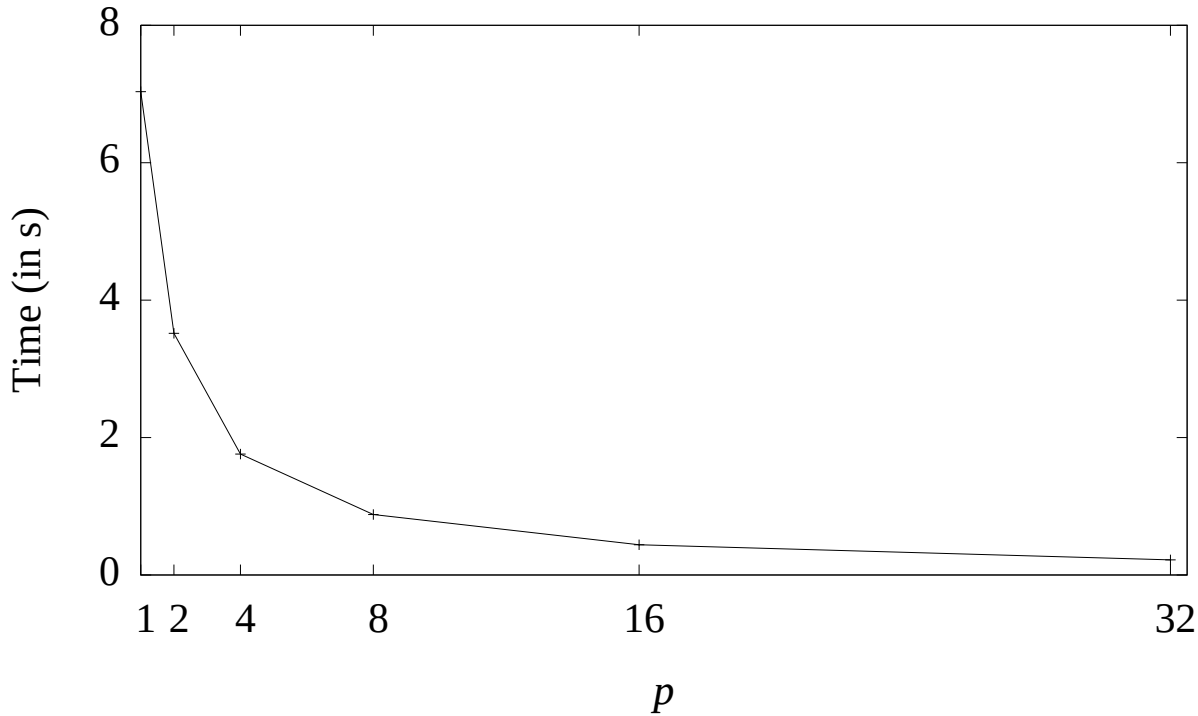
Time of an h-relation with `MPI_Alltoallv` for $p = 256$



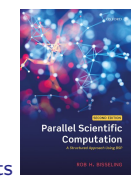
- We investigated the inconsistent behaviour of the parameter g in the range $p = 64$–512.
- Using `mpibench` from the first edition, we observe two straight lines with a downwards jump indicating a suboptimal switching point between two communication mechanisms.
- Therefore, the culprit is most likely MPI.



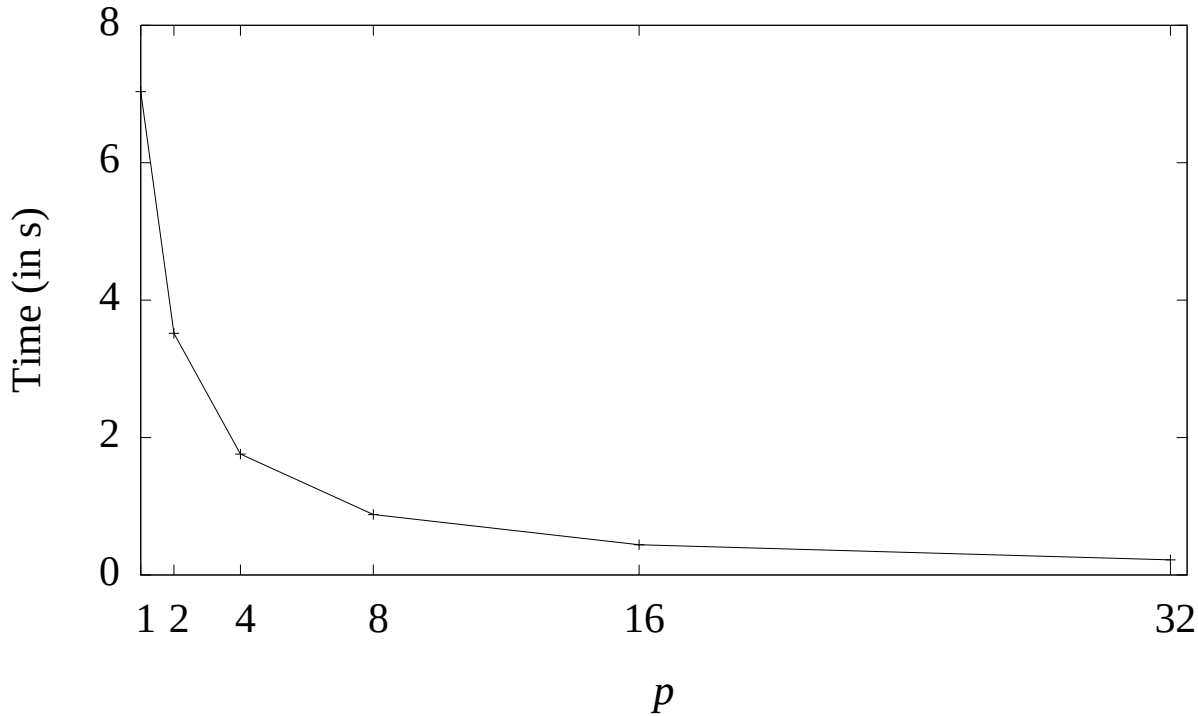
Time of a parallel FFT of length $n = 2^{26}$



What do you think? Good or bad?

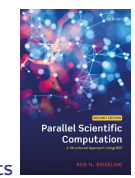


Time of a parallel FFT of length $n = 2^{26}$



What do you think? Good or bad?

Surprise! This is the time of a theoretical, perfectly parallelized FFT, based on a time of 7.035 s for $p = 1$.

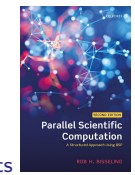


Measured time $T_p(n)$ (in ms) of parallel FFT

p		Length n					
		2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}
1	(seq)	2.6	13.9	67.6	356	1 603	7 035
1	(par)	2.6	13.5	65.9	372	1 603	7 017
2		1.6	6.9	37.3	177	927	3 972
4		0.9	3.5	18.9	90	510	2 160
8		0.4	1.9	9.2	48	278	1 238
16		0.2	1.0	4.4	25	195	842
32		0.2	0.6	2.5	16	172	743
64		4.3	4.5	5.5	12	85	382
128		0.8	12.8	13.0	17	36	214
256		1.1	1.3	32.1	33	45	147
512		1.4	0.8	1.3	72	76	107

blue: problem fits in L2 cache

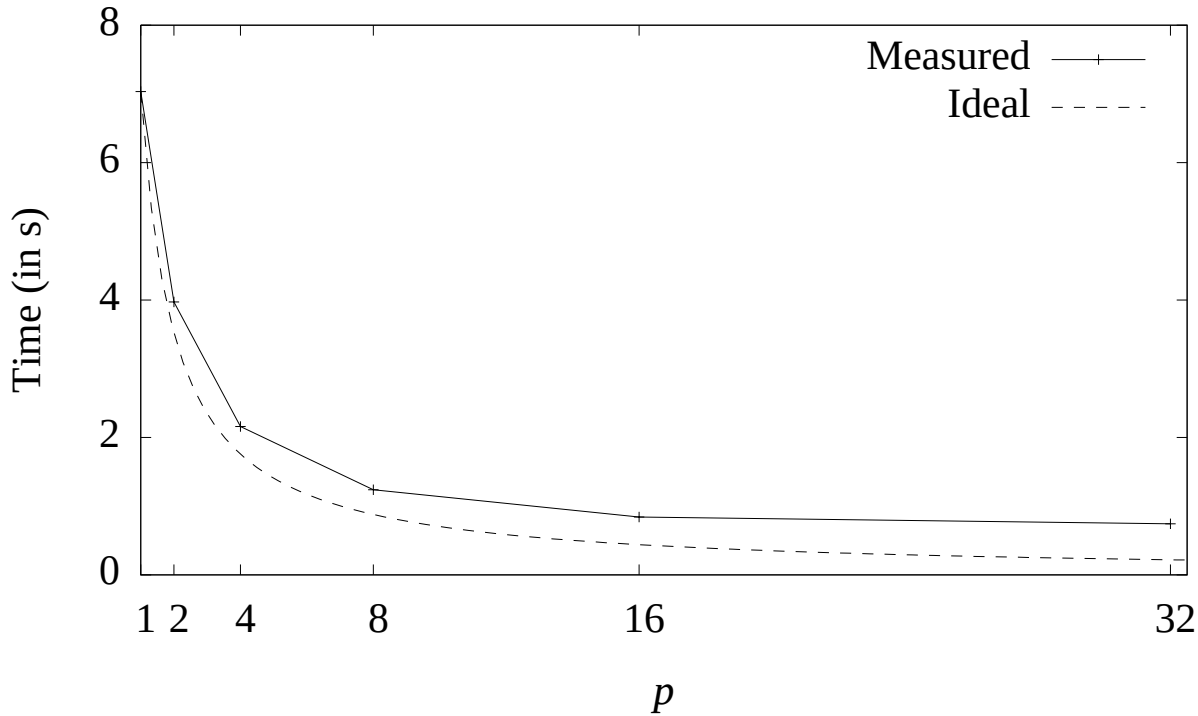
red: problem fits in L1 cache (fastest, smallest)



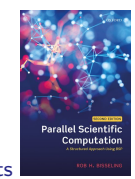
Competing trends with an increase in p

- Computation time decreases because more processors do the work.
- Computation time may also decrease because we have more caches, so that the local problems may fit into cache.
- Communication time increases because $g = g(p)$ increases.
- Synchronization time increases because $l = l(p)$ increases.
- The measured time is a result of all these competing trends.

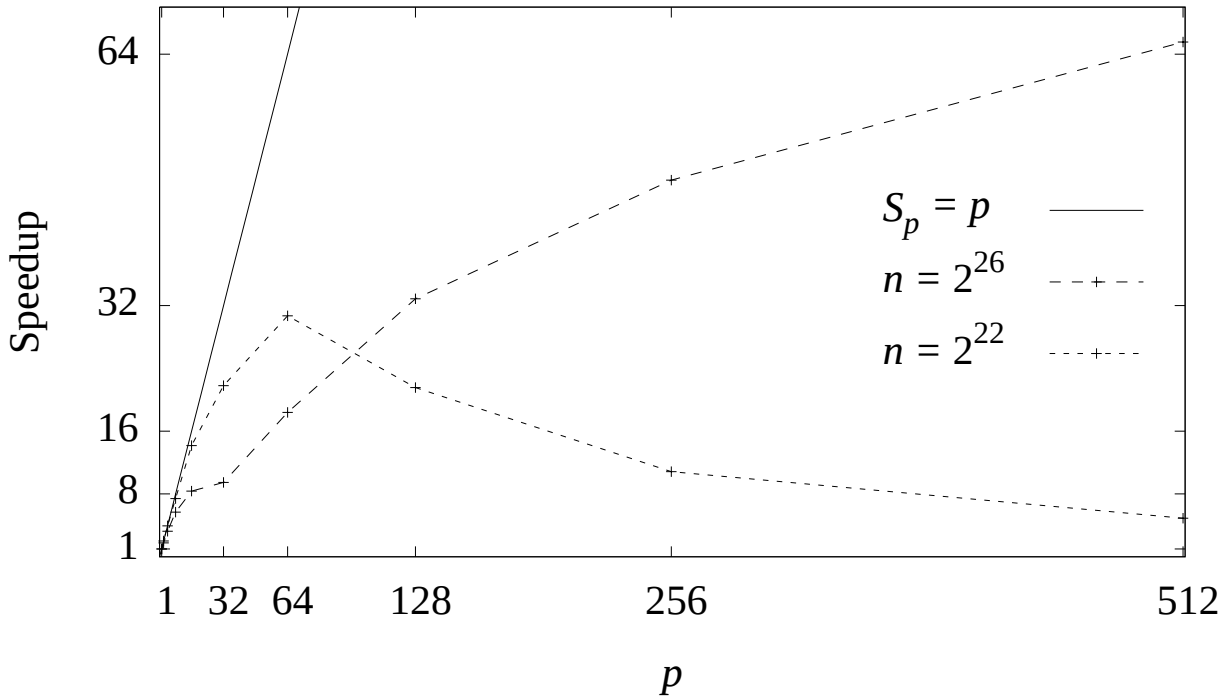
Time T_p of an actual parallel FFT of length $n = 2^{26}$



Warning: this kind of picture gives some insight, but it is not the best representation of the results.



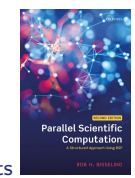
Measured speedup $S_p(n)$ of parallel FFT



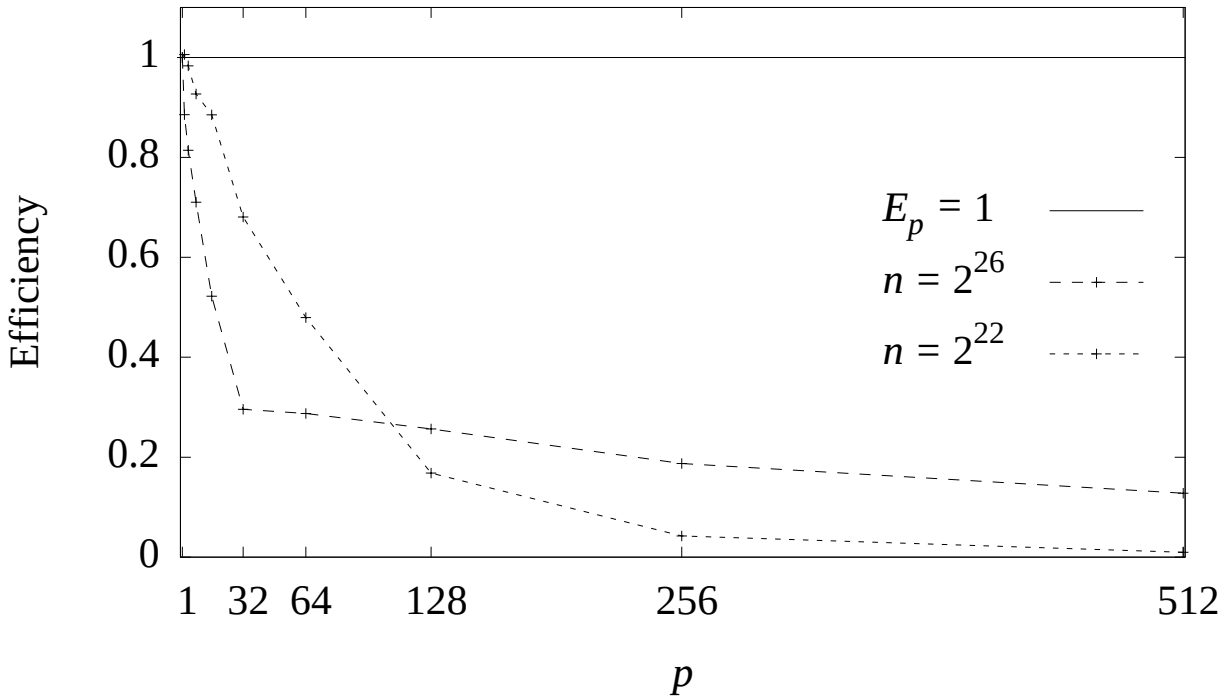
- The speedup on p processors for a problem of length n is defined by

$$S_p(n) = \frac{T_{\text{seq}}(n)}{T_p(n)}.$$

- A speedup picture gives much more insight and it allows comparing different problem sizes.



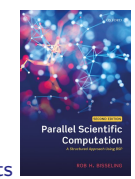
Measured efficiency $E_p(n)$ of parallel FFT



- The efficiency on p processors is defined by

$$E_p(n) = \frac{S_p(n)}{p} = \frac{T_{\text{seq}}(n)}{pT_p(n)}.$$

- The ideal value is 1.



Normalized cost

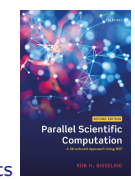
- The normalized cost (or inefficiency) $C_p(n)$ is the ratio between the time of the parallel program and the time of a perfectly parallelized version of the sequential program,

$$C_p(n) = \frac{T_p(n)}{T_{\text{seq}}(n)/p} = \frac{pT_p(n)}{T_{\text{seq}}(n)} = \frac{1}{E_p(n)}.$$

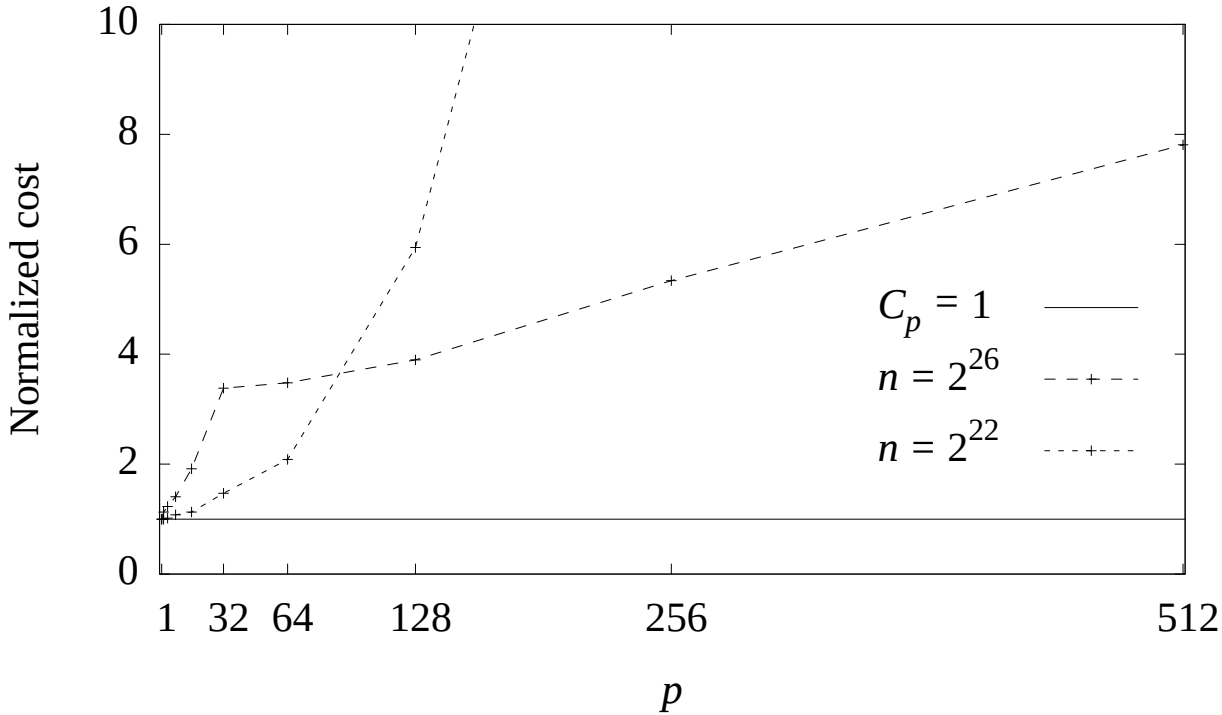
- Lower bound on the inefficiency:

$$C_p(n) \geq 1.$$

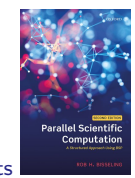
- The parallel overhead equals $C_p(n) - 1$. It usually consists of:
  - load imbalance,
  - communication time,
  - synchronization time.



Normalized cost $C_p(n)$ of parallel FFT



- The ideal value is 1.

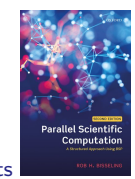


Breakdown of predicted and measured execution time

p	Predicted				Measured		
	T_{comp}	T_{comm}	T_{sync}	T	T_{comp}	T_{comm} $+T_{\text{sync}}$	T
1	1 476	0	0.004	1 476	7 048	0	7 048
2	738	4 004	0.032	4 742	3 548	593	4 141
4	369	1 882	0.062	2 251	1 788	428	2 223
8	184	925	0.109	1 109	942	315	1 268
16	92	492	0.167	584	640	209	852
32	46	319	0.238	365	601	154	757

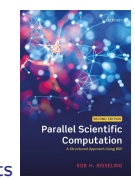
- Time (in ms) for $n = 2^{26}$. T is the total time.
- Prediction is based on

$$T_p(n) = 5\frac{n}{p}\log_2 n + 2\frac{n}{p}g + 3l.$$



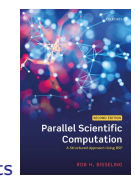
Insights gained from the breakdown

- It is difficult to predict the total time correctly, for instance due to misprediction of the sequential computation time.
- The DAXPY benchmark of length $n = 2^{10}$ fits in L1 cache, but the FFT of length $n = 2^{26}$ does not fit in any cache.
- This reduces the computing rate from $r = 5.91$ Gflop/s to 1.26 Gflop/s.
- Of course, the benchmark of r can be adapted to the application, if desired.



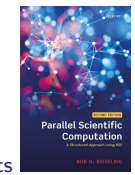
More insights

- Communication is the bottleneck, even though we perform only one data permutation.
- The prediction overestimates the communication time, because it is based on pessimistic g-values.
- The actual parallel FFT, however, was optimized to send data in packets, so that it attains optimistic g-values.
- Synchronization time is insignificant for this problem size.



"Predictions are difficult, especially about the future"

- Wisdom often attributed to Danish quantum physicist Niels Bohr and sometimes to US baseball player Yogi Berra, but never to Dutch soccer player Johan Cruijff.
- The quote probably was first expressed by an anonymous person in Danish in or before 1948, according to `https://quoteinvestigator.com`
- We can improve our predictions by using more specific measurements: in-cache flop rates, optimistic g-values, etc.
- This way, we end up predicting the past.

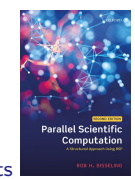


Total computing rate $R_p(n)$

- The total computing rate of the FFT is defined by

$$R_p(n) = \frac{5n \log_2 n}{T_p(n)}.$$

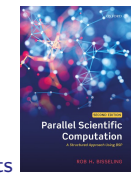
- The rate is based on the sequential flop count $5n \log_2 n$. This count is commonly used to measure FFT rates, even for FFT variants with fewer actual flops.
- Radix-4 FFTs have $4.25n \log_2 n$ flops.



Computing rate $R_p(n)$ (in Gflop/s) of parallel FFT

p		Length n					
		2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}
1	(seq)	2.04	1.74	1.58	1.32	1.28	1.26
1	(par)	2.10	1.79	1.62	1.26	1.28	1.26
2		3.32	3.47	2.87	2.66	2.21	2.23
4		6.28	6.83	5.65	5.19	4.01	4.10
8		12.64	12.70	11.59	9.79	7.37	7.16
16		21.65	24.90	24.37	18.70	10.47	10.52
32		27.48	41.63	42.23	28.76	11.90	11.92
64		1.24	5.42	19.44	40.52	24.15	23.16
128		6.96	1.89	8.22	28.45	56.76	41.36
256		5.02	18.94	3.33	14.33	45.74	60.40
512		3.78	28.44	83.85	6.49	27.05	82.54

- Computing rate $R_p(n)$ is useful for comparing different problem sizes and different applications.



Summary

- We have introduced several metrics to express the performance of a parallel program:
  - $T_p(n)$, the time (in s)
  - $S_p(n) = T_{\text{seq}}(n)/T_p(n)$, the speedup
  - $E_p(n) = S_p(n)/p$, the efficiency
  - $C_p(n) = 1/E_p(n)$, the normalized cost or inefficiency
  - $C_p(n) - 1$, the overhead
  - $R_p(n) = (5n \log_2 n)/T_p(n)$, the total computing rate (in flop/s).
- Speedup plots give much insight.
- Cache effects play a major role for the FFT.
- Parallelizing the FFT only makes sense for large problem sizes.