Ubiquity, an ACM publication
January, 2011

Ubiquity Symposium

What is Computation?

What is the Right Computational Model for Continuous Scientific Problems?

by Joseph Traub,

Columbia University

Editor's Introduction

In this tenth piece to the Ubiquity symposium discussing 'What is computation?' Joseph Traub shares his views about using the Turing Machine model and the real number model for solving continuous scientific problems.

Peter J. Denning
Editor

Association for Computing Machinery
Advancing Computing as a Science & Profession

Ubiquity Symposium

What is Computation?

What is the Right Computational Model for Continuous Scientific Problems?

by Joseph Traub,
Columbia University

I propose to address the question of whether the Turing machine is the right computational model for continuous scientific problems. The Turing Machine model is an abstraction. We know that because the tape is potentially infinite and hence not physically realizable.

The real number model, which is used in scientific computations, is another abstraction. The crux of this model is that we can store real numbers and perform arithmetic operations and comparisons on them exactly and at unit cost. Specifically, the two assumptions of the real number model are:

1. Operations on real numbers can be performed exactly.
2. They can all be performed at unit cost.

I'll discuss these assumptions in turn. Assumption 1 is forced on us because if there were an error associated with each operation, the complexity analysis of the complicated problems of scientific computation would be impossible. The only viable strategy is to identify an optimal algorithm and then follow up with a rounding error analysis—but the follow-up advice is frequently honored in the breach.

The second assumption is not restrictive. Comparisons and the four arithmetic operations all cost about the same in floating point arithmetic on a modern digital computer. The assumption about unit cost is just scaling.

The real number model has a long history. Alexander Ostrowski used it in his work on the computational complexity of polynomial evaluation in 1954. I used it in optimal iteration theory in the early 1960s. Shmuel Winograd and Völker Strassen used the real number model in their work on algebraic complexity in the late 1960s. For these and many more examples, see [2]. Since we cannot store even a single real number on a digital computer, the real number model is clearly an abstraction. For more on the real number model see [2, 4, 5, 6].

So we have two abstractions. Which one should be chosen? That depends on how useful that abstraction is for the purpose at hand. When the purpose is solving continuous scientific problems, the real number model is the better choice. Most of the work on algorithms and computational complexity of continuous scientific problems use the real number model. By continuous scientific problems, I include those occurring in the physical sciences, engineering, parts of computer science (such as vision, animation, and graphics), mathematical finance, and even parts of biology (Anton is a special-purpose supercomputer that works on biological problems such as protein folding, and computes continuous molecular forces [3]). Examples of the kind of problems occurring in these fields include partial differential equations, integration in hundreds of variables, continuous optimization, nonlinear equations, and systems of ordinary differential equations. Typically, these problems cannot be solved exactly; they have to be approximated numerically.

Here are three reasons for the use of the real number model for continuous scientific problems:

1. Most scientific computation uses fixed-precision floating point arithmetic. Analysis using the real number model is predictive of running times for scientific computation. Erich Novak and Henryk Wozniakowski proved this must be the case, modulo a couple of fairly nonrestrictive assumptions; see [2].
2. The real number model permits the full power of continuous mathematics. The argument for using the power of analysis was already made in 1948 by John von Neumann, one of the leading mathematical physicists of the century, and a father of the digital computer. In his Hixon Symposium lecture, von Neumann argues for a “more specifically analytical theory of automata and information.” See, for example, [5].
3. The mathematical models of science are often continuous, and use real (and complex) numbers. That is, scientists assume the continuum. Therefore it seems natural to use the real numbers in analyzing the numerical solution of continuous models. (I want to emphasize that the goal is to obtain an optimal algorithm and obtain the computational complexity of computing an approximate solution to the continuous mathematical model proposed by the scientist.)

Some of my colleagues in theoretical computer science used to say to me that the Turing Machine model was needed for deep questions such as the complexity hierarchy and NP-completeness theory. Then Lenore Blum, Michael Shub, and Steve Smale [1] formalized the real number model, and showed one can ask the same deep questions as with the Turing Machine model. I don’t regard the formalization as very significant; as I said earlier, the real number model has been widely used for decades. Their major achievement was showing the real numbers had the same deep structural questions as were encountered with the Turing machine model. This deprived the Turing Machine model of its unique status.

I’ve given three reasons in favor of using the real number model for continuous scientific problems. I’ll now give two reasons against using the Turing Machine model for scientific computations.

1. Estimated running times on a Turing Machine are not predictive of scientific computations on digital computers. As I’ve said before, scientific computation is usually done with floating point arithmetic. Therefore the cost of arithmetic operations is essentially independent of the size of the operands. On the other hand, the cost of Turing Machine operations depends on operand size.

2. It's often claimed that all models of computation are polynomially equivalent to the Turing Machine model. But consider the example of a random access machine, where multiplication is the basic operation, and memory access and the operations of multiplication and addition can be performed at unit cost. (This is called a UMRAM.) This seems like a reasonable abstraction of a digital computer, since multiplication and addition on floating point numbers cost about the same. But an UMRAM is *not* polynomially equivalent to a Turing Machine! (See, e.g., [7]).

I consider the invention of the Turing Machine by Alan Turing to settle a decidability question in logic as one of the greatest intellectual achievements of the twentieth century. Nonetheless, it is the wrong model of computation for the continuous problems of science.

About the Author:

Joseph F. Traub is the Edwin Howard Armstrong Professor of Computer Science at Columbia University and External Professor, Santa Fe Institute http://cs.columbia.edu/~traub.

References

1. L. Blum, M. Shub, and S. Smale, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions, and universal machines. *Bull. Amer. Math. Soc. 21* (1989), 1–46.

2. E. Novak and H. Wozniakowski. *Tractability of Multivariate Problems, vol. 1, Linear Information.* European Mathematical Society (2008).

3. D. E. Shaw et al. Anton, a special-purpose machine for molecular dynamics simulation. *Commun. ACM 51*, 7 (2008), 91–97.

4. J. F. Traub and H. Wozniakowski. *A General Theory of Optimal Algorithms.* Academic Press (1980).

5. J. F. Traub and A. G. Werschulz. *Complexity and Information*. Cambridge University Press (1998).

6. J. F. Traub, G. Wasilkowski, and H. Wozniakowski. *Information-based Complexity*. Academic Press (1988).

7. P. van Emde Boas. Machine models and simulations, pp. 1–66 of van Leeweun, J., Ed., *Handbook of Theoretical Computer Science: vol. A, Algorithms and Complexity*, MIT Press (1990).