Personal Reflections on 50 Years of Scientific Computing: 1967–2017

Layne T. Watson*
Departments of Computer Science, Mathematics, and
Aerospace and Ocean Engineering
Faculty of Health Sciences
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061 USA

Computer hardware, software, numerical algorithms, and science and engineering applications are traced for a half century from the author's perspective.

Keywords: computer hardware, software, numerical analysis, scientific computing

1. Introduction and Disclaimer

This essay constitutes purely personal and inevitably biased reflections and opinions, with likely faulty memory and a lack of technical references, due to vendor technical hardware and software user manuals for machines that existed for only a couple years (or less) being held mostly in private personal libraries. I feel privileged to have witnessed and participated in computing from close to its beginning, and as Newton said, "If I have seen further, it is by standing on the shoulders of giants." Many of my quotes of professors and colleagues are from my course and research notes, and I apologize if the quotes are not exact.

2. The 1960s

My first exposure to computing was a (quarter long) sophomore level mathematics course (since computer science departments did not exist then) on computing in 1966 at Evansville College, a small private school in southern Indiana. The machine was an IBM 1620 with (modern at the time!) transistors and a magnetic core memory of exactly 20,000 decimal digits. It was programmed in machine language (not assembler!), using two decimal digit op codes, decimal addresses, and fields defined by flagging the content of a memory location. It had no registers and no hardware decimal arithmetic instructions (arithmetic was done by software, storing the multiplication table and doing lookups!). All I/O was via 80-column IBM punched cards, a line printer, and a rotating magnetic disk that only the college administration could access. I had no problem coding with the low level machine language, but struggled mightily with the IBM 026/029 card punches that regularly jammed, mangled cards, and randomly failed to print at the top of the card the characters that were punched in the card. I recall spending more time punching the program onto cards than writing the program!

The next quarter computing course was on numerical methods (more mathematics than computer science), used a very early (NCE FORTRAN) compiler, and more resembled modern computing, with one huge difference — the 20,000 decimal digit memory and the only input device being a card reader. Any serious numerical computation had to be done in stages, with intermediate results punched into cards and then read back in for the next stage. Note that loading all

* Email: ltwatson@computer.org

program stage card decks and intermediate punched result decks into the card reader was done manually, so you had to be a machine operator in addition to a scientist and programmer.

Apparently I had a knack for writing and debugging machine language code, because the computing course instructor (Professor C. W. Buesking) asked me to become a lab assistant for the spring quarter of 1967. That experience led to an engineering co-op (cooperative education, a program whereby you alternated one quarter in school with one quarter working for industry) position in the summer of 1967 with U. S. Naval Ammunition Depot, Crane in Crane, Indiana (really, there is a naval ammunition depot in the center of the state of Indiana, nowhere near an ocean or river). Crane had an early model IBM 360/30 (a modern CISC architecture, base 16) and was getting a Honeywell 2200, and the engineers and accountants were trying to figure out how to use them. Honeywell ran a short course on COBOL, and I mainly worked with IBM FORTRAN and Honeywell COBOL. The machines were restricted, and I had little access to them (other than some number theoretic computations I ran, and managed to justify to my superior as relevant to USNAD Crane).

Computing got much more interesting in the summer of 1968, when I worked for Nathan I. Lieberman, CPA (I remember his 'NIL' ink stamp) writing machine language code for an RPC 4000. This machine had paper tape I/O, 8192 32-bit words of memory, hardware base 2 arithmetic, several registers, and a revolving drum main memory. The interesting challenge on the RPC was to not just get the program code correct, but to place the data and instructions on the drum so as to minimize the rotational latency. My lifelong career in optimization began here, when I realized that a greedy, locally optimal data placement would likely not be globally optimal for the entire program.

Paper tape is considerably worse than cards except in one aspect — tape occupies less physical space per character. The contents of a punch card can be read from the printing at the top (assuming a working card punch!) — paper tape punches do not print on the tape. Assuming sequence numbers are punched into cards (standard procedure, e.g., FORTRAN reserves columns 73–80 for sequence numbers), a dropped card deck can be quickly reassembled by a collating machine. A dropped spool of paper tape (think fishing line tangled in a fishing reel) is, well, a disaster. During my time at NIL, every programmer dropped a 100 ft spool of tape exactly *once* — and after spending the night, on their own time, untangling the 100 ft of paper tape, *never* again. Colored paper tape chad, by the way, makes great wedding confetti — I still have a box of it.

The RPC machine code was for an interpreter called ABLE (Account Balance and Ledger Entry system), which was extremely profitable for NIL. We routinely did quarterly balance sheets and weekly payrolls for companies with 1,000 employees, all with a 8192-word RPC and paper tape data files. When I tell students what can be done with 8K words, they refuse to believe me. Current commercial software, some requiring gigabytes to run, is obscenely bloated.

In 1969, after graduating from Evansville with majors in mathematics and psychology, I worked for another accounting firm (Wright, Griffin, and Davis, CPAs) in Ypsilanti, Michigan while a graduate student in mathematics at The University of Michigan. That work involved the niche IBM System 3, a machine with BCD (binary coded decimal) integer arithmetic, 32-bit words, a rotating disk, line printer, an assembler, and 96-column punched cards about 1/3 the size of IBM 80-column cards for I/O. I led a team (of mathematics graduate students, with no prior experience with computing) to design and build an interpreter called PROFIT (Periodic Reporting Of Financial Input Transactions) with functionality similar to ABLE, including a monitor and all the physical I/O device drivers. The input was on punched cards, the financial reports were printed, and the financial data was stored on a hard disk drive. The disk data structures involved binary

trees, linked lists, and garbage collection, all created by students with no prior knowledge of any of these concepts. Years later, I was contacted about why PROFIT, rewritten in COBOL (which had become available for the System 3 by that time), might be running so much slower than the original assembly language version.

3. The 1970s

Michigan was a numerical analysis graduate student's dream — an IBM 360/67 (later 370/168) with special virtual memory hardware built just for Michigan, the Michigan Terminal System (MTS) operating system (the most sophisticated, user friendly OS I have ever seen — bar none), the Merit network, and star faculty such as Jim Wilkinson, Cleve Moler, Carl de Boor, David Kahaner, and George Fix. The 360/67 took time shared batch job and interactive computing to a new level, and is more impressive in retrospect than it seemed at the time.

MTS was an extremely usable operating system, embodying human-computer interaction (HCI) principles long before HCI was a recognized specialty within computer science and industrial engineering. Many of the most powerful features of Multics and UNIX were foreshadowed in MTS (sophisticated pattern matching, long and short versions of command names and options, scripting and composition of commands, powerful editors and utilities, and on and on). As a scientific computing user of MTS, the feature I liked the best was the "principle of least surprise:" a command or option should do what a reasonably competent user would expect it to do. So when in doubt make a reasonable guess, and 90% of the time in MTS you'd be correct. Contrast this with current commercial software (Microsoft, Adobe, etc.), where 90% of the time a reasonable guess is wrong.

Later (1975), on the faculty at Michigan State University, which had a CDC 6500 (one in a long line of outstanding scientific computing machines — 60-bit floating point numbers, excellent compilers and mathematical libraries), I made extensive use of the Merit network connecting major universities in the state of Michigan. In terms of file access and remote computing, Merit in 1975 was every bit as functional as today's internet. E-mail would come a few years later.

When EISPACK (a precursor of LAPACK) was under development by a consortium of universities and national laboratories, Jim Wilkinson would visit Michigan for a summer school on numerical linear algebra, hosted by Cleve Moler. As a graduate student and aspiring numerical analyst, I attended all of Wilkinson's lectures. At the conclusion of the school, Moler hosted a party for all the attendees at his house. My wife and I, on the way to the party, encountered Wilkinson on the Michigan diag (campus landmark), looking a bit lost. We offered him a ride to the party, and in thanks he gave me an autographed copy of *Handbook for Automatic Computation* that he was using for his lectures. That is one of my most treasured books.

At Michigan, I was privileged to have learned from Wilkinson and Moler, took the core numerical analysis courses from Carl de Boor, studied under David Kahaner, and officially finished the Ph.D. under George Fix (my thesis topic and early supervision actually came from Katta G. Murty, Department of Industrial and Operations Engineering, before Fix arrived). From my notes during those years, I have some quotes that I find especially trenchant. Cleve Moler, on the inverse of a matrix: "there is nothing you can do with the inverse that you can't do better without it." Carl de Boor, on splines: "the only good thing about the natural cubic spline is its name." George Fix, on theorems concerning random matrix properties, in the context of PDEs: "there is no such thing as a random matrix." Larry Shampine, on comparing ODE algorithms: "the implementation of an algorithm is more important than the algorithm itself." Richard Hamming, on everything: "the purpose of computing is insight, not numbers."

At this point in time, numerical analysis and scientific computing were searching for an academic home. Numerical analysis was regarded by traditional mathematics departments as "not real mathematics" and by computer science departments as "too mathematical." Moving from the cold north to the old south, I found a welcoming Department of Computer Science in Blacksburg. Virginia Polytechnic Institute & State University (now known as Virginia Tech for publicity purposes) in 1978 had an Amdahl 470 (IBM 360 clone) and only punched card batch access (it wasn't Michigan), but sufficient to do meaningful solid and fluid mechanics applications with numerous engineering and mathematics collaborators.

4. The 1980s

Things improved quickly at VPI&SU; the Amdahl was replaced by a Honeywell 68/60 running Multics, enabling interactive computing from my office via a DECwriter terminal (132 column fan feed paper, and a dot matrix print head). I was still using this DECwriter as a printer in 2000, and gave it up only because the ink ribbons were no longer available. This decade included a severe economic recession, and disruptive computing technologies: personal computers and minicomputers. The interaction of the recession and the new technologies had profound consequences.

The first VAX 11/780 minicomputer at VPI&SU went to the Department of Geology to run canned software, the next one to Computer Science, and the third one to the Spatial Data Analysis Laboratory. The latter two ran DECnet between them for the first computer network (1980) at VPI&SU. Getting approval from the university to run this DECnet cable through the campus steam tunnels was an eye-opening experience in how entrenched bureaucracies respond to new technology. Suddenly VAXes and other minicomputers were sprouting like weeds, and the central campus computing center saw an elegant solution to their budget crisis: computing would henceforth be "decentralized" with departments, colleges, and laboratories running their own minicomputers, and the center supporting only administrative computing and one token VAX 11/780. Concurrent with this was the adoption of a "personal computer requirement" for computer science and engineering incoming freshmen, essentially forcing them to provide their own computing resources (VPI&SU was the first public university to have such a requirement).

The profound result of this confluence, replicated throughout the country, was that now scientific computing either had to be done on personal computers, minicomputers, or at national supercomputing centers (requiring competitive proposals). Once the "big iron" was gone locally, it would take a long time for it to return to university computing centers. Ironically, "small" scientific computing (a la Matlab, Mathematica) got easier, but "large" scientific computing got much harder (unless your laboratory or company had a supercomputer).

This decade also saw the emergence of parallel computing in almost every conceivable flavor, a truly exciting period for computer and computational scientists. Alas, the market could not sustain this cornucopia of parallel architectures, and almost all the companies and/or products failed. I lost this decade working on the Elxsi 6400 (the first parallel machine at Sandia National Laboratories in 1984), Intel iPSC-32 (the "hypercube", serial number 2, at General Motors Research Laboratories in Warren, Michigan), NCUBE-10 (the company sold numerous machines but went out of business without ever producing a working FORTRAN compiler), Sequent Balance 21000, Sequent Symmetry S81, Cray 2, Cray Y-MP C90, Intel Paragon XP/S, SGI Origin 2000. Despite the commercial failure of parallel computing, the community learned a lot about parallel programming, interconnection topologies, communication technologies, and the tradeoffs between

shared and distributed memory systems. The community also learned that parallel programming was, and still is, *very* hard.

5. The 1990s

After the hardware architecture wild west of the 80s, mainframe hardware evolution settled down to conventional, incremental, and predictable. Computing has always combined hardware, software/algorithms, and the human usability of those two for productivity. Usability and scientific productivity saw quantum leaps in the 1990s. First, there were notable advances in numerical algorithms and mathematical software. Homotopy algorithms for solving nonlinear systems of equations, using new differential geometry theory, were widely applied in science and engineering, e.g., aerospace engineering, mechanical engineering, systems biology, chemistry, solid mechanics, fluid mechanics, computer vision, robotics, material science, and many other disciplines. The deep mathematics, and computational experience of numerical analysts, were made accessible and usable via mathematical software (HOMPACK90, 1997, [11]). New algorithms and software tools became available for multidisciplinary design optimization (MDO), which fundamentally changed the way large systems (aircraft, ships, automobiles) were designed [3]. By definition, MDO requires supercomputing, and MDO became both a user and a driver of high performance computing. There were major advances with new approaches to old problems: H_2 control theory (versus the standard H_∞ approach) [12], wavelets for signal processing [4], the DIRECT algorithm for global optimization (Don Jones, 1993) [8, 5, 6, 7].

Perhaps the largest strides occurred in improving the usability of computing for scientists and engineers, through problem solving environments (PSEs), related to older concepts such as expert systems, decision support systems, automated assistants, and recommender systems. A PSE is narrowly domain specific, employs HCI principles for the user interface, and often transparently uses distributed or supercomputing for the solution of problems [10]. (Technically Matlab and Mathematica are PSEs for mathematical models already formulated, and do not assist in the *formulation* of those models.) Examples of PSEs at VPI&SU alone include Vizcraft (high performance aircraft design), WBCSim (analysis and design of wood based composite materials), L2W (analysis of the effects of land use changes on watersheds and wildlife: involved computer science, hydrology, economics, biology, ecology, urban planning, civil engineering) [10], JigCell (chemical kinetic modeling of biological systems, e.g., the eukaryotic cell cycle) [1]. The advance in PSEs is having systems that match the users' terminology, thought processes, and workflow, rather than forcing the user to learn "how the computer works."

While distributed computing is as old as networking (DEC famously advertised that with the VAX and DECnet, "the network is the computer"), new concepts and technologies emerged, generically called grid computing [9]. The new technologies were both high level (e.g., automatic acquisition of a needed service) and low level (e.g., remote authentication), and intended to expand computing capability (e.g., solve a larger problem) or improve reliability (e.g., provide redundant sources for some service) or increase machine utilization (e.g., by routing a task to a lightly loaded system). The major players in grid computing were Globus (Argonne National Laboratory), Legion (University of Virginia), Condor (University of Wisconsin), and SORCER (National Bureau of Standards, now known as the National Institute of Science and Technology).

Last but not least, and in the same vein of usability, comes HTML and the World Wide Web (WWW, 1990). Certainly computational scientists extensively use the WWW, but whether the WWW has significantly improved computational science is moot. Computer e-mail, from the

previous decade, and the open internet (circa 1990) for file transfer and remote login have more directly aided scientific computing collaboration.

6. The 2000s

The ballyhooed Y2K disaster (when all the world's computer databases would turn into pumpkins at the stroke of midnight marking the turn of the century) never happened. However, the .com bust and ensuing deep recession, a real life version of the tale of Icarus, did happen. In 2003, an NSF equipment grant to Srinidhi Varadarajan, myself, and others was leveraged, in a rare confluence of departmental, college, and university priorities, to build System X ("ten," for ten teraflops, and for running Apple OS/X). System X consisted of 2200 Apple G5 processors, connected with Infiniband, and running OS/X. At 12.5 teraflops, System X came in at number three in the 2003 world Top 500 computer rankings, behind two national laboratory machines that cost at least three times as much. After a story in the New York Times about System X, and some national laboratory managers were asked to justify the budgeted cost of their proposed comparable new machines, there was a shocking backlash against VPI&SU's accomplishment. I was in the audience at a conference at Oak Ridge National Laboratory when a speaker from a national laboratory (not ORNL) said System X was built by "pizza fueled undergraduates," was "a fake," and "not a serious machine." I assured the speaker that System X was real and being used for serious work, and invited him to come up the road to Blacksburg and see for himself. He did not respond.

System X was the beginning of the return of "big iron" to university campuses, but the return was slowed by another deep recession and depressed computer science and STEM enrollments. Computer companies that were once the heart of the industry went under, bought up for their reputation but not their product research and development. DEC, SUN, and SGI disappeared. Computer chip manufacturing underwent a similar global consolidation.

As the end of Moore's Law appeared nearer and nearer, chip manufacturers considered other materials and other manufacturing processes, and computer architects pursued parallelism to maintain the steady increase in performance. Multicore (2, 4, 8 cores) architectures (shared memory parallelism) with compiler support became common, although even a modest speedup of seven with eight cores on a general purpose and highly variable instruction stream was rarely achieved. Everyone knew this day would likely come (the end of increases in single thread performance), but now it was uncomfortably close.

An emerging major consumer of supercomputing was bioinformatics (the intersection of some parts of computer science with some parts of biology), as academic departments hired bioinformaticians and universities created bioinformatics research centers (the Virginia Bioinformatics Institute at VPI&SU was one of the first such centers, built, ironically, with Tobacco Master Settlement Agreement money). The mix of scientific computing jobs run on computing center machines began to shift from almost exclusively the physical sciences and engineering toward the life sciences (biology, bioinformatics, medicine, pharmacology) and data analysis from all disciplines, including business and the social sciences.

7. The 2010s

The current frontier is exascale computing, but the path is far from certain. Exascale power requirements, and power consumption of large machines in general, have become major issues. Both automated power management systems, based on data mining and profiling, and individual programs controlling their own power usage are being studied. Chip manufacturers are increasing the number of cores (128, so far) on a single chip, bringing shared memory programming to the

forefront again. Distributed memory programming and communication topologies are still the backbone of high performance scientific computing.

As computing systems move toward exascale, the mean time to failure (MTF) of some component becomes less than the run time of a typical scientific computation, making hardware/software resilience a *sine qua non* for production computing at the exascale. As the complexity of scientific computations grows, and the inherent variability of hardware performance also grows (the latter has been confirmed recently by a NSF CSR project), the reproducibility of both scientific results and system (hardware, operating system, application code) behavior becomes a serious concern, affecting the confidence in the very computational science enterprise itself.

Large segments of existing computer science, statistics, mathematics, and electrical engineering have been co-opted under the name of "machine learning" (ML, also known as "data mining") to analyze "big data;" presumably all analyzed data heretofore had been "small." More erudite sounding terms for the general activity are "data analytics" and "knowledge discovery," as if finding a pattern in data constitutes an explanation or understanding. As the statisticians say, "correlation does not imply causality." One is reminded of genetic algorithms, evolutionary computation, and "biologically inspired" algorithms, as if a mere analogy with something biological confers superiority. Nature is, in fact, often far from optimal, as any biologist will attest. To paraphrase my earlier de Boor quote: "The only thing good about genetic algorithms or artificial neural networks is their name."

Many phenomena that are stochastic are modeled as deterministic (e.g., chemical reactions by ordinary differential equations), or a stochastic quantity is replaced by a deterministic functional of that quantity (e.g., a random variable by its mean), for theoretical or computational convenience. Theoretical and algorithmic progress has been made in dealing directly with the stochastic quantities without assuming something is deterministic. MDO under uncertainty (known as robust design) or stochastic systems biology problems (e.g., parameter estimation for a stochastic cell cycle model) can now be attacked directly without making ensembles of runs to estimate a deterministic functional (e.g., a variable's mean). One such new algorithm, supported by rigorous convergence theory, is called QNSTOP (quasi-Newton stochastic optimization), for which high quality mathematical software exists [2]. Similar important advances have been made in the theory and numerical solution of stochastic ordinary and partial differential equations, which are often more faithful representations of reality.

6. Epilogue

These past 50 years of scientific computing have been quite a run, and I wish the following generations an equally exciting and enjoyable 50 years. Pergite Veneti!

References

[1] N. A. Allen, C. A. Shaffer, M. T. Vass, N. Ramakrishnan, and L. T. Watson, *Improving the development process for eukaryotic cell cycle models with a modeling support environment*. Simulation 79 (2003), pp. 674–688.

[2] B. D. Amos, D. R. Easterling, L. T. Watson, W. I. Thacker, B. S. Castle, and M. W. Trosset, *Algorithm XXX: QNSTOP—quasi-Newton algorithm for stochastic optimization*. Technical Report TR-14-02, Dept. of Computer Sci., VPI & SU, Blacksburg, VA, 2014.

[3] S. Burgee, A. A. Giunta, V. Balabanov, B. Grossman, W. H. Mason, R. Narducci, R. T. Haftka, and L. T. Watson, *A coarse-grained parallel variable-complexity multidisciplinary optimization paradigm*. Internat. J. Supercomputer Appl. High Performance Comput. 10 (1996), pp. 269–299.

[4] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia, 1992.
[5] J. He, A. Verstak, L.T. Watson, and M. Sosonkina, *Performance modeling and analysis of a massively parallel DIRECT: Part 1*. Internat. J. High Performance Comput. Appl. 23(1) (2009), pp. 14–28.
[6] J. He, A. Verstak, M. Sosonkina, and L.T. Watson, *Performance modeling and analysis of a massively parallel DIRECT: Part 2*. Internat. J. High Performance Comput. Appl. 23(1) (2009), pp. 29–41.
[7] J. He, L.T. Watson, and M. Sosonkina, *Algorithm 897: VTDIRECT95: Serial and parallel codes for the global optimization algorithm DIRECT*. ACM Trans. Math. Software 36(3) (2009), Art. 17, pp. 1–24.
[8] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, *Lipschitzian optimization without the Lipschitz constant*. J. Optim. Theory Appl. 79(1) (1993), pp. 157–181.
[9] N. Ramakrishnan, L. T. Watson, D. G. Kafura, C. J. Ribbens, and C. A. Shaffer, *Programming environments for multidisciplinary grid communities*. Concurrency Comput.: Pract. Exper. 14 (2002), pp. 1241–1273.
[10] L. T. Watson, V. K. Lohani, D. F. Kibler, R. L. Dymond, N. Ramakrishnan, and C. A. Shaffer, *Integrated computing environments for watershed management*. J. Comput. Civil Engrg. 16 (2002), pp. 259–268.
[11] L. T. Watson, M. Sosonkina, R. C. Melville, A. P. Morgan, and H. F. Walker, *Algorithm 777: HOMPACK90: A suite of Fortran 90 codes for globally convergent homotopy algorithms*. ACM Trans. Math. Software 23 (1997), pp. 514–549.
[12] D. Žigić, L. T. Watson, E. G. Collins, Jr., and D. S. Bernstein, *Homotopy approaches to the H_2 reduced order model problem*. J. Math. Systems, Estimation, Control 3 (1993), pp. 173–205.