Physics vs. Computer Science

Robert Pike

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

With computers becoming more frequently used in theoretical and experimental physics, physicists can no longer afford to be ignorant of the basic techniques and results of computer science. Computing principles belong in a physicist's tool box, along with experimental methods and applied mathematics, and the easiest way to educate physicists in computing is to provide, as part of the undergraduate curriculum, a computing course designed specifically for physicists. As well, the working physicist should interact with computer scientists, giving them challenging problems in return for their expertise.

1. Computing as a Part of Physics

Computing has become an integral part of the practice of physics. Computers are now used throughout physics, from experimental control to plotting graphs, from simulations to typesetting papers, yet many physicists are not adequately educated in their use. As with mathematics, proper use of computing requires some knowledge of the techniques of the field and how they can be brought to bear on a problem. Although from a physicist's point of view there is less to know about computing than mathematics, the importance of computing in physics is increasing dramatically and physicists well-versed in the methods of computer science can use computers to advance science more effectively.

By computing I mean the manipulation of data in a practical sense — the engineering of problem solving on a computer — rather than formal computer science. Physicists face many practical computing problems in their research, problems that the practise of computer science addresses. The general feeling among physicists is that computing is a form of math which can be picked up just by trying it — basically, that it is an easy subject for the physicist. I am not going to dispute that assessment, but will argue that a modest investment of time in the study of the basics of computing will pay off richly in time saved and, sometimes, quality of research. Effective use of computers requires a fair amount of expertise, and much of that expertise is difficult to acquire without expert help. The situation is analogous to learning applied mathematics, say integral calculus: to become adept at calculus, considerable practise is required, but no one is expected to derive the basic results of calculus, discover all the tricks such as integration by parts, and then memorize the integral tables. Similarly, good computer programmers have plenty of experience, but they haven't invented for themselves all the techniques: they have books which are basically algorithm catalogues, and have learned, from other programmers, computer scientists, books and courses, enough about the theory to choose suitable algorithms or programs for a problem.

There is much to know about computing. Some of the areas where computing know-how can help the physicist are:

Experimental control
: Running an experiment with various detectors and instruments operating simultaneously is similar to keeping a time-sharing operating system running.

Data Reduction
: Interactive statistical packages and computer graphics enable a scientist to understand the implications of an experiment much sooner than line printer graphics on a batch computer.

Discrete simulations
: Many physical problems involve more than solutions to differential equations. Group theory, non-rectangular lattices and other components of modern problems can be approached more easily using ideas from graph theory and other computing fields far removed from the typical FORTRAN program.

Numerical simulations

Many numerical problems are now considered solved, in the sense that commercial software packages are available that give accurate solutions efficiently.

Symbolic manipulation

Using computers to solve algebraic problems is more of a computing task than a mathematical one, and using an algebraic manipulation system requires a thorough understanding of the programming art, as well as mathematics.

The techniques of computing belong in a physicist's toolkit, alongside the techniques of instrumentation and applied mathematics. A physicist need not be an expert computer scientist, just as he or she need not be a research mathematician, but well-informed use of computers can be more effective than the ad hoc use that pervades the physics community. Also, knowing what computers can do and how they do it demystifies them. Any result generated by a computer should be regarded with healthy skepticism — the output from an erroneous program does not describe a new physical effect — but a computer can also solve problems that are unmanageable by other means. It is important to be able to criticize constructively research based on computer methods.

Nowadays, many experiments take place largely inside a computer: experimental high-energy physics would be unthinkable without computers to design the apparatus, control the experiment, detect interesting events and reduce the data. Many theoretical ideas are explored by using computer simulations or symbolic manipulation programs to test hypotheses, perform laborious calculations or create worlds inaccessible to experimentalists. But despite the importance of computing to experimental and theoretical methods, the details of the computing procedures used in a problem are rarely published in the literature. Experimental methods are presented in the literature thoroughly enough to permit others to reproduce the experiment, or at least assess its validity, but computing methods are usually dismissed with a phrase such as "computer simulations of the model showed that...." The computing techniques deserve attention in the literature as part of the experimental methods. When a result depends on a computer model or an innovative program, the validity of the result can be assessed only when the programming techniques are presented well enough to critically evaluate them, if not reproduce them. Just as a paper would mention that second-order perturbation theory was used to derive a result, or state which spectrum analyzer generated the plots in a figure, it should name the numerical algorithm or commercial subroutine used to integrate a differential equation, since the accuracy of a result generated by numerical integration depends on the algorithm used. It is not necessary to reproduce the program, just to provide a reference so that the critical reader can assess the suitability of the methods used.

On a related topic, with computers controlling and interpreting so much in modern experiments, the computer methods and programs in an experiment should be designed along with the apparatus and data reduction techniques, not left for a first-year graduate student after the experiment comes on-line. There are important issues in the design of large experiments that depend on understanding of a computer's capabilities for proper solution: questions such as what the interface from the experiment to the controlling processor should be, which data should be collected for later analysis, or how much processing should be done immediately and how much deferred for later analysis. Without some experience in real-time programming, an experimentalist might be surprised that the on-line processing power is insufficient to keep up with the data flowing from the apparatus; for example, the processor may be too busy servicing I/O interrupts keeping the experiment running to perform first-order event rejection.

2. The Importance of Data Structures and Algorithms

A great deal has been written about the suitability of various languages, particularly (in the physics community) FORTRAN, to modern programming problems, so I won't spend much time on the subject. Nonetheless, there are a few points that cannot be overemphasized. FORTRAN is a language of the 1950's. Although FORTRAN was an impressive accomplishment for its time, in the 25 years since its creation, great progress has been made not just in programming languages, but in understanding models of computation and how we use computers. FORTRAN reflects none of that progress, remaining alive primarily because it is the lingua franca of scientific computing, and the vast bulk of existing physics software is written in it. FORTRAN's position is reminiscent of that of Latin in the 18th century, supported more by tradition than merit.

Perhaps the greatest weakness in FORTRAN is the lack of facilities for structuring data. The notion of a data structure is best conveyed by example: In a program simulating an ideal gas, each particle in the gas

might be described by a position 3-vector, a velocity 3-vector, a mass and some identification of its composition. In FORTRAN, these variables would have to be stored in separate arrays, and a particular particle identified by an index into the arrays. In more modern languages, the variables of each particle can be grouped together in a single structure and treated as a unit. The following data declaration (the language is C) should make the idea clear:

```
struct Particle {
        float   pos[3];
        float   vel[3];
        float   mass;
        char    comp[10];
};
```

The Particle structure contains all the information about a particle in a single place, and the different components of the data have names with mnemonic significance.

It is convenient to have a particle's information localized. For instance, if the program is simulating a chemical process, particles may change in type and number as compounds form or dissociate, and maintaining an array of particles may be difficult in practice. With all the data for a particle in one place — in a Particle structure — the storage for a particle may be allocated dynamically by the language's storage allocator, and returned to the pool of available storage when the particle "dies." Then, a particle is referred to by its address stored in a pointer to the data, and the pointer can be conveniently used to pass all the information about the particle to other parts of the program. Such techniques are difficult to arrange in FORTRAN, which has neither dynamic storage allocation nor pointers.

Using a different data structure for a problem, perhaps a graph instead of an array, can sometimes permit use of a more efficient algorithm. A good algorithm can mean the difference between an unsolvable problem and an tractable one. In percolation, the obvious algorithms take time proportional to n^2, where n is the number of occupied sites in a cluster. Algorithms based on sets and graphs solve the same problem in time proportional to $n \log n$. Using an optimal algorithm on a cluster with a million sites can reduce a calculation from 50 years of computer time to a few hours (in percolation, clusters must be large to be inside the critical region, the area of physical interest).

The point here is not that the modern ideas make possible computations that were impossible in FORTRAN, but that problems can become easier to solve, and the programs easier to write, understand and debug, when expressed in a better language. Many physicists see computers only as what FORTRAN provides to them, but computers can be much friendlier and more flexible. And even if the programming environment supports only FORTRAN, knowledge of what other languages do can dramatically simplify the programming process. For example, many algorithms are most easily expressed recursively, but FORTRAN does not permit recursive subroutines. But if FORTRAN is the only language supported on the computer, there exist tools for converting recursive algorithms into iterative ones, and knowing how to exploit recursion in a problem may lead to a quicker, simpler solution, even if the final program is, of necessity, iterative.

Not all programs are written to solve physics problems directly: many are tools for the programming process. Examples, besides the obvious ones such as compilers and text editors, are libraries of useful numerical subroutines, sorting programs, graph plotting packages and interactive languages for restricted problems such as statistical analysis. Some programming environments make tool-building and tool-using easy. This applies not just to languages, but computing environments: subroutine libraries, operating systems and computer centers. It is often much easier to throw together a few small programs that can collectively perform a task than to write one from scratch. If the problem can be broken into small pieces, some of the subproblems may be solved by available programs, and it will be unnecessary to write a new, large program for each problem. Knowing this, a programmer can make better use of the machine by writing small programs with clean interfaces to other programs — tools — and combining them quickly, perhaps writing a new tool or two, when a new problem arises. Many of the basic tools should already exist in the environment: a programmer should *never* have to write a matrix subroutine or a sorting program. But often the required tool does not exist, and the programmer must create it. If designed carefully, it will fit smoothly into the programmer's tool kit and be useful on other problems.

As a specific example, the Unix† time-sharing system provides a simple mechanism, called a pipe, for combining programs interactively. Programs typically read some data from a single input, process it, and write the result on its output file. The pipe notion is simply a mechanism to connect the output of one program to the input of another. The syntax is simple:

```
A | B | C | D
```

causes A's output to be processed through B, B's output through C and C's through D. A could be a program that reads a data tape from an experiment and extracts events, putting them into a simple format on its output. B could be a program that recognizes a certain class of event, C might sort the events based on some property, and D could produce a graph of the resulting data. If the right tools are available, a program (a pipeline like this is really a little program) can be put together in a few seconds, to try out some hypothesis. Although each program must read and write data, so there is inherently some loss of efficiency, programmer time is a resource scarcer than computer time; small tools are easier to write and can to combine quickly, leaving most of the work for the machine.

3. Education

Just as physicists learn mathematics, they must learn some computer science if they are to make proper use of the machines. It is certainly not good enough to take a first-year university course in FORTRAN programming; a physicist must learn some of the basics of computing at a level above that provided by a single, ancient, language. The following is a proposed curriculum for one or two computing courses for physicists:

Machine models

What a computer can do; the notion of an instruction; recursion and iteration; basic complexity theory. Writing good programs requires some understanding of what computers are, how they perform computation, and how long the computation will take.

Algorithms and Data structures

Basic algorithms; sorting; graph theory; abstract data types. The simplest solution to a problem may be very different from a DO loop, and the data to describe parameters in the problem may be represented poorly by integers or arrays. Some understanding of the basic algorithms and data structures of computer science can make simple solutions easier to find.

Programming languages and parsing

FORTRAN viewed of a language, rather than *the* language; Pascal or some other language with data structures; parsing techniques. Learning two or more languages gives a much better understanding of what a computer can do, and how to make it do it. As with the wave and matrix formulations of quantum mechanics, some problems are easier to handle in one language than another. The FORTRAN part of the course should be handled as necessary background, teaching not only how to use FORTRAN but how to avoid it when possible, or at least overcome its inadequacies. Physicists sometimes write special programs that read some input language, and knowledge of parsing theory may help in designing a language that can be easily handled by both the physicist and the computer.

Numerical Analysis

Basic error analysis; what numerical problems computers can solve; where to get subroutines to do the job. It is much more important to give guidelines to finding a solution than to develop the skills to write libraries of numerical analysis subroutines. Physicists must be aware of what problems are solvable by current numerical software, what commercial subroutines are available, and how to express their problems in a form suitable for solution.

Operating systems and real-time programming

The I/O architecture of computers; interrupts; real-time processing; multiprogramming. Modern experiments require computer control, and a physicist needs at least the basic notions of how a computer can control a machine. Interrupts allow a processor to service multiple I/O ports conveniently, but a program which spends much time servicing interrupts may be unable to keep up with the incoming data. Operating systems face the same sorts of problems, and the techniques developed to solve them — processes and inter-process communication, buffering, etc. — are directly applicable to real-time control of experiments.

---

† Unix is a trademark of Bell Laboratories

Graphics

Bitmap and vector displays; interactive graphics; data display; 3-d graphics. Physics describes the interrelationships between the variables in a system, and graphs are used constantly to present and explore relationships. Computers are good at drawing graphs on paper, but they can also be used interactively to explore dynamically the properties of a function or the parameters of a system in pictorial form. As with numerical methods, familiarity with commercial software is more important than being able to create new graphics packages.

These subjects should be taught in courses specifically designed for physicists, not computer science students.

4. Communication

Along with education goes communication: talking with experts in computing. Computer scientists have solved many problems that arise in physics, and physicists should waste no time solving them again. On the other hand, physics is full of interesting computational problems that computer scientists would find challenging and enriching. More interaction between the disciplines can only help both sides.

Numerical analysts, in particular, are always looking for new problems to solve, and know where to find solutions to old problems. But to get the most out of interacting with a numerical analyst, a physicist must not try to second-guess the solution. A numerical analyst is better, from experience at least, at deciding how to best code a problem for computer solution, and a physicist should spend little time trying to rearrange the problem for simpler solution on a computer. For example, a physicist might try to convert a moving boundary value problem into a more difficult equation with a stationary boundary, unaware that second order moving boundary value problems are basically a solved discipline of numerical analysis.

In summary, computer science has a great deal to offer the practicing physicist, theoretical or experimental, and physicists would do well to add computing techniques to their repertoire of problem-solving methods. The result can be more effective use of computers, and more time free for solving the real problems of physics.

5. Acknowledgements

The ideas in this paper can hardly be considered my invention; I have just been asked to write them down. Stimulating conversations with Stu Feldman, Norm Schryer and Stephen Wolfram helped me get the ideas into a cogent form, and Al Aho, Charlie Harris, Brian Kernighan, and Doug McIlroy made invaluable comments about early manuscripts.

* * *

QUESTIONS

MR. M. METCALF CERN

- Q ---> The proposed curriculum was excellent, but unfortunately the contents would change. Therefore the course would have to be repeated at regular intervals.

- A ---> It is often enough to get physicists started on the right road, then they would be able to look after themselves.

MR. R.BOCK CERN

- COMMENT ---> Certainly most present in the audience would agree with the proposal to educate physicists in 'computer thinking' and to improve the communication between physics and conputer science. However, education happens before physicists become high energy physicists and come to CERN. Also, the speaker is probably preaching to the converted: for every physicist attending the workshop there might be five others who wouldn't accept anybody's help.

MR. D. WILLIAMS CERN

- COMMENT ---> The abstract mentions design of experiments and data reduction. Could the speaker say a few words on the subjects?

- A ---> The speaker replied with a story of a complicated cosmic ray detector which had to be simulated after it was launched into the sky, but did not go into details about the subject.