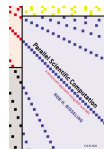


Experimental results for the FFT **(PSC §3.7)**



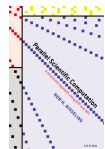
Test computer: SGI Origin 3800



Photo: Walter de Jong

<http://www.xs4all.nl/~walterj/sara>

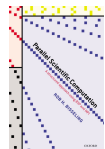
- Teras, the national supercomputer in the Netherlands, located in Amsterdam. Installed in 2000; overtaken by an SGI Altix 3700 (Aster) in 2003. **Machines come and go.**
- Named after Teraflop/s computing rate (10^{12} flop/s) and after the Greek word for 'monster', *τερας*.
- 1024 processors, split into 6 partitions with 512, 256, 128, 64, 32, 32 processors.



SGI Origin 3800 is a CC-NUMA machine



- Each processor has:
 - MIPS RS14000 CPU with a clock rate of 500 MHz and a theoretical peak performance of 1 Gflop/s
 - primary data cache of 32 Kbyte
 - secondary cache of 8 Mbyte
 - memory of 1 Gbyte.
- Cache Coherent Non-Uniform Memory Access:
 - cache is kept coherent, so user views a shared memory
 - physically, the memory is distributed; hence, access time to local and remote memory differs

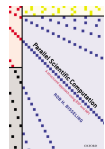


Benchmarked BSP parameters of SGI Origin 3800

p	g	l	$T_{\text{comm}}(0)$
1	99	55	378
2	75	5118	1414
4	99	12743	2098
8	126	32742	4947
16	122	93488	15766

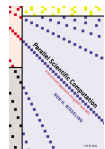
$r = 285$ Mflop/s.

$T_{\text{comm}}(0)$ is the time of a 0-relation.

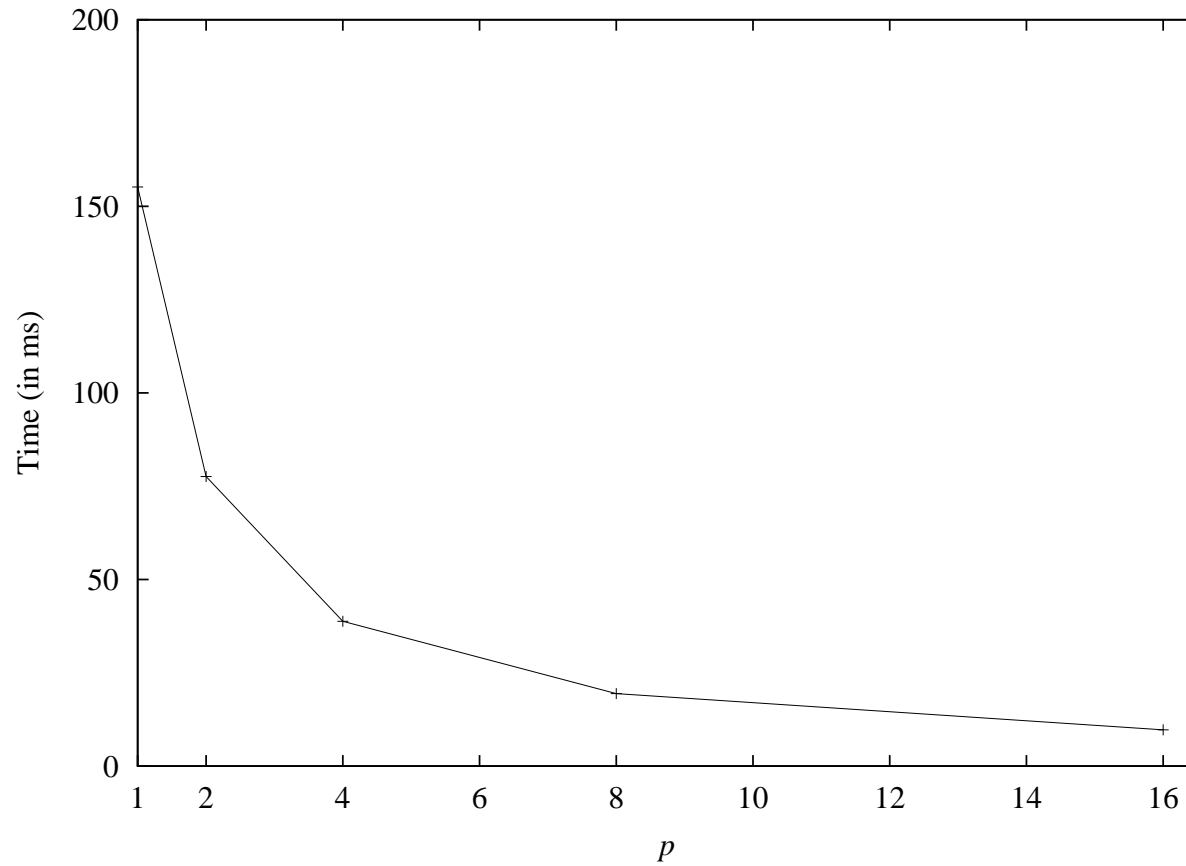


Aggressive optimisation

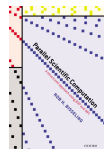
- Initial tests: maximum optimisation level -O3 for newly installed C compiler gave benchmark rate 981 Mflop/s.
- This is almost the theoretical peak rate. For a DAXPY, such a speed is impossible.
- The new compiler discovered our true intention of just measuring the computing rate, and cleverly removed some unnecessary statements.
- We reduced the optimisation level for benchmarking to -O2.
- We may have been fooled before (predecessor Origin 2000, Chapter 1), with a measured rate of 326 Mflop/s. This high rate is partly due to having the machine to ourselves, but perhaps also to overly aggressive optimisation.
- Always be cautious about results!



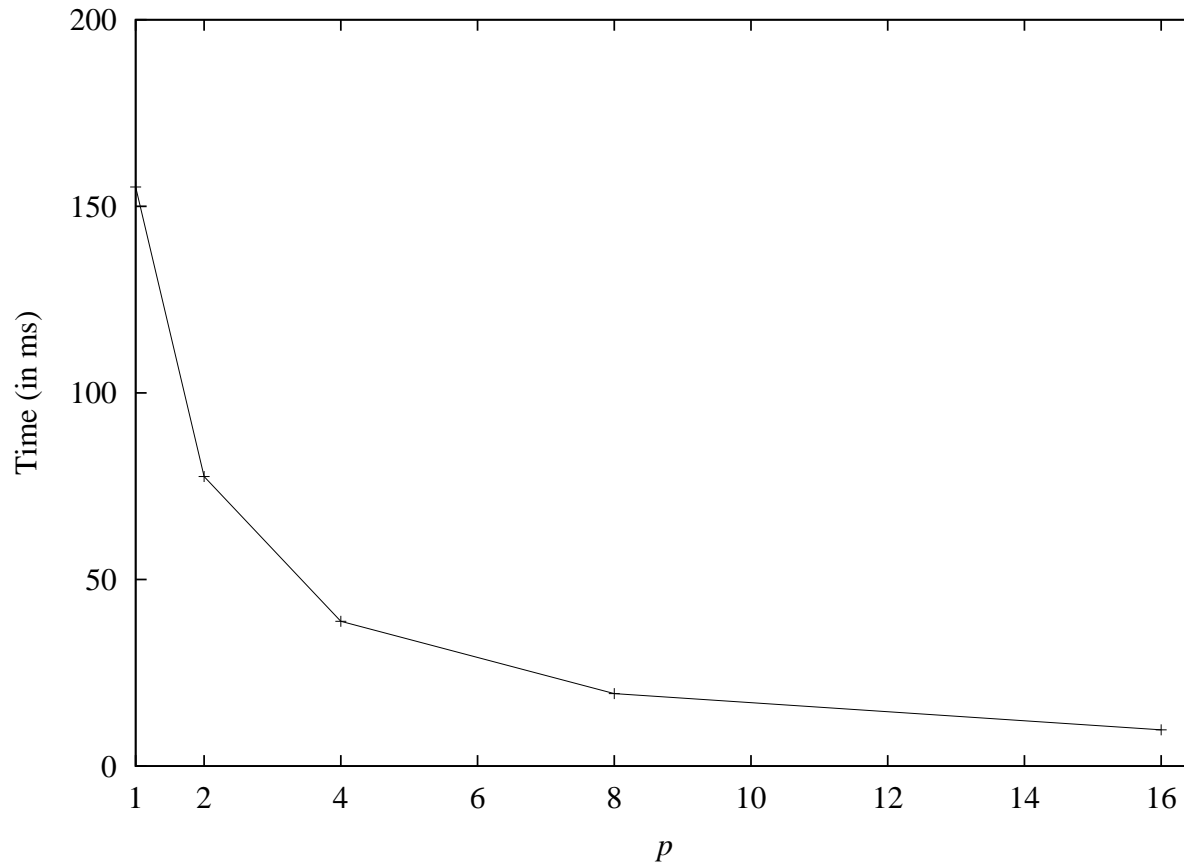
Time of a parallel FFT of length 262144



What do you think? Good or bad?

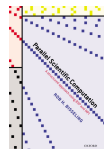


Time of a parallel FFT of length 262144



What do you think? Good or bad?

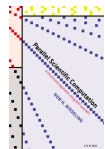
Surprise! This is the time of a theoretical, perfectly parallelised FFT, based on a time of 155.2 ms for $p = 1$.



Measured time $T_p(n)$ of sequential and parallel FFT

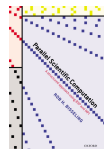
		Length n			
p		4096	16384	65536	262144
1	(seq)	1.16	5.99	26.6	155.2
1	(par)	1.32	6.58	29.8	167.4
2		1.06	4.92	22.9	99.4
4		0.64	3.15	13.6	52.2
8		1.18	2.00	8.9	29.3
16		8.44	11.07	9.9	26.8

Time in ms.

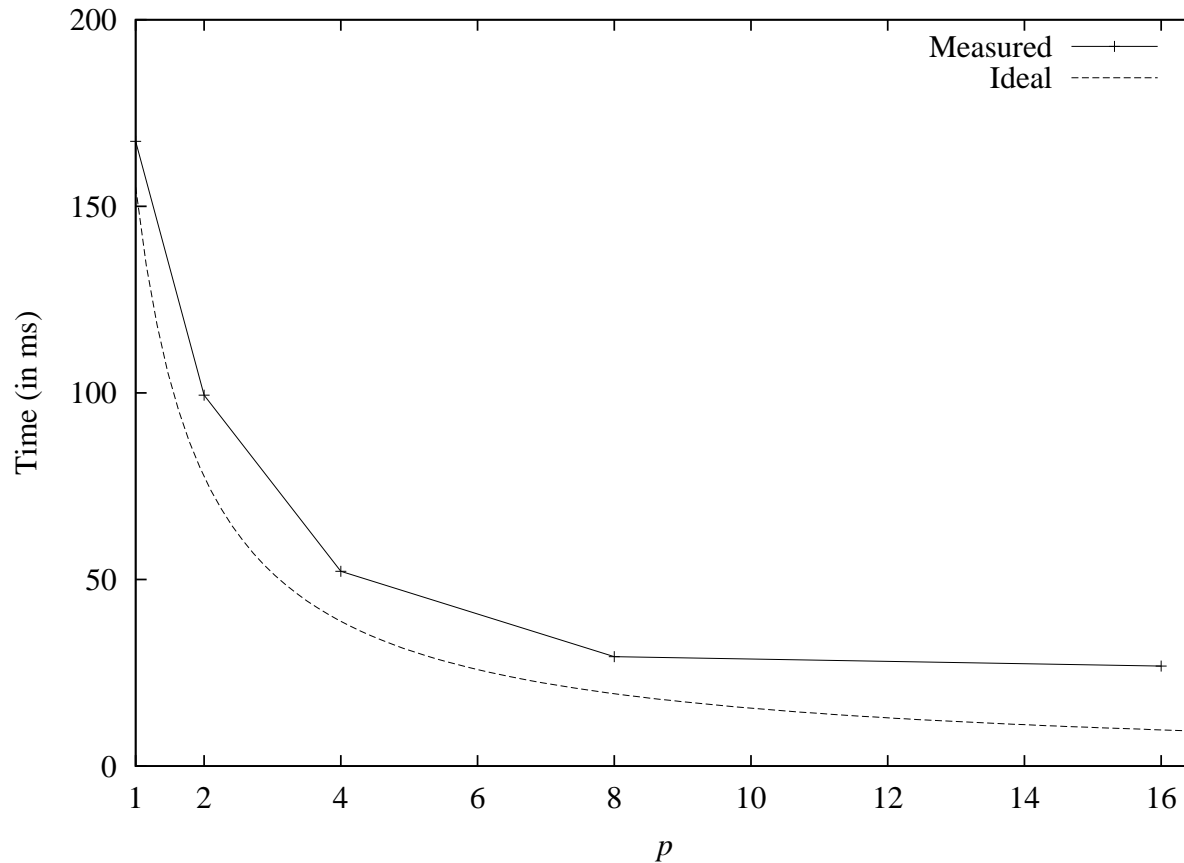


Time measurements are difficult on the Origin

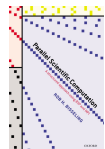
- Timings may suffer from **interference** by other programs (caused e.g. by sharing of communication links).
- **Best of three**: we run each experiment 3 times, and take the best result.
- Often, the best two timings are within 5% of each other, and the third result is worse.



Time T_p of actual parallel FFT of length 262144



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

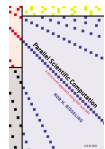


Speedup

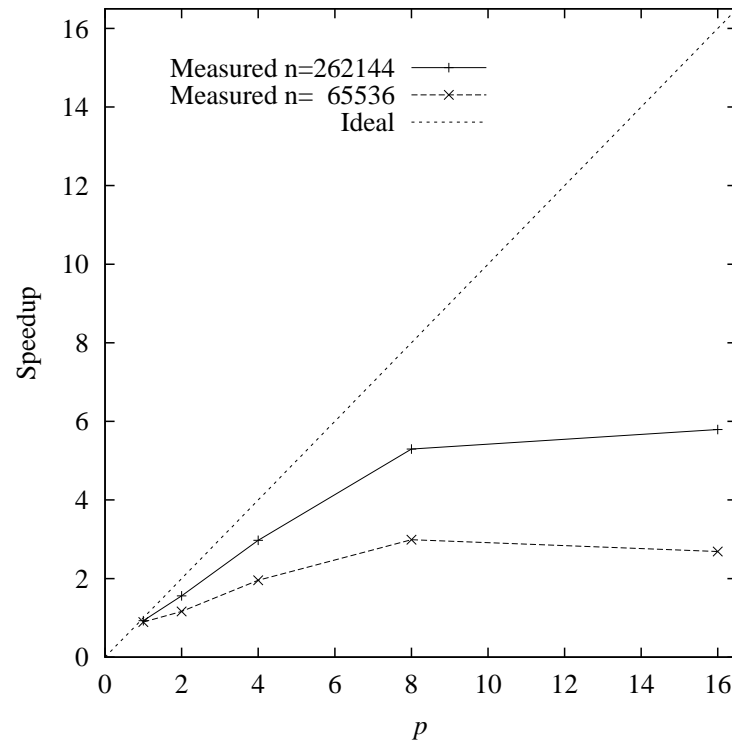
- The **speedup** $S_p(n)$ of a parallel program is the increase in speed of the program running on p processors compared to the speed of a sequential program with the same level of optimisation,

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

- Do not compare with T_1 instead of T_{seq} , since this may be **too flattering**. The parallel program run with $p = 1$ may have much overhead. Here: 8%.
- Often, it is easy to simplify a parallel program into a sequential one by removing overhead.
- If this is too much work, then be at least clear about the reference **'sequential'** program.

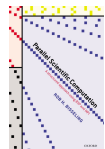


Speedup $S_p(n)$ of parallel FFT



This kind of picture gives much more insight.

It allows comparison for different problem sizes.

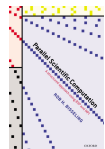


Superlinear speedup

- Bound on speedup:

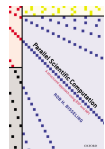
$$0 \leq S_p(n) \leq p.$$

- $S_p(n) < 1$ is called a **slowdown**. It usually happens for $p = 1$, and sometimes for $p = 2$.
- $S_p(n) > p$ is called **superlinear speedup**. In theory, this cannot happen, but in practice it does. Possible causes:
 - **Cache effects**: in the parallel case, each processor has less data to handle than in the sequential case, so that the local data may fit in the cache.
 - Different order of the computations, giving **less work** in the parallel case. Example: search algorithms, where the search stops when one processor finds a solution. (Trick often used in demos by parallel computer vendors.)



Superlinear speedup: blessing or curse?

- Effects that cause superlinear speedups make it difficult to judge the **quality of the parallelisation**. Even if no actual superlinear speedups are observed ...
- Still, a faster computation is always welcome. Besides, you paid for the **multiple caches** of a parallel computer.



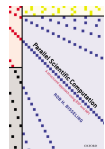
Efficiency

- The **efficiency** $E_p(n)$ of a parallel program is the fraction of the total computing power that is usefully employed. It is defined by

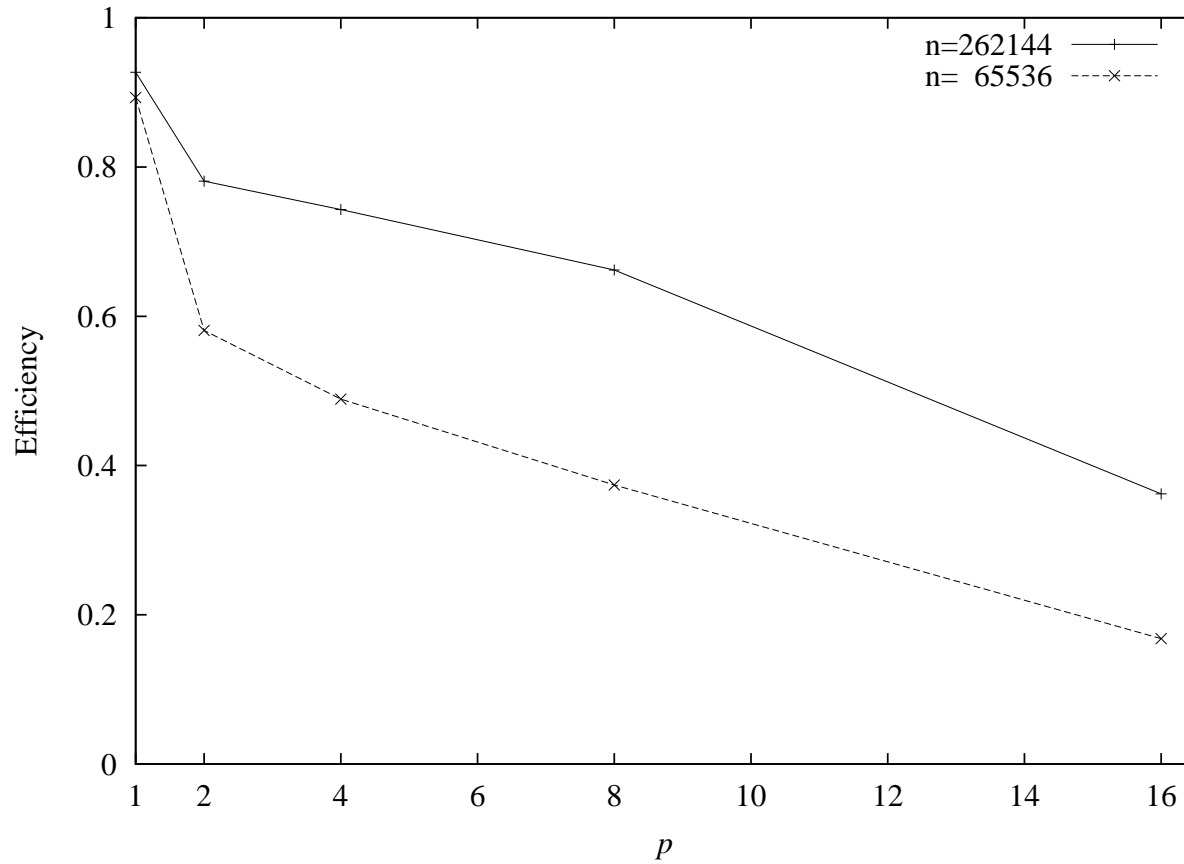
$$E_p(n) = \frac{S_p(n)}{p} = \frac{T_{\text{seq}}(n)}{pT_p(n)}.$$

- Bound on efficiency:

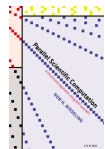
$$0 \leq E_p(n) \leq 1.$$



Measured efficiency $E_p(n)$ of parallel FFT



The ideal value is 1.

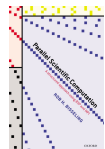


Inefficiency

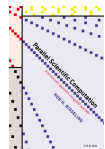
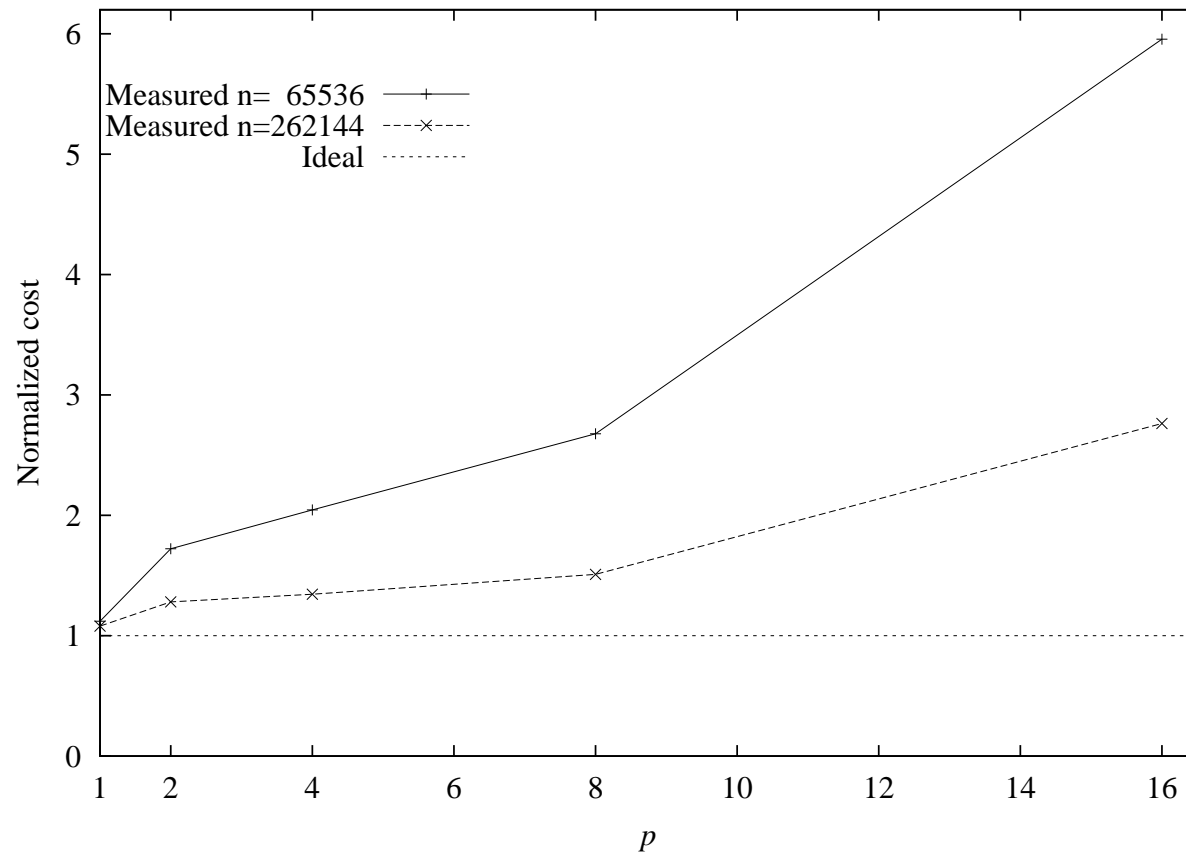
- The **normalised cost** (or **inefficiency**) $C_p(n)$ is the ratio between the time of the parallel program and the time of a perfectly parallelised version of the sequential program. It is defined by

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

- 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
 - synchronisation time



Normalised cost $C_p(n)$ of parallel FFT



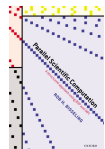
Breakdown of predicted execution time

p	T_{Comp}	T_{Comm}	T_{Sync}	T_{FFT} (pred.)	T_p (meas.)
1	82.78	0.00	0.00	82.78	167.4
2	41.39	68.99	0.05	110.43	99.4
4	20.70	45.53	0.13	66.36	52.2
8	10.35	28.97	0.35	39.67	29.3
16	5.17	14.03	0.98	20.18	26.8

Time in ms. $n = 262144$.

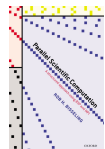
Prediction is based on time

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



Insights gained from breakdown

- It is difficult to predict the total time correctly, mainly due to misprediction of the sequential computation time.
- $n = 1024$ DAXPY benchmark fits in cache, but $n = 262144$ FFT does not. This reduces the rate from 285 Mflop/s to 144 Mflop/s.
- Benchmark of computing rate r can be adapted to application, if desired.
- **Communication is the bottleneck**, even though we perform only one data permutation.
- Prediction overestimates the communication time, being based on a **pessimistic g -value**, but the actual parallel FFT was optimised to send data in packets.
- Synchronisation is unimportant for this problem size.

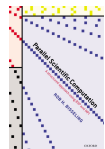


Total computing rate $R_p(n)$

- 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.

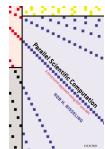


Computing rate $R_p(n)$ of sequential and parallel FFT

		Length n			
p		4096	16384	65536	262144
1	(seq)	220	197	202	155
1	(par)	193	179	180	144
2		239	240	234	243
4		397	375	395	462
8		216	591	607	824
16		30	107	545	900

Rate in Mflop/s. Measured on SGI Origin 3800.

Note: we need **at least 4 processors** to exceed sequential benchmark speed of 285 Mflop/s.



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 **normalised 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.
- Always take a **critical look** at experimental results obtained on a parallel computer.

