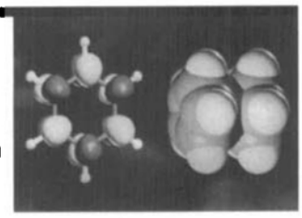
CHIMIA 47 (1993) Nr. 1/2 (Januar/Februar)

COMPUTATIONAL CHEMISTRY COLUMN

Column Editors:
Prof. Dr. J. Weber, University of Geneva
Prof. Dr. H. Huber, University of Basel
Dr. H. P. Weber, Sandoz AG, Basel



Chimia 47 (1993) 22–23

ISSN 0009–4293

Your Speedup is a Megaflop!

'I just bought a new personal computer, the LAST, with a speed of 50 MHz, whereas your old MADOS has only a 25 MHz frequency'. 'You are joking, the LAST makes only 2 Mflops, whereas mine makes 25 Mips.' 'Never mind, your old scalar processors will not do it for long, I just read about a supercomputer of the Teraflop generation with a speedup of over 100!'

Not far from reality – perhaps virtual reality. Such a talk shows how confusing the mixing of abbreviations and misconceptions in the field of new developments in computing is. As many applications in chemical research are real number crunching problems, the measures of speed of a computer have generally a large importance, *e.g.* if one has to decide whether it pays to port a program to a certain computer, whether one should buy a work-station or rather try to get computer-time on a so-called supercomputer. These reasons prompted us to publish in previous *Columns* two comparisons of computer-performances with two different typical applications in the field of chemistry, quantum chemical calculations [1] and simulations [2]. In this *Column*, we discuss some of the measures often used for scalar, vector, and parallel processors and show how carefully one has to deal with them.

One of the most basic comparisons is performed by clock-rate. Each computer has a clock, which gives the working *frequency* of the computer. If we compare two processors of the same type, *e.g.* Motorola 68030 found in many Macintoshes, one with a frequency of 25 MHz and one with 40 MHz, then the second one is normally 1.6 times faster. This might not be true for the whole computer as other components may not be scaled by the same ratio! The clock-rate is also a bad measure for a comparison between different processor-types as different processors might need a different number of clock-cycles per instruction.

Another common measure is the *millions* of *instructions* performed *per second* (*Mips*). Again this is not a very useful measure for number crunching as different processors need a different number of instructions to perform a useful calculation, *e.g.* a multiplication. For example workstations usually have risc-processors (reduced instruction set computer) which in average need slightly more instructions to perform floating point operations than the 'classical' main-frame processors.

After all, if we are interested in number crunching, why do we not use the number of floating point operations per second as a measure? Indeed, the *Mflops* (*Megafloating point operations per second*) are quite a good measure for most comparisons of scalar computers. The problems start with some of the new risc-architectures, as well as with the vector- and parallel-computers! This is the point where we should perhaps say something about vector- and parallel-processing for readers who are not familiar with the subject.

Vector-processing is perhaps best compared with the work done at an assembly line. Let us assume you have in your program a loop with 20 instructions to be performed and the loop is repeated 100 times with different data. If you have one worker (scalar processor) you give him the first instruction which he applies to the first number, than the second instruction is applied to the same number (or its result) and so on until the last of the 20 instructions is performed with the first number. Then the whole process starts again with the second number *etc.*, *i.e.* you wait 100* 20 or 2000 steps until your loop is finished. For simplicity let us assume now you have instead of one worker an assembly line with 20 workers (vector-processor of a vector-length twenty). At the first step the first number enters the assembly line and is handled by the first worker as in a scalar processor. However, in the second step the second worker performs the second instruction on the first number at the same time as the first worker performs the first instruction on the second number, *etc.*. For the first 20 steps no product is leaving your assembly line (no result is leaving your vector-processor) but after that you get a product (result) in each step, *i.e.* after a total of 120 steps your loop is processed! You gain a factor of about 17, *i.e.* if your scalar processor performs 1 Mflops, your vector-processor yields 17 Mflops, which would roughly correspond to the performance advertized by the computer manufacturer. However, any scientific program has instructions other than loops which do not vectorize and, therefore, the efficiency is in practice much lower! Depending on your problem and on the specific architecture of the processor, the computing power might be quite different. Moreover, believe it or not, by making the programming run slower, you can rise the Mflops rate. Just add to your code a large loop, which vectorizes excellently, but does not produce any useful numbers for your problem and you will see that your program uses more computer time but shows (due to the additional well vectorized part) a higher Mflop rate. This might look strange to you, but such things happen often when comparing programs based on different algorithms and containing such portions of code which are unknown to the tester. A typical example is a scalar program where some tests (if) are present in a loop, which might tell the computer to perform the loop only if a condition is satisfied. To make the loop vectorize, one has usually to eliminate the test (if), which rises the Mflops rate due to vectorizing, but also makes the program performing unnecessary operations.

As vector-processors have probably already passed their climax of success and parallel- or massively *parallel-processing* (*MPP*) are the key words for the near future, we would like to say also a few words about the speedup measure used in parallel computing. Parallel computing means that your workers are not standing at an assembly line, but do the same operation at the same time on different data (single instruction multiple data; SIMD) or do even different things with different data (multiple instructions multiple data (MIMD); we are not going into more de-

tails here). The 'speedup' is the factor you gain in speed if you use n processors instead of one to solve a problem. If you have for example 20 workers (processors) working together, they will usually loose some time for communication, *i.e.* exchange of their working pieces (data), and, therefore, reach a 'speedup' of less than 20, let us assume only 12. Such a low number would result if they need to communicate significantly and are often blocking each others way. Again you could 'cheat' by giving each worker (processor) additional work which is not needed, but which forces him to sit longer at his table and do relatively less communication, making the process slower, but the speedup higher!

Finally, we would like to give an example from the real world of quantum chemistry, where people are not 'cheating' their processors, but nevertheless similar effects can be found. *Lüthi et al.* [3] reported results from a calculation on a Cray Y-MP/8-128 supercomputer with the DISCO-program for bis(2,6-dimethylphenyl) carbonate ($C_{17}O_3H_{18}$), a molecule with 38 atoms (314 contracted / 610 primitive basis functions), in which one iteration took about 400 s (this number is different from the one in [3], which was wrong, due to an input error [4]) and a performance of 1531 Mflops was achieved. The speedup for 8 processors was 7.65. *Brode* [5] has carried out a very similar calculation on the same molecule (356 contracted/592 primitive basis functions) with the TURBOMOLE-program on a workstation cluster of 14 machines performing to a maximum rate of 660 Mflops. The speedup was only 11.6. Although the loss in parallelization was higher and the Mflop-rate was much smaller (the formal rate for the 8 Cray processors would even be 2660 Mflops) the time for the first iteration (taking usually most time) was only 524 s, *i.e.* slightly more than with DISCO. Similar experiences have been made by *Vogel et.al.* [6] with a version of DISCO on a network of workstations.

Summarizing, we can state that the Mflop- and the speedup-measure are often not very useful criteria for the real world of vector- and parallel-computers, but that *a comparison between programs solving problems as similar as possible is the best way to estimate the performance of computers for a specific task.* Policies such as that of CSCS in Manno, enforcing that only programs yielding 275 Mflops should run on the NEC, are questionable in view of the difficulty to accurately estimate the efficiency of application programs running on vector processors. An alternative policy would be to supply a supplementary national cluster of workstations or a super parallel computer for codes performing badly on a vector processor making the scientists choose the most efficient facility for their purpose.

We thank Dr. *Stefan Brode*, *BASF* Aktiengesellschaft, Ludwigshafen/Rhein and Dr. *Hans Peter Lüthi*, Interdisziplinäres Projektzentrum für Supercomputing, ETH, Zürich for the example.

[1] Th. Bally, P.-A. Carrupt, J. Weber, *Chimia* **1991**, *45*, 352.
[2] R. Eggenberger, H. Huber, *Chimia* **1992**, *46*, 227.
[3] H.P. Lüthi, J.E. Mertz, M.W. Feyereisen, J.E. Almlöf, *J. Comput. Chem.* **1992**, *13*, 160.
[4] H.P. Lüthi, private communication.
[5] S. Brode, private communication.
[6] S. Vogel, J. Hutter, T.H. Fischer, H.P. Lüthi, *Int.. J. Quantum Chem.*, accepted for publication.

Chimia 47 (1993) 23–24

ISSN 0009–4293

How Computer Science is Taught to Our Students in Chemistry. Part II

Heiner G. Bührer*

In a recent survey [1], the present state of the compulsory introduction courses in computer science taught in different Swiss universities and federal institutes of technology were compared with a report called 'Recommendation for the Introduction of Computer Science in the Chemistry Curriculum' [2]. The survey did not take into account the seven chemistry departments of the Swiss Schools of Engineering (Ingenieurschulen, HTL), which confer about one third of all diplomas in chemistry in Switzerland [3]. In the following *Table* and list of contents this gap has been filled. Some conclusions can be drawn with regard to [1]:

- All Schools of Engineering offer to their chemistry students compulsory courses in the first two semesters. Most of them propose advanced courses in higher semesters. As many schools emphasise on chemical engineering, automation, and electronics are additional elements of training. The School of Engineering at Geneva is different inasmuch as its curriculum lasts five years and is meant for chemical engineering students only.
- The total number of hours per week varies among the different Schools of Engineering. Most of them devote a significant part of time to computer applications in the laboratory or pilot plant (simulations, data comprehension, processing and modelling), which appears only partially in the *Table*.
- A first introduction to computer architecture and programming is often given by a computer specialist or mathematician. Chemical applications are usually taught by a chemist.
- As is the case with university students, PASCAL is here too the preferred programming language. However, many Schools of Engineering tend to reduce the number of lessons devoted to programming for the benefit of computer applications in chemistry.
- In general, the differences between the university students' results [1] and those of students at Schools of Engineering are small. There may be less time for programming for the latter but more for chemical and technical applications; this reflects the difference in interest and requirements of the two groups of chemists.

**Correspondence*: Prof. H.G. Bührer
Technikum Winterthur Ingenieurschule TWI
Postfach 805
CH-8401 Winterthur