Truth in advertising: Reporting performance of computer programs, algorithms and the impact of architecture and systems environment

Scott Hazelhurst

University of the Witwatersrand, Johannesburg

ABSTRACT

The level of detail and precision that appears in the experimental methodology section computer science papers is usually much less than in natural science disciplines. This is partially justified by different nature of experiments. The experimental evidence presented here shows that the time taken by the same program varies so significantly on different CPUs that without knowing the exact model of CPU, it is difficult to compare the results. This is placed in context by analysing a cross-section of experimental results reported in the literature. The reporting of experimental results is sometimes insufficient to allow experiments to be replicated, and in some case is insufficient to support the claims made for the algorithms. Comparing the computational costs of two programs by running them on different computers – even with similar labels and nominal ratings — can be very misleading. New standards for reporting on algorithms results are suggested.

CATEGORIES AND SUBJECT DESCRIPTORS: G.4 [**Mathematical Software**]: Algorithm design and analysis; F.2 [**Analysis of algorithms and problem complexity**]: General; C.4 [**Performance oF Systems**]: measurement techniques

KEYWORDS: comparison, experiments, reproducibility

"As part of my research, I have obtained from others a number of computer codes for algorithms and have run fairly extensive tests on them. Unfortunately, I have discovered that some of them do not perform as 'advertised.' Why this should be I do not know; however, it is clear that there are cases where an algorithm performs well on published problems but poorly on others. Because of such experiences, I have learned to accept the published results of algorithms only after I have verified their results personally." [1]

1 INTRODUCTION

This paper was motivated by research I was doing which involved the development of an algorithm and its implementation. Referees naturally asked how much better my algorithm was compared to those before me.

Email: Scott Hazelhurst `scott.hazelhurst@wits.ac.za`

Consider the following hypothetical claims:

1. Our algorithm A ran in 96s on an Intel Xeon 2.33 GHz computer. Our competitors report that their algorithm B runs in 251s on a Xeon 3.66 GHz machine.
2. Our algorithm A ran in 18.8s on an Intel Xeon 2.33 GHz computer. Our competitors report that their algorithm B takes 362s on an Intel Xeon 2.4GHz machine.

At face value, these seem like impressive performance gains. However, this paper argues that this type of evidence is at best insufficient and that current standards for reporting experimental results are inadequate. The paper argues these points by first showing that 'evidence' such as the hypothetical examples presented above are not much evidence at all and then shows that such examples occur often enough.

The objective of this paper is to explore the standards for the methodology of reporting the performance of computer algorithms and to show

that how performance results are presented really matters.

Structure of paper

This paper is organised as follows.

Section 2 presents experimental evidence of the same program run on over 40 different computers. It shows that the performance of a program on computers with apparently similar specifications can be very different, and that CPU performance can have a second-order impact on program performance.

Section 3 considers the trade-off between quality of result and time taken. In many scientific applications, approximations may be computed due to lack of domain knowledge or because of underlying computational complexity issues. Two different programs may tackle the 'same' problem from slightly different angles, computing slightly different results. Making small sacrifices in quality of result can make very large differences in time taken.

Section 4 explores the effects of different compilers on compuational costs.

Section 5 presents evidence that show that claims of the form of the hypothetical example above are found often in the literature.

Section 6 examines issues related to research in parallel algorithms.

Section 7 concludes and offers some suggestions for the presentation of results.

Related literature

Complaints about the difficulty in understanding performance results go back at least 35 years. In a letter to the editor of *Operations Research*, Ignizio [1] describes some common problems and makes suggestions (like the code being made available). The editor, while being sympathetic to the problem, felt that the suggestions were too expensive to implement. Many of these concerns of practicality have been dealt with by the emergence of the internet.

In the area of mathematical programming, there have been several papers which deal with the issues raised here. Crowder *et al.* [2] present an excellent analysis of designing and analysing experiments as well as reporting previous work surveying the state of experimental algorithmics. This includes a nice discussion of reproducibility — Jackson [3] takes this work forward. Good computer architecture texts such as [4] are also important to read.

McGeoch and Moret [5] present advice to researchers on how to present algorithmics research papers. Some of the issues discussed are how to deal with platform dependence, difficulties and anomalies in experimentation, and the need to present sufficient detail. They do stress, however, the need not to overwhelm readers with detail.

Rardin and Uzsoy [6] discuss the interplay between quality and performance. They give good suggestions of how quality can be assessed in some circumstances.

Good references on parallel computing performance include [7, 8]. Gustafson [9] discusses the impact of architecture on performance and speedup.

Veretnik *et al.* [10] discuss how the availability of software published in the literature affects the reproducibility of results.

Reproducibility in systems research has also received attention. A number of papers have shown that performance figures can easily be misleading. Good examples of such papers include [11, 12, 13, 14], which point out the pitfalls and problems in the methodology of reporting system performance.

Finally, Feitelson presents a philosophical approach to computer science as an experimental science [15].

2 EXPERIMENTAL RESULTS

We now look at detailed experimental results because it helps analysing the literature later on. This section makes the obvious point that the performance of an algorithm really does depend on the exact architecture on which it is run. This point is stressed in many fundamental texts on computer architecture (e.g. [4]); but some algorithms texts do not give this enough attention – it would be fair to say in extreme examples they ask readers to suspend disbelief.

The code described in this paper can be downloaded from `http://code.google.com/p/wcdest` and the data can be found at `http://www.bioinf.wits.ac.za/~scott/wcdsupp`.

This section shows the results of benchmarking the same program on a number of different computers. The program is the `wcd` program (version 0.3.2) [16]. This program clusters expressed sequence tags (ESTs), an important application in computational biology. The problem is computationally expensive (quadratic in data set sizes, which are large – several hundred megabytes is not uncommon). On large data sets, `wcd` can take

over a day in single processor mode (it is parallelised but here we focus on sequential performance). Memory representation is compact and so even with large data sets, the data easily fits into RAM. I/O costs are negligible.

Figure 1 and Table 1 show the results on a range of different data sets. Figure 1 summarises the results, plotting CPU speed versus time taken. Table 1 shows the results in more detail. The key columns are: processor identification; CPU frequency rating; the L2 cache size[1]; the operating system and gcc compiler used; and two columns with times (in seconds), *T1* and *T2*. These show the time `wcd` takes with different parameters for heuristics. For the moment, consider only column T1 which shows the time taken under the default parameters for `wcd` to process the SANBI 10000 data set: this is a small data set of 10000 ESTs (about 3.2M of data). The programs were run at least three times on otherwise unloaded systems and the average time taken (though there is very little variability).

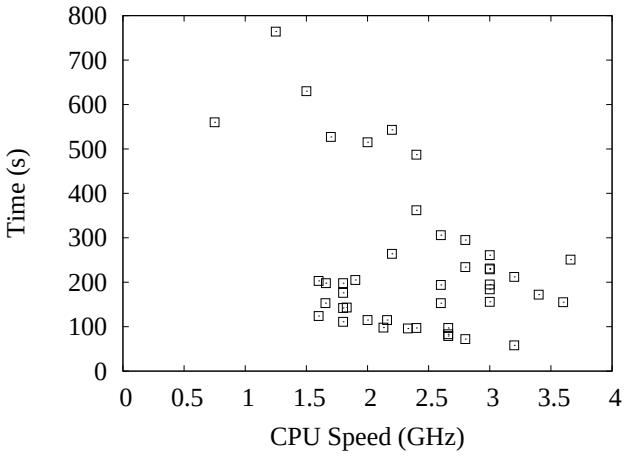


Figure 1: Time versus CPU speed: wcd 0.3.2 on the SANBI10000 data set. The Pearson correlation coefficient is −0.15.

This table shows how important it is to be careful in citing machine models. Although CPU speed has an influence on performance, L2-cache size is much more important. Other factors such as front-side bus speed are also likely to play a role (see the difference between the 5150 and 5355, which both have nominally the same clock and L2-cache size per core).

Even within the Xeon "family", we see remarkable differences. There are numerous models and within the model different versions (steppings). The fastest Xeon time, on the 2.66 GHz 5355, is 3.16 times faster than the 3.6 GHz Xeon, stepping 9. The 2.3 GHz 5345 is 3.7 times faster than the 2.4 GHz Xeon (stepping 7). The stepping 10 of the 3.6GHz Xeon is 1.7 times faster than stepping 9. Beyond the Xeon family the differences are even bigger.

The column labelled *T2* shows the performance of `wcd` with a different set of parameters, which uses more aggressive heuristics. This will be explored in more detail in Section 3. However, it is worth noting here that the times are significantly different, in some cases by a factor of four. A big reason for the change is that with these parameters, the program is much less demanding on L2 cache. Not only are the times different, but the order of the machines is different. Even with closely related machines there are some anomalies: the 3.0 GHz Pentium D is about 10% faster than the 3.4 GHz Pentium D under the default parameters; with the more aggressive parameters, the 3.4 GHz Pentium D is 61% faster than the 3 GHz Pentium D. Figure 2 plots the time T1 versus the time T2. Although the correlation is high (Pearson correlation coefficient 0.79), there are a number of anomalous results which show that different CPUs are relatively better for slightly different parameters for the same algorithm.

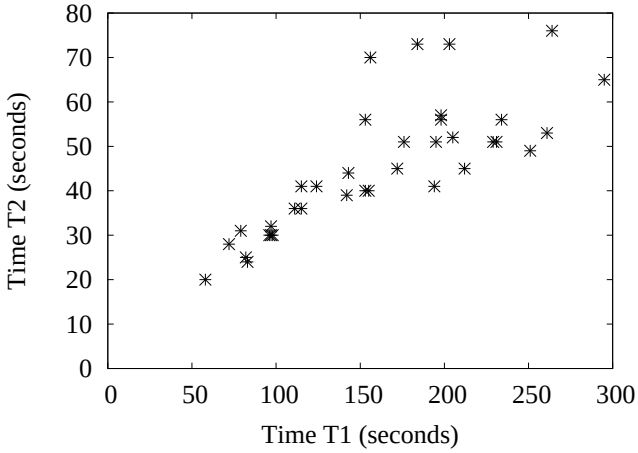


Figure 2: Plot of Time T1 versus Time T2

3 COMPARING APPLES WITH ORANGES

Some algorithms have well-defined solutions, and a program is either right or wrong. However, many algorithms are fuzzier. There may be many

[1]Some of the machines are multi-core. The program was run sequentially and so where the L2 cache is split between the cores, the per-core L2-cache size is shown. Of course, the meaning of the term "L2-cache" varies across architectures, so it would have been more accurate to define this column as "the largest cache for main memory for the chip".

Processor	GHz	OS	gcc	L2	T1	T2
i5 650	3.2	U10.4	4.4.3	4M	58	20
Xeon E5462	2.8	X10.5	4.0.1	6M	72	28
Core Duo E8335	2.66	X10.5.4	4.0.1	6M	79	31
Core Duo P8800	266	X10.6.5	4.2.1	3M	82	25
Xeon E5355	2.66	SL5	4.1.1	4M	83	24
Xeon E5345 (15,7)	2.33	FC7	4.2.1	4M	96	30
Xeon E5150	2.66	X10.4	4.0.1	4M	97	30
Core Duo T7700	2.4	X10.5.2	4.0.1	4M	97	32
Xeon E5506	2.13	SL5.14	4.1.2	4M	98	30
Xeon E5320	1.8	CentOS4	–	4M	111	36
Core 2 Duo T7400	2.16	X10.4	4.0.1	4M	115	36
Core 2 Duo T7200	2	X10.5	4.0.1	4M	115	41
Xeon 5110 GHz	1.6	SuSE10.1	4.1.1	4M	124	41
Pentium E2160 (15,13)	1.8	U7.10	4.3.0	1M	142	39
Core 2 Duo T5600	1.83	X10.5.2	4.0.1	2M	143	44
AMD Opteron 2218	2.6	SLES9.1	3.3.3	1M	153	40
Power 5	1.656	RHEL4.1	3.4.3	1.875M	153	56
Xeon St 10 (Irwindale)	3.6	SL5.0	4.1.1	2M	155	40
Pentium D (6,4)	3.0	FC7	4.3.0	2M	156	70
Pentium D (6,4)	3.4	U7.10	4.3.0	2M	172	45
Core Duo T2400 (14,8)	1.8	U7.05	4.1.2	2M	176	51
Pentium 4 (4,3)	3	FC7	4.1.2	2M	184	73
AMD 285 (33,2)	2.6	RHEL4	3.4.6	1M	194	41
Xeon (4,10)	3.0	U7.04	4.1.2	2M	195	51
AMD 265	1.8	SL5.4	4.1.2	2M	198	56
Core Duo	1.66	X10.4	4.0.1	2M	198	57
Itanium2 iA-64 (2,2)	1.6	SL4.1	3.4.3		203	73
Power4+	1.9	SLES10	4.1.0	1.5M	205	52
Xeon (6,4)	3.2	RH4	4.1.2	2M	212	45
Pentium 4 (4,1)	3.0	U7.10	4.3.0	1M	229	51
Xeon (4,1)	3.0	FC7	4.1.2	1M	231	51
Xeon (4,1)	2.8	RHEL4	3.4.3	1M	234	56
Xeon MP St 9	3.66	SL4.5	4.1.1	1M	251	49
Pentium 4 (3,4)	3.0	SuSe9.2	3.3.4	1M	261	53
Xeon MP (2,6)	2.2	SLES10	4.1.2	2M	264	76
Pentium 4 (2,5)	2.8	U8.04	4.2.4	516K	295	65
Xeon (2,9)	2.6	SuSE8.2	3.3	512K	306	72
Xeon St 7	2.4	FB 4.11	3.4	512K	362	75
Pentium 4 (2,7)	2.4	FC8	4.1.2	512K	487	94
Celeron St 9	2	FB 6.3	3.4.6	128K	515	256
PPC970FX rev 3	2.2	SLES10	4.1.0		543	72
Celeron (1,3)	1.7	FC4	4.0.2	128K	527	269
Sun Sparcv9	0.75	Solaris	—	—	560	330
Power G4 1.1	1.5	X10.5.2	4.0.0	512K	630	132
Power G4	1.25	X10.3	—	512K	764	203

Table 1: Comparison of the performance of the sequential version of wcd on different architectures. For each processor the model and stepping is given when known. In all cases gcc was used with the O2 flag. The OS column gives the operating system: FB – FreeBSD; FC – Fedora Core; RHEL – RedHat Enterprise; SL – Scientific Linux; X–Mac OS X; U – Ubuntu. Some values in the table are unknown

ways to model the underlying reality; there may be heuristics that are used to speed-up the algorithm. In such cases, the question of quality of answer cannot be separated from timing since there is no agreed perfect answer, and there is a trade-off between quality and timing.

To put this into context, look at `wcd` as an example. It clusters DNA fragments based on a measure of distance. If two fragments are less than a distance θ apart they are clustered together. There is no way of determining the 'right' value of θ. The default value in `wcd` is 40, but values between 20 and 60 are used in practice. θ is a *modelling* parameter as we use it to tune our mathematical model to represent the physical reality accurately. The parameters B and K are *heuristic* parameters. By changing these parameters we can trade-off time for quality. With an infinitely fast computer, θ would still exist; B and K would not. Table 2 shows the effects of changing the parameters on clustering a sample data set known as the Public Cotton set. We use as the base, the clustering produced by the default parameters of `wcd` ((B, K, θ)=(10,1,40)) and compare the quality of the answer using the Jaccard Index, which is the primary measure used in the literature. It is clear that changing the modelling parameter has more of an impact than changing the heuristic parameters. Using the more aggressive heuristic parameters, clustering is roughly 2.5-4 times faster. (The discussion here is somewhat simplified because there are other heuristic and modelling parameters.)

Parameters	$\theta = 40$	$\theta = 45$	$\theta = 47$
(B, K)=(10,1)	1.000	0.843	0.733
(B, K)=(8,4)	0.978	0.848	0.764
(B, K)=(8,5)	0.966	0.864	0.766

Table 2: Jaccard Index of clustering produced with given parameters with respect to the default (B, K, θ)=(10,1,40). JI is measured on a scale of 0 through 1, with 1 being a perfect match.

If we compare different programs that produce exactly the same results, then we can check that both produce the correct results and compare the performance results. Our problem is different: other EST clustering algorithms use different theoretical frameworks for determining distance, and use different heuristics for computational speed-up. Thus, it is impossible to compare programs on some common set of input parameters. The only way to do a comparison is to run the codes on the same data set to see the quality of the output. In this area there are no large, universally-agreed benchmarks for quality, and, the only way to do this properly would be to do an exhaustive search of the entire parameter space of both programs being compared to find the optimal settings for that program (though there are probably no universal settings for all data sets).

4 COMPILER EFFECTS

"Computer technology is in such a rapid process of development that it has become virtually impossible to completely reproduce a computational experiment to any arbitrary order of accuracy. This is true even in the same computing environment. Compilers and operating systems change, resulting in different sequences of operations for computing the final results. These different sequences, together with the effect of numerical variations due to round-off, produce outcomes that agree only to a limited number of significant figures." [2].

Table 3 shows the impact of choice of compiler and compiler options on the timing of the program on two different architectures. The GNU `gcc` and Intel `icc` compilers were used with different flags. The choice of compiler optimisation level has a very important effect on timing. This is true of "sensible" compiler options (not comparing for example switching off optimisation completely or tuning for specific architectures).

In summary, differences in performance of up to 25% for sensible optimisation flags are often found, and in some cases more. Some of the anomalous results here are:

- Increasing levels of optimisation do not necessarily improve performance – and the patterns differ between compilers and machines.
- On the Xeon tested, for `wcd` 4.5 on the Public Cotton set, gcc 3.3 with `-O2` is just over 10% faster than icc with the `-O2` option. For `wcd` 3.2 on the SANBI10000 set, icc with the `-O2` and `-fast` options is 18% and 25% faster than gcc 3.3 with the `-O2` option.
- The use of the `-m64` option on the Core Duo (Apple iMac) makes a very big difference in performance, but not for the Xeons.
- An additional result not shown in the table that with `wcd` 3.2 on SANBI10000 with gcc 4.0.1 (flags `-O3-m64`) the Core Duo 2GHz T7200 (MacMini) is faster than any compiler and option tested for the Xeon.

	2.4GHz/T7700		2.3GHz/E5345				
	4.0.1	4.2.2	3.4.3	4.2.1	4.3.0	icc 10.1	4.4.1
`wcd` 3.2 on SANBI10000							
O0	174	174	213	188	187	223	186
O1	96	95	97	98		90	
O2	97	96	95	97	82	82	75
O2 m64	73	80					
O3	91	91	84	80	72	80	68
O3*	65	67				75	
`wcd` 4.5 on Public Cotton Set							
O0	625	625	611	608	658	663	621
O0 m64	597	580					
O1	273	309	260	200	200	205	193
O1 m64	182	184					
O2	273	311	192	196	197	216	189
O2 m64	175	179					
O3	271	308	195	203	201	214	189
O3*	175	180				198	

Table 3: The table shows the performance of two different versions of wcd using different compilers on two different data sets. The top part of the table shows the performance of wcd 3.2 on the SANBI10000 benchmark. The bottom half shows the performance of wcd 4.5 on the Public Cotton data sets. Compiler options — O3*: -O3 -m64 on gcc; -fast on Intel compiler. The -m64 option has no impact on the Xeon. The T7700 is an Intel Core Duo chip running Mac OS 10.5.2 (an iMac), and has a 4M cache. gcc 4.0.1 is Apple's standard gcc release. The E5345 is a Xeon 2.3GHz chip, and is a dual quad-core processor with 4M per processor. We ran the same E5345 with two versions of Linux: gcc 4.3.0 and icc10.1 were run under Fedora Core 7 (kernel 2.6.23), and gcc 4.4.1 was run under Ubuntu 9.10 (kernel 2.6.31). The margin of error is 2s.

Note Compiling with the Intel 10.1 compiler (`-fast` option) and using (8,4,47) as the parameters for (B, K, θ), the 2.3G E5345 processes the SANBI10000 set in 18.8s. Compare this to the 362s that the Xeon 2.4GHz/stepping 7 (gcc 4.1.1 `-O2`) takes with parameters (10,1,40). The quality of the clusterings produced in the two cases would be indistinguishable from differences caused by different methodologies adopted by competing approaches. But, if we tried to compare by simply scaling CPU speed we could claim a 20.09251 fold improvement.

The results shown in this section are not extreme. Showing variability of performance as compilers and systems change is easy[2] to do, and a number of experiments have been omitted due to space constraints[3].

[2]In fact too easy. This research was carried out over a three year period as a side-project. During this period I had to re-run experiments several times as operating systems and compilers changed.

[3]The most telling of these was an experiment which compared two tools for multiple sequence alignment, MUSCLE [17] and Clustal [18]. On one benchmark, we found that Clustal was 2.34 times faster than MUSCLE on one computer with a given compiler and compiler flags, but 15% slower on another machine with a different compiler.

5 CURRENT PRACTICE

Overview

A key part of many research papers in computer science which contain some experimental algorithmics is the measurement of the time that a program takes to run. The experimental methodology is to pick some data sets, justify the use of those data sets, run the program on some piece of hardware, and report the time taken.

This section presents some examples from the literature and a disclaimer is required first. The papers presented below are examples and are not chosen because they are particularly bad — it is easy enough to find other examples. I am critical of some of the sections of the papers reviewed below, but do so in the spirit that I have sinned as much as those I now point fingers at. Most of the papers I think are of high quality. In the text of this paper, I refer to these examples as Paper A, Paper B and so on, so as to semi-anonymise the papers. In the appendix, a table is given which allows an interested reader to verify that I have not taken text out of context. The papers come from high quality journals or conferences, and are

chosen *because* they are good papers in high quality publications.

A common description of an experimental methodology in the computer science literature is to justify the use of some data sets, run them and report the timing as follows (more detailed examples are given later).

> *Time was measured using a personal computer with an Intel Pentium 2.8 GHz processor with 1024 MB of RAM. [Paper A]*

Comparing to experimental work reported in other disciplines is instructive. The following is an example from molecular biology (only two paragraphs of many are shown):

"For protein labelling in vivo*, cultures were washed, incubated for 30 min in a DMEM medium lacking met-cys and labelled for 1 h at 36 hours post-transfection with 35S-met-cys to a final concentration of 200 μCi/ml (48). Total extracts were prepared in Laemmli sample buffer and processed by polyacrylamide gel electrophoresis and autoradiography.*

. . .

"RNA samples were co-precipitated with a molar excess of labelled riboprobe and resuspended in 400 mM NaCl-40 mM Pipes-5 mM EDTA-80% deionised formamide, pH 6.4. After denaturation for 5 min at 858° C, hybridization was carried out overnight at 508° C. The mixture was diluted in a RNAse solution containing 300 mM NaCl-5 mM EDTA-10 mM Tris-HCl, pH 7.5, 50 mg/ml of RNAse A and 1 u/ml of RNAse T1 and incubated for 2 h at 378° C. After proteinase K/SdS treatment as indicated above, phenol extraction and ethanol precipitation, the samples were resuspended in formamide loading buffer and analysed by polyacrylamide-urea gel electrophoresis and autoradiography. Quantitation was performed in a phosphorimager." [Paper B]

Note the difference in level of detail. Both papers come from the same journal, *Nucleic Acids Research*, an impact factor 6 journal with extremely high standards for accepting papers. In the paper with a computing emphasis, the equipment used is described in one line in a footnote — brief and high-level. In a paper that describes biological experiment, the set up is described in great, boring detail, describing exact percentages of concentration, temperature and so on. This is the standard that is expected.

One line of argument, might be that it doesn't matter, because the nature of experiment in computing and in the biological sciences is quite different – see [19, 20, 21] for a discussion.

One big practical difference is that in computer science, the algorithm/program is both a means and an end. It is both a part of the experimental apparatus and what is to be proved by the experiment. And, generally, the piece of hardware used is chosen as an example architecture. In the biological experiment cited above, a mixture was incubated at 378°C. The only things that matter are that (a) *validity*: the choice of mixture and temperature is valid and (b) *replicability*: that the experiment can be replicated with the level of detail given. Other choices of medium and temperature might or might not work, they may be cheaper, easier to work with, but that doesn't really matter. In the case of the computer science experiment, the validity and replicability are important, but usually, *generalisability* is as important. The fact that it runs on a 2.5GHz Mac G5 is usually not as important as that we expect the results of the experiment to generalise to a range of other CPUs. (Usually) an algorithm that works well on a 2.5GHz Mac G5 but doesn't work on a 3.2 GHz Intel Xeon, or even a 2.6GHz Mac G5 would not be interesting.

This line of argument is valid, and explains at one level why a computer science experiment can have a lot less detail than a molecular biology experiment.

However, what this line of argument misses is the replicability. Sometimes, a computer science experiment will justify its algorithm by referring to the performance of the algorithm without reference to any other algorithm. However, it is common for an algorithm to be justified by showing it superiority over other algorithms. [4, p. 33] argue:

The guiding principle of reporting performance measurements is reproducibility *– list everything another experimenter would need to duplicate the results.*

It is true that [Paper A] appears in a journal aimed primarily at the biological community, and in this case the time taken by the program is important but secondary. However, the same can be found in many papers in highly respected computer science journals. The rest of this section gives recent examples from the literature (and more examples can be found in the appendix).

Perhaps not much has changed since the survey of Jackson and Mulvey [22] done 30 years

ago. While no claim to statistical sampling can be made, the examples below show that the concerns of this paper are not far-fetched.

An examination of the 2008 *SIAM Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments*, a hard-core algorithms conference with papers from leading, hard-core computer science groups confirms the pattern. Of the 13 papers which reported timed experimental results

- 5 gave full details of the experimental system (full processor identification, configuration and compiler);
- 4 gave a generic description of the architecture and clock frequency (e.g. Pentium 4, 2.4 GHz) but no cache size, plus compiler details;
- 3 gave a generic description of architecture and clock frequency, but neither cache nor compiler details.
- One presented timing without any description of the architecture.
- Two of the above papers attempted a comparison of their results with others published in the literature based on experiments run by others.

Specific examples from the literature

This section reviews some examples from the literature. I do not claim that these are representative, merely that examples are easy enough to find. These papers were not picked because they were egregious, but because they are good papers and the way the experiments are done are not out of line with what is reported elsewhere in the literature.

Case 1: The first example comes from the *ACM Journal of Experimental Algorithmics* which appears to have one of the most rigorous standards.

> *All runtime experiments were performed on a 2.5 GHz Mac G5 with 4 GB of RAM. [Paper C]*

In [Paper C], the different algorithms are all run on the same machine and so the results reported are valid and internally consistent. This is an example of where the central claims of the authors are well substantiated, but the lack of detail will make replicability by others difficult.

The problem is that there are two "Mac G5 2.5GHz" models, with PowerPC 970fx and 970mp processors. The latter is a dual core machine. Among other differences, the former has 512K of L2 cache, the latter has 1M of L2 cache per core. As was shown before, different cache sizes may have a big impact on program performance. The fact we cannot tell which G5 it was run on shows the difficulty in interpreting results. It may make it difficult for other researchers who wish to tackle the same problem to understand and to compare their work.

Case 2: The second example comes from the *ACM/IEEE Transactions on Computational Biology and Bioinformatics*:

> *"We have run these programs on the same Linux machine with a Pentium 4 2.40 GHz processor and a core memory size of 1 Gbyte. The results of [other researchers' program] are the ones reported in [Paper E] on a machine with the same processor and core memory size of 512 Mbytes." [Paper D]*

The authors of [Paper D] ran their algorithms on a Pentium 4 2.4 GHz machine. But, they did not run the competing algorithm themselves – instead they report what their competitors did. In [Paper E], the researchers also used a "2.4GHz Pentium 4". On some of the experiments performed, [Paper D] reports improvements of 8.6s to 5s, 108 minutes to 53 minutes, and 22 minutes to 9 minutes, which shows a roughly 2.5 fold improvement.

This comparison relies on the assumption that the two researchers used the same hardware except for a different amount of RAM; it relies upon the term "2.4GHz Pentium 4" being well-defined. However, Intel's documentation shows that there were about 30 Intel Pentium 4s models rated at 2.4GHz (see `http://processorfinder.intel.com/`). Some of the variants are minor differences, but there were differences in manufacturing technology (130 and 90 nm), bus speed (400, 533, 800 MHz), and L2 cache (512K and 1MB), with at least 4 major steppings. As seen in Section 2, a reader should be very sceptical that the two platforms are equivalent and should ask what contribution the improved algorithm makes, and what contribution a different variation of hardware makes.

In an attempt to replicate the experiments, the programs were obtained from their authors (who both responded promptly). Both programs were run on different architectures, and the results are shown in Table 5. However, the computations performed are not exactly the same. In both cases, the results were based on randomly generated data, but the ways in which it was

Algorithm	Different machines					
	A	B	C	D	E	F
	3.2GHz	3GHz	2.8GHz	2.6GHz	2.33GHz	1.5GHz
[Paper D]	347	1070	652	1066	535	1066
[Paper E]	174	511	359	874	550	1331
Ratio	0.51	0.48	0.55	0.82	1.03	1.25

Table 4: The programs were both run with parameters $(l, d) = (15, 5)$. Time in seconds. Machines listed from left-to-right in descending chip speed. A: Intel i5 (650), 4M L2 cache; B: Pentium 4 (4,3) 3GHz, 2M L2 cache; C: Xeon E5462 2.8GHz, 6M L2 cache; D: Xeon (2,9) 2.6GHz 512K L2 cache; E: Xeon E5345 2.33 GHz 4M L2 cache; F: Power 4 1.1 1.5GHz 512K L2 cache;. The row labelled *Ratio* gives the value of [Paper E] to [Paper D]. On each machine the same compiler/optimisation values were used (though the compilers differed from machine to machine). Note the ratio appears inversely proportional to nominal chip speed though it is not clear whether this is chance.

done means that comparisons cannot be made from the published work[4]. For these reasons, the results shown below are not compared to the results computed in [Paper D], and nor should they be taken to compare the performances of the two algorithms *per se*, but rather illustrate the following points:

- The difference in the way the experimental detail of these papers compared to the experimental detail of biological paper described above is underlined.
- The relationship between chip speed and performance is weak.
- Even if you compare the two programs on the same machine, which algorithm you would say is better would depend on which machine you ran the code on (assuming for the sake of arguments that oranges are in fact apples). If you try to compare the performance of D on one computer with that of E on another computer, you could make whatever point you liked.

Case 3: Another example can be found in a recent paper in *Journal of Experimental Algorithms* [Paper F]. There can be no quibbling with the results of [Paper F] since their results are so overwhelming, but it is fair to criticise their reasoning, especially as it legitimates a dangerous line of argument — the idea that "the" Xeon can be used as a reference machine is misplaced. (To be fair as well, this also shows the need for source code to be available for comparison: when faced only with executables, it is hard to do better.)

"Our evaluations were performed on two different machines, since opt-k-decomp is only available as Microsoft Windows executable. Thus opt-k-decomp was evaluated under Microsoft Windows 2000 on a 2.4-GHz Intel Pentium 4 processor with 512-MB main memory and both det-k-decomp and BE were evaluated under SuSe Linux 9.2 on a 2.2-GHz Intel Xeon processor (dual) with 2-GB main memory. Note that the different memory sizes have no relevant influence on the results, since the memory usage of det-k-decomp and BE is far less than 512 MB when applied to our instances. We chose the Intel Xeon as reference machine for normalizing the execution times."

Case 4: The next example shows a more extreme example. The following comes from [Paper G], which appears in an impact factor 3.78 journal. They assess their HECT algorithm as follows:–

"*The performance of HECT is compared against ... d2_cluster ... and PaCE (Parallel Clustering of ESTs) ... There are benchmarks available for the d2_cluster (*`http://www.sanbi.ac.za/bench.html`*), which also influenced our decision in choosing the d2_cluster for comparison purposes. The running time for clustering is highly dependent on the length of sequences in the data set and the number of exact matches. The greater the number of matches, the less the time required. Unless both algorithms use the same data set, we would not have a valid comparison. Fortunately, we had access to the same benchmark that was used by the d2_cluster*

[4]The programs do a type of approximate string matching, finding patterns with a specified Hamming distance. The [Paper E] experiment generates random data and sample patterns and then finds all patterns within distance d. The [Paper D] experiment generates random data and patterns and then salts the data with patterns that are exactly distance d from the patterns. Thus, the programs compute different things. The impact of the differences may be small, or they may be large, but an independent reader cannot know.

. . . For this benchmark, which contained 10,000 sequences, d2_cluster took 18,905 sec to run on a SUN SPARK [sic] 400-MHz processor. The same dataset is clustered with HECT on a 650-MHz SUN Ultrasparc IIi workstation in 317 sec. Reverse complement comparison is also performed by HECT, but this information is not available for the d2_cluster. Considering the fact that a 650-MHz machine is (650/400 = 1.625) times faster than the 400-MHz machine Speed up of HECT over d2_cluster *=18905 / (317*1.625) = 36.67.*

"Here, we note that computing the relative performance of two programs by using only the ratio of the processor clock speeds is a gross approximation. There are a lot of other factors (memory and cache size, I/O speed, processor architecture, etc.) that also affect performance. Since the detailed information about the systems used in benchmarking was not available to the authors at the time of this writing, only the processor clock speeds are used for computing the relative performances. A similar comparison is performed against PaCE . . . , which is developed at Iowa State University. PaCE clustered the 10,000-sequence data set in 392 sec on a dual-processor 1126-MHz Pentium III, whereas HECT took 207 sec on a 1 [sic] Pentium III. This means HECT runs 4.16 times faster than PaCE."

An argument is given that leads to a conclusion, accurate to two decimal places, of the performance improvement of their algorithm. There are disclaimers of the problem with this approach, but the authors' take-home message is clear. The claim of a 36-fold improvement is clearly made without any qualification in both the abstract and conclusion. Note the danger in this: the journal itself is subscription-only; the abstract is free to anyone who can use Google.

Before criticising the claim, I accept that HECT is likely to be substantially faster than at least d2_cluster. But, the claims as published, especially in the abstract, are not substantiated. First, as can be seen from Table 1, the correlation between CPU frequency and performance is low. It is easy to find examples in Table 1 where applying the same reasoning as quoted above, one could claim a 7 fold improvement in performance with exactly the same piece of code. Thus, while HECT is likely to be faster than d2_cluster, it is unlikely to be 36.**67** times faster. It might be 400 times faster, it might be 5 times faster. The claimed improvement with respect to PaCE could easily be an artifact of a different architecture. Second, the tools produce different results. No comparison of quality is made between the tools using the same data as was used for performance analysis, and the analysis of quality that was done does not use recognised measures such as the Jaccard Index. As can be seen by comparing columns *T1* and *T2* of Table 1, small changes in parameters can have an important effect on performance. The reader is invited to re-read the concluding paragraphs of Section 4. By running `wcd`, with slightly different parameters, and using the same line of argument, we can show a 20-fold improvement.

Case 5: The next example comes from the graphics area. Consider the extract from [Paper H], comparing to the work of [Paper I] (both in *ACM Transactions on Graphics*, an impact factor 4, leading edge computer science journal):
"Performance in Frames/Sec. (at 1024 × 1024 pixels including simple shading) for OpenRT, Coherent Grid Traversal, MLRT, and BVH. For Toys, Runner, and Fairy, a single time step has been used. MLRT performance data is taken from [Paper I], and corresponds to a Xeon 3.2 GHz with hyperthreading; all other data was gathered on a 2.6 GHz Opteron Machine. "

The algorithm in [Paper H] solves a more general problem than that of [Paper I]. The purpose of the figures quoted above is to show that their algorithm (BVH) pays a reasonable penalty on the simpler cases where the algorithm of [Paper I] can be used. No compiler information is given. [Paper H] also mistakenly describes the CPU o f [Paper I] as a Xeon (it was a Pentium 4). There is no description of cache size.

In turn, [Paper I] compared their results to previous work published by [Paper J]. The authors of [Paper I] state that their algorithm ran on a "2.4GHz P4" and the algorithm of [Paper J] ran on a "2.5GHz P4", and claim an improvement of "up to an order of magnitude". [Paper I] gives no elaboration of their machine; in [Paper J] the machine is described as a Pentium 4 notebook (notebooks commonly have less powerful chips than their desktop or server cousins). No compiler data is given in either paper. There is also a two year gap in publication dates. Convincingly faster – most likely. An order of magnitude faster – not proved.

Availability of software

In order to make fair comparisons, we need to be able to run the same code on the same machine

(or machines). This means that other researchers' code must be available.

In order to see how whether our anecdotal experience[5] was representative, a small survey was conducted using volume 12 of the *ACM Journal of Experimental Algorithmics* as a sample. *JEA* is likely to have much higher standards than most fora for publishing experimental results (indeed instructions to authors strongly encourages uploading of software and data).

There were 18 papers in the volume.

- 5 of the papers had software that was available from the web (either this was advertised in the paper itself or was easily found by using Google).
- I wrote to the first or corresponding author of 9 of the papers asking whether the code was available for experimental evaluation:[6]
  - 2 responded within two days pointing me to software that was packaged and available on the web.
  - 2 responded within two days offering to send me their code once it had been packaged neatly for me.
  - 1 responded two months later (the person had moved).
  - 1 responded with a binary, saying that the source code was not available for distribution. The author could not give any compilation details.
  - 1 responded with source code, but indicated that the code required commercial libraries to run.
  - 1 responded saying the code was probably not in suitable format for others to use.
  - 1 failed to respond.
- So as not to be over-scrupulous, I judged 4 papers to be of the sort where the availability of an implementation of the algorithms discussed not be necessary or useful for people building on the work.

Almost all authors responded positively, and offered help. However, some of the code was not available for various reasons or took trouble to get. Not all the code had documentation that would have made it easy for someone else to compile and run the code.

[5] In our case, we had three months to revise a paper with additional experimentation. For some software which required us to sign a non-commercial use licence and be sent the code it took over four weeks from when we sent our first request to when we received the correct version of the code (and the intervention of collaborator of a collaborator).

[6] Without explaining why I wanted to know.

6 PARALLELISATION

In parallelisation papers, the currency is how well your algorithm parallelises. An examples of such a paper reports as follows [23].

> *All the simulations discussed in this section were run on a Linux cluster with total 18 nodes. Each node has two 3.2 GHz processors and 2G memory. [24]*

First, there are more parameters in reporting parallel computing: nominal network speed, switches, software libraries (e.g., versions of MPI). This makes comparison with other work much more difficult. Moreover, the choice of data used to illustrate results may have a much more profound effect that on a sequential system.

Second, the impact of architecture can be more significant, especially with the need to take into account the impact of sequential algorithm performance.

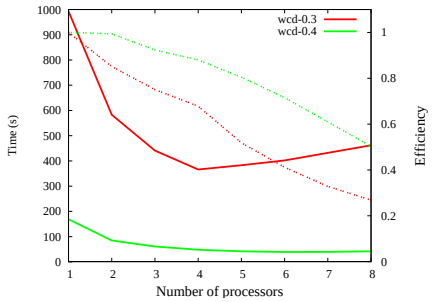
Better sequential algorithms can lead to worse parallelisation. Conversely: to show good speedup, don't pay attention to the algorithm. Figure 3(a) shows how well `wcd` 3.2 and 4.5 parallelises under pthreads on the double quad-core E5345. A lot of effort went into improving the parallelisation between the two versions; however `wcd` 4.5 scales worse. However, what this hides is that the fact that `wcd` 4.5 is much faster sequentially, which makes the synchronisation costs much higher. Even though it scales worse, `wcd` 4.5 is always faster.

Figure 3(b) shows the performance of `wcd` 4.5 with three different settings of the heuristics, chosen so as to slow `wcd` 4.5 down by factors of 1, 5, and 10. As can be seen, with the two latter settings, the efficiency of parallelisation is close to 1 (the ideal).

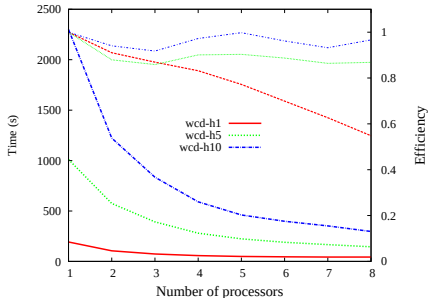
The other influence is the impact of architecture. Figure 4 shows the performance of `wcd` on different multicore architectures. We see different performances of exactly the same algorithm. I also encountered an optimisation which had a big effect on one architecture (changing efficiency from 0.42 to 0.58 on one experiment) but had no effect on other architectures.

7 CONCLUSION

This paper has argued that the current practice of reporting experimental results is inadequate. Often the information does not give a referee or a reader enough information to judge the quality of the results. Particularly bad are papers in which

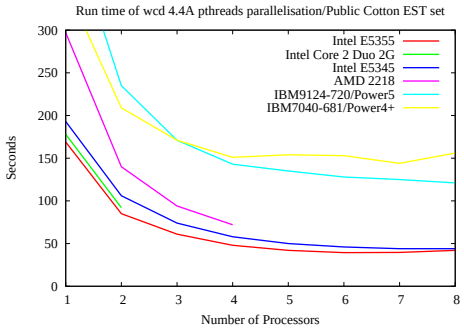


(a) Comparison of wcd 3.2 and 4.5. Absolute time is shown on the left y-axis, efficiency on the right y-axis

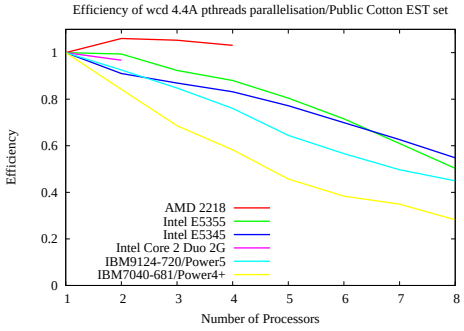


(b) Speed versus efficiency of parallelisation

Figure 3: Impact of sequential performance on parallelisability



(a) Absolute time



(b) Efficiency

Figure 4: Performance of wcd on a range of different multicore architectures

the performance of an algorithm on one computer is compared to the performance of another algorithm on another computer, as if meaningful comparisons can be made.

The points made in this paper are not new – they have been made by others many times over the last few decades. But, the message is still worth hearing and saying again. The contribution of the paper is to show how seemingly small changes to the architecture or system can impact greatly on performance.

It must be emphasised that this study focussed on few variables. One of the most important factors is glossed over – the choice of data for experimentation and analysis. I/O and virtual memory performance is not addressed.

Guidelines

Very clear and sensible guidelines can be found in the literature [1, 2, 5, 8], and there is no need to repeat them here. However, I would like to emphasise that the availability of the internet changes what is possible. Some of the ideas of [1] were not possible 37 years ago. But while the practical dissemination of code through the post using punched cards and magnetic tape were real limitations, the availability of cheap networking and resources like Sourceforge and Google Code changes things.

The most important improvement is to require that algorithms that are compared should be compared on the same machine. For this to be practical, software must be made available for others to use. We need to learn from other disciplines. In most biological journals it is a requirement that any DNA or protein sequences that have been created by the author should be deposited in an appropriate public database. Many bioinformatics papers require that software is made available to referees and even in some cases that the software is freely available to non-commercial users for a period of two years. The benefit of software being open source and available directly from a web or ftp site over other arrangements is large.

McGeoch and Moret [5] make the point that the reader should not be drowned in detail. This is very important. However, the experimental results reported in this paper show the importance of detail. They also underline the importance of comparing algorithms across different data sets, and perhaps on different computers. While it would be wrong to include all of these results in a paper, the use of supplementary resources as done in other disciplines allows authors to make judicious choices of experimental results in the paper itself, while providing full details of additional experimentation and data in the supplementary

material. The internet means that papers can be concise and readable without having to sacrifice accuracy.

- All experimental results should clearly state which model computer was used for experimenting. If necessary, references to the manufacturers' web sites can be made. To help a casual reader, information like CPU speed, memory, and cache configuration should be given.
- The compiler, compiler flags and operating system should be stated.
- Sample data sets should be made available for testing purposes. This should not be an optional extra.
- It may be acceptable or useful to cite the performance figures others have achieved; however, comparisons should be made circumspectly, and it is doubtful that promoting performance improvement gains on these grounds to prominent parts of abstracts or introductions is justifiable.
- In algorithms where there is no clear right answer, and where heuristics and approximations are employed, there must be an explicit discussion of the relationship between algorithm computational and quality performance.
- It is probably desirable to use more than one computer system for experimental purposes. The more important the empirical experimental results are to support the claims of the paper, the more important this becomes. One of the implications of the different orderings of the two timing columns in Table 1 is that an algorithm A may do better than algorithm B on one architecture but not another. This is confirmed by the results of Table 5 and other experiments we carried out. Thus, to claim generalisability without compelling analytic evidence, it may be necessary to explore and elaborate a range of algorithms.

New tools and systems that would help researchers share tools and results, and to validate previous results are now possible. Suggestions such as Dataforge [25], paralleling Sourceforge, and the use of cloud computing [26] are very good examples of approaches that would help researchers in experimental algorithmics and systems research.

ACKNOWLEDGMENTS

I thank the authors of volume 12 of JEA and the authors of [Paper D] and [Paper E] who were unwitting guinea pigs in my experiment; the way the responded is a credit to the spirit of scientific cooperation. Philip Machanick made useful comments on a draft of the paper. I would also like to thank the very careful and insightful comments made by an anonymous referee.

REFERENCES

[1] J. Ignizio. "Validating Claims for Algorithms Proposed for Publication (letter to the editor)". *Operations Research*, vol. 21, no. 3, pp. 852–854, 1973.

[2] H. P. Crowder, R. S. Dembo and J. M. Mulvey. "Reporting computational experiments in mathematical programming". *Mathematical Programming*, vol. 15, no. 1, pp. 316–329, Dec. 1978.

[3] R. Jackson, P. Boggs, S. Nash and S. Powell. "Guidelines for reporting results of computational experiments. Report of the ad hoc committee". *Mathematical Programming*, vol. 49, 1991.

[4] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 4 edn., 2006.

[5] C. McGeoch and B. Moret. "How to present a paper in experimental algorithmics". *SIGACT News*, vol. 30, no. 4, Dec. 1999.

[6] R. Rardin and R. Uzsoy. "Experimental Evaluation of Heuristic Opitimization". *Journal of Heuristics*, vol. 7, no. 1, pp. 261–304, 2001.

[7] R. Barr and B. Hickman. "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions". *ORSA Journal of Computing*, vol. 5, no. 1, pp. 2–18, 1993.

[8] L. A. Crowl. "How to Measure, Present, and Compare Parallel Performance". *IEEE Parallel Distrib. Technol.*, vol. 2, no. 1, pp. 9–25, 1994. ISSN 1063-6552. doi:http://dx.doi.org/10.1109/88.281869.

[9] J. Gustafson. "The "Tar Baby" of Computing: Performance Analysis". *ORSA Journal on Computing*, vol. 5, no. 1, pp. 19–21, 1993.

[10] S. Veretnik, J. L. Fink and P. E. Bourne. "Computational Biology Resources Lack Persistence and Usability". *PLoS Computational Biology*, vol. 4, no. 7, p. e1000136, Jul. 2008.

[11] A. Georges, D. Buytaert and L. Eeckhout. "Statistically rigorous Java performance evaluation". In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07,

pp. 57–76. ACM, 2007. URL http://doi.acm.org/10.1145/1297027.1297033.

[12] J. Mogul. "Brittle Metrics in Operating Systems Research". In *Workshop on Hot Topics in Operating Systems*, p. 90. IEEE Computer Society, Los Alamitos, CA, USA, 1999. ISSN 1530-1621. doi:http://doi.ieeecomputersociety.org/10.1109/HOTOS.1999.798383.

[13] T. Mytkowicz, A. Diwan, M. Hauswirth and P. Sweeney. "Producing wrong data without doing anything obviously wrong!" In *Proceeding of the 14th International Conference on Architectural support for Programming Languages and Operating Systems*, ASPLOS '09, pp. 265–276. ACM, New York, NY, USA, 2009.

[14] T. Mytkowicz. *Supporting Experiments in Computer Systems Research*. Ph.D. thesis, University of Colorado, 2010.

[15] D. G. Feitelson. "Experimental Computer Science: The Need for a Cultural Change", 2005. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.76.1883. Manuscript, Department of Computer Science, Hebrew University of Jerusalem.

[16] S. Hazelhurst, W. Hide, Z. Lipták, R. Nogueira and R. Starfield. "An overview of the wcd EST Clustering Tool". *Bioinformatics*, vol. 24, no. 13, pp. 1542–1546, Jul. 2008. doi:10.1093/bioinformatics/btn203. Doi:10.1093/bioinformatics/btn203.

[17] R. Edgar. "MUSCLE: multiple sequence alignment with high accuracy and high throughput". *Nucleic Acids Research*, vol. 32, no. 5, pp. 1792–97, 2004.

[18] M. Larkin, G. Blackshields, N. Brown, R. Chenna, H. McGettigan, P.A. a nd McWilliam, F. Valentin, I. Wallace, A. Wilm, R. Lopez, J. Thompson, T. Gibson and D. Higgins. "Clustal W and Clustal X version 2.0". *Bioinformatics*, vol. 23, no. 21, pp. 2947–2948, 2007. doi:10.1093/bioinformatics/btm404. URL http://bioinformatics.oxfordjournals.org/cgi/content/abstract/23/21/2947.

[19] P. J. Denning. "Is computer science science?" *Commun. ACM*, vol. 48, no. 4, pp. 27–31, 2005. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/1053291.1053309.

[20] G. Dodig-Crnkovic. "Scientific Methods in Computer Science". In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*. Skövde, April 2002. URL http://www.mrtc.mdh.se/index.php?choice=publications&id=0446.

[21] A. Eden. "Three Paradigms of Computer Science". *Minds Mach.*, vol. 17, no. 2, pp. 135–167, 2007. ISSN 0924-6495. doi:http://dx.doi.org/10.1007/s11023-007-9060-8.

[22] R. Jackson and J. Mulvey. "A critical review of comparisons of mathematical programming algorithms and software (1953-1977)". *Journal of Research of the National Bureau of Standards*, vol. 83, no. 6, pp. 563–584, 1978.

[23] F. Silva, E. Lopes, E. Aude, F. Mendes, H. Serdeira and J. Silveira. "Parallelizing Black Oil Reservoir Simulation Systems for SMP Machines". In *ANSS '03: Proceedings of the 36th Annual Symposium on Simulation*, p. 224. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 0-7695-1911-3.

[24] C. Yang, S. Chakraborty, D. Gope and V. Jandhyala. "A parallel low-rank multilevel matrix compression algorithm for parasitic extraction of electrically large structures". In *DAC '06: Proceedings of the 43rd Annual Conference on Design Automation*, pp. 1053–1056. ACM, New York, NY, USA, 2006. ISBN 1-59593-381-6.

[25] P. Dutta, A. Ganapathi and D. Molnar. "A Case for DataForge: A SourceForge For Experimental Data", 2005. http://www.cs.berkeley.edu/~prabal/pubs/papers/dataforge.pdf, Last accessed 16 November 2010.

[26] J. Dudley and A. Butteabutte. "*In silico* research in the era of cloud computing". *Nature Biotechnology*, vol. 28, pp. 1181–1185, 2010. doi:10.1038/nbt1110-1181.

Table of Papers used as examples

- [Paper A]: doi:10.1093/nar/gkm288
- [Paper B]: doi:10.1093/nar/gkm230
- [Paper C]: doi:10.1145/1227161.1227167
- [Paper D]: doi:10.1109/TCBB.2007.70241
- [Paper E]: http://www.comp.nus.edu.sg/~wongls/psZ/apbc2005/camera-ready/212.pdf
- [Paper F]: doi:10.1145/1412228.1412229
- [Paper G]: doi:10.1089/dna.2004.23.615
- [Paper H]: doi:10.1145/1141911.1141913
- [Paper I]: doi:10.1145/1073204.1073329
- [Paper J]: http://graphics.cs.uni-sb.de/Publications/webgen///EGStar03/download/star03.pdf