URL: http://www.physicstoday.org/vol-58/iss-1/p35.shtml Published: January 2005

ARTICLES

Computational Science Demands a New Paradigm

The field has reached a threshold at which better organization becomes crucial. New methods of verifying and validating complex codes are mandatory if computational science is to fulfill its promise for science and society.

Douglass E. Post and Lawrence G. Vott

Computers have become indispensable to scientific research. They are essential for collecting and analyzing experimental data, and they have largely replaced pencil and paper as the theorist's main tool. Computers let theorists extend their studies of physical, chemical, and biological systems by solving difficult nonlinear problems in magnetohydrodynamics; atomic, molecular, and nuclear structure; fluid turbulence; shock hydrodynamics; and cosmological structure formation.

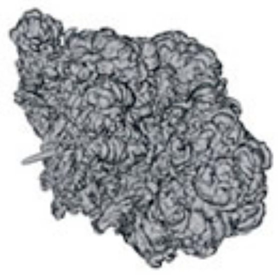


Figure 1

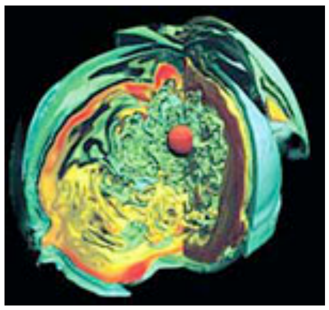


Figure 2

Beyond such well-established aids to theorists and experimenters, the exponential growth of computer power is now launching the new field of computational science. Multidisciplinary computational teams are beginning to develop large-scale predictive simulations of highly complex technical problems. Large-scale codes have been created to simulate, with unprecedented fidelity, phenomena such as supernova explosions (see figures 1 and 2), inertial-confinement fusion, nuclear explosions (see the box on page 38), asteroid impacts (figure 3), and the effect of space weather on Earth's magnetosphere (figure 4).

Computational simulation has the potential to join theory and experiment as a third powerful research methodology. Although, as figures 1–4 show, the new discipline is already yielding important and exciting results, it is also becoming all too clear that much of computational science is still troublingly immature. We point out three distinct challenges that computational science must meet if it is to fulfill its potential and take its place as a fully mature partner of theory and experiment:

- *the performance challenge*—producing high-performance computers,

- *the programming challenge*—programming for complex computers, and
- *the prediction challenge*—developing truly predictive complex application codes.

The performance challenge requires that the exponential growth of computer performance continue, yielding ever larger memories and faster processing. The programming challenge involves the writing of codes that can efficiently exploit the capacities of the increasingly complex computers. The prediction challenge is to use all that computing power to provide answers reliable enough to form the basis for important decisions.

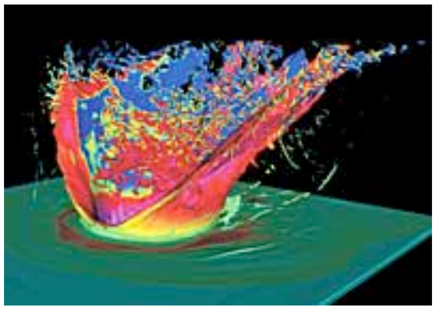


Figure 3

The performance challenge is being met, at least for the next 10 years. Processor speed continues to increase, and massive parallelization is augmenting that speed, albeit at the cost of increasingly complex computer architectures. Massively parallel computers with thousands of processors are becoming widely available at relatively low cost, and larger ones are being developed.

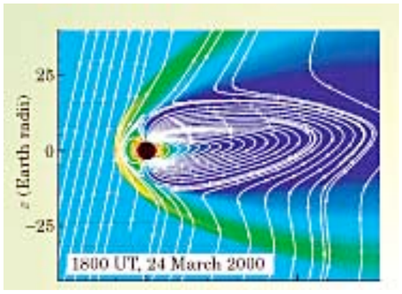


Figure 4

Much remains to be done to meet the programming challenge. But computer scientists are beginning to develop languages and software tools to facilitate programming for massively parallel computers.

The most urgent challenge

The prediction challenge is now the most serious limiting factor for computational science. The field is in transition from modest codes developed by small teams to much more complex programs, developed over many years by large teams, that incorporate many strongly coupled effects spanning wide ranges of spatial and temporal scales. The prediction challenge is due to the complexity of the newer codes, and the problem of integrating the efforts of large teams. This often results in codes that are not sufficiently reliable and credible to be the basis of important decisions facing society. The growth of code size and complexity, and its attendant problems, bears some resemblance to the transition from small to large scale by experimental physics in the decades after World War II.

A comparative case study of six large-scale scientific code projects, by Richard Kendall and one of us (Post),[1] has yielded three important lessons. Verification, validation, and quality management, we found, are all crucial to the success of a large-scale code-writing project. Although some computational science projects—those illustrated by figures 1–4, for example—stress all three requirements, many other current and planned projects give them insufficient attention. In the absence of any one of those requirements, one doesn't have the assurance of independent assessment, confirmation, and repeatability of results. Because it's impossible to judge the validity of such results, they often have little credibility and no impact.

Part of the problem is simply that it's hard to decide whether a code result is right or wrong. Our experience as referees and editors tells us that the peer review process in computational science generally doesn't provide as effective a filter as it does for experiment or theory. Many things that a referee cannot detect could be wrong with a computational-science paper. The code could have hidden defects, it might be applying algorithms improperly, or its spatial or temporal resolution might be inappropriately coarse.

The few existing studies of error levels in scientific computer codes indicate that the defect rate is about seven faults per 1000 lines of Fortran.[2] That's consistent with fault rates for other complex codes in areas as diverse as computer operating systems and real-time switching. Even if a code has few faults, its models and equations could be inadequate or wrong. As theorist Robert Laughlin puts it, "One generally can't get the right answer with the wrong equations."[3]It's also possible that the physical data used in the code are inaccurate or have inadequate resolution. Or perhaps someone who uses the code doesn't know how to set up and run the problem properly or how to interpret the results.

The existing peer review process for computational science is not effective. Seldom can a referee reproduce a paper's result. Generally a referee can only subject a paper to a series of fairly weak plausibility checks: Is the paper consistent with known physical laws? Is the author a reputable scientist? Referees of traditional theoretical and experimental papers place some reliance on such plausibility checks, but not nearly to the degree a computational-science referee must. The plausibility checks are, in fact, sometimes worse than inadequate. They can discriminate against new and innovative contributions in favor of modest extensions of prior work.

Lessons from the past

Henry Petroski has described the history of a number of technological fields as they mature.[4]From his study, we use the example of suspension-bridge technology to identify four stages through which engineering technologies typically pass on their way to maturity (see figure 5). The first stage, in this case, involved adapting the basic concept of the ancient rope suspension bridges into sound engineering designs with modern materials. Design limits were initially not well known, and the first modern designs were conservative. An example is the Széchényi chain bridge that spanned the Danube between Buda and Pest in 1849. Although the retreating Germans blew it up in 1945, the bridge has since been rebuilt according to its original design, and it's still in use.

Figure 5

The second stage involved cautious design improvements based on experience gained during the first generation. The Brooklyn Bridge, opened in 1883, nowadays carries a modern traffic load. In the third stage, engineers pushed the limits of the existing technologies until major failures occurred. A notorious cautionary example is the 1940 collapse of the newly built Tacoma Narrows Bridge, forever after known as "Galloping Gertie."

The civil engineering community studied and analyzed the causes for such failures and developed solutions that then became part of the design methodology for all future suspension bridges. The fourth, mature stage is solidly based on the development and adoption of the lessons learned from past failures and successes. The Akashi Kaikyo Bridge in Japan, completed in 1998, is an example of the very big suspension bridges now being built. Four kilometers long, it spans the Akashi strait near Kobe, a region prone to severe earthquakes.

We assert that computational science is currently in the midst of the third, rather precarious stage of this paradigm. The field began in the early 1950s and moved through the first and second stages during the next 30 years. The pioneering computational scientists had strong backgrounds in the disciplines that were the subjects of their computations. So they retained a healthy skepticism about computational results and a sound perspective on what was correct and what wasn't. At that stage, therefore, computational predictions were usually thoroughly checked and validated.

By the mid-1990s, computing power had reached the point where it became possible to develop codes that integrate many competing and strongly interacting effects. Today the computational-science community is learning how to create and apply such ambitious codes, but not always successfully.

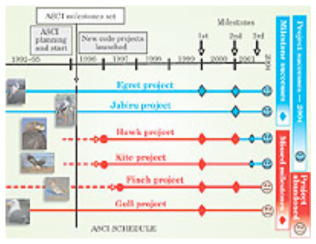
Examples abound of large-scale software failures in fields like information technology (IT) and aerospace. The 1995 failure of the European Space Organization's Arianne 5 rocket and the 1999 loss of NASA's *Mars Climate Orbiter* are still fresh in memory.[5] After the *Columbia* space shuttle's ill-fated February 2003 launch and first reports of possible problems with the mission, a NASA– Boeing team's computational assessment of potential failure modes yielded misleading conclusions that may have contributed to the tragedy.[6]

The quest for fusion energy provides two more examples of problematic computation. By stretching boundary conditions far beyond what could be scientifically justified, computer simulations were able to "reproduce" the exciting but wrong experimental discovery[7] of sonoluminescent fusion (see Physics Today, April 2002, page 17). With regard to the International Thermonuclear Experimental Reactor (ITER), preliminary computational predictions in 1996 of inadequate performance by the proposed facility were wrongly characterized as definitive.[8]Those predictions contributed to the 1998 US withdrawal from that important and promising international undertaking. The US is now seeking to rejoin ITER.

Although computational science is beginning to play an important role in society, those examples illustrate that the prediction efforts are not always successful. To fulfill its promise, computational science has to do better. It must achieve a level of maturity at which its results are as accurate and reliable as those of traditional theoretical and experimental science. As was the case with suspension bridges, this maturation must include retrospective analysis of the community's failures and successes, and adoption of the lessons they teach. The experience of the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) projects, as summarized in the box on page 38, provides detailed evidence that strong emphasis on verification, validation, and software project management is essential for success.

Verification and validation

A computational simulation is only a model of physical reality. Such models may not accurately reflect the phenomena of interest. By verification we mean the determination that the code solves the chosen model correctly. Validation, on the other hand, is the determination that the model itself captures the essential physical phenomena with adequate fidelity. Without adequate verification *and* validation, computational results are not credible. From the difficulties that ASCI and other computational projects have faced in trying to do comprehensive verification and validation, it is evident that present efforts toward those ends fall far short of what's needed.



Box 1

The bigger and more complex the code, the harder it is to verify and validate. A sophisticated climate-modeling code might include models for 50 or more effects—for example, ocean evaporation, currents, salinity, seasonal albedo variation, and so forth. The code would predict observables such as mean surface temperature and precipitation levels. The accuracy of the predictions depends on the validity of each component model, the completeness of the set of all the models, the veracity of the solution method, the interaction between component models, the quality of the input data, the grid resolution, and the user's ability to correctly set up the problem, run it, and interpret the results.

In practice, one must first verify and validate each component, and then do the same for progressively larger ensembles of interacting components until the entire integrated code has been verified and validated for the problem regimes of interest.[9]

Here we list five common verification techniques, each having limited effectiveness:

- Comparing code results to a related problem with an exact answer
- Establishing that the convergence rate of the truncation error with changing grid spacing is consistent with expectations
- Comparing calculated with expected results for a problem specially manufactured to test the code
- Monitoring conserved quantities and parameters, preservation of symmetry properties, and other easily predictable outcomes
- Benchmarking—that is, comparing results with those from existing codes that can calculate similar problems.

Diligent code developers generally do as much verification as they think feasible. They are also alert to suspicious results. But diligence and alertness are far from a guarantee that the code is free of defects. Better verification techniques are desperately needed.

Once a code has been verified, the next step is validation. Verification must precede validation. If a code is unverified, any agreement between its results and experimental data is likely to be fortuitous. A code cannot be validated for all possible applications, only for specific regimes and problems. Then one must estimate its validity for adjacent regimes.

One has to validate the entire calculational system—including user, computer system, problem setup, running, and results analysis. All those elements are important. An inexperienced user can easily get wrong answers out of a good code in a validated regime.

The need to validate codes presents a number of challenges. Various types of experimental data are used for validation:

- Passive observations of physical events—for example, weather or supernovae
- Controlled experiments designed to investigate specific physics or engineering principles—for example, nuclear reactions or spectroscopy
- Experiments designed to certify the performance of a physical component or system—for example, full-scale wind tunnels
- Experiments specifically designed to validate code calculations—for example, laser-fusion facilities.[10]

Many codes, such as those that predict weather, have to rely mainly on passive observations. Others can avail themselves of controlled experiments. The latter make for more credible validation. The most effective validation method involves comparing predictions made before a validation experiment with the subsequent data from that experiment. The experiments can be designed to test specific elements of a code. Such experiments can often be relatively simple and inexpensive.

Successful prediction before a validation experiment is a better test than successful reproduction after the fact, because agreement is too often achieved by tuning a code to reproduce what's already known. And prediction is generally the code's raison d'être. The National Ignition Facility at Livermore is an example of a validation facility for the ASCI codes.

A paradigm shift

The scientific community needs a paradigm shift with regard to validation experiments. Experimenters and funding agencies understand the value of experiments designed to explore new scientific phenomena, test theories, or examine the performance of design components. But few appreciate the value of experiments explicitly conducted for code validation. Even when experimenters are interested in validating a code, few mechanisms exist for funding such an experiment. It's essential that the scientific community provide support for code-validation experiments.

Verification and validation establish the credibility of code predictions. Therefore, it's very important to have a written record of verification and validation results. In fact, a validation activity should be organized like a project, with goals and requirements, a plan, resources, a schedule, and a documented record. At present, few computational- science projects practice systematic verification or validation along those lines. Almost none have dedicated experimental programs. Without such programs, computational science will never be credible.

Large-scale computer-simulation projects face many of the challenges that have historically confronted large-scale experimental undertakings.[11] Both require extensive planning, strong leadership, effective teams, clear goals, schedules, and adequate resources. Both must meet budget and schedule constraints, and both need adequate flexibility and contingency provisions to adjust to changing requirements and the unexpected.

Using a code to address a technical issue is similar to conducting an experiment. Project leaders require a variety of skills. They need a coherent vision of the project, a good technical overview of all its components, and a sense of what's practical and achievable. They must be able to command the respect of the team and the trust of management. Sometimes they must make technical decisions whose correctness will not be apparent for years. They have to estimate needed resources, make realistic schedules, anticipate changing needs, develop and nurture the team, and shield it from unreasonable demands.

Scientists with such an array of talents are relatively rare. But without them, few code projects succeed. We have found that almost every successful project has been carried out by a highly competent team led by one or two scientists with those qualities. Examples are numerous. The ASCI case study[1] demonstrates in detail that good software-project management is essential.

Scientific-software specialists can learn much from the broader IT community.[12] The IT community has had to address the problem of planning and coordinating the activities of large numbers of programmers writing fairly complex software. But, as we survey the emerging computational-science community, we find that few of even the simplest and best-known proven methods for organizing and managing code- development teams are being employed. The most common approach is the painful rediscovery of lessons already learned by others. Unfortunately, that approach leads to wasted effort and sometimes failure.

Software quality is another key issue. Improvements in quality promise easier maintenance and greater longevity. It's not enough to test a code at the end of its development. Quality has to be built in at every stage. Lack of attention to quality at every manufacturing step nearly destroyed the US automobile industry in the 1970s and 1980s. Many techniques developed by the IT industry for improving software quality are applicable to scientific computing. But each technique must be individually evaluated to match costs and benefits to project goals. It would be counterproductive, for example, to dogmatically apply the rigorous quality-assurance processes necessary for mission-critical software—where one bug could crash a plane—to the development of scientific codes that thrive on risky innovation and experimentation.

The way forward

Computational science has the potential to be an important tool for scientific research. It also promises to provide guidance on key public-policy issues such as global warming.

The field is meeting the performance challenge with computers of unprecedented power. It is addressing the programming challenge. But programming for large, massively parallel computers remains formidable. To meet the prediction challenge, however, the computational-science community must achieve the maturity of the older disciplines of theoretical and experimental science and traditional engineering design.

All the essential tasks for reaching that maturity require attention to the process of code development. Such projects must be organized as carefully as experimental projects are. But, as we have learned from the experience in the IT industry and from the detailed ASCI case study, good software-project management by team leaders is not enough. Even well-organized projects will not succeed if sponsoring institutions do not provide appropriate goals, schedules, and support.

Like the bridge builders, computational scientists can learn from the experience of predecessors. Case studies are an essential part of the path toward maturity. The findings of such studies must be freely available to the whole community. Computational scientists must look back and assess their failures and successes. Otherwise, they are doomed to repeat the mistakes of the past. And they will never fulfill the tremendous promise that powerful computers offer them.

We are leading a team that's developing a body of case studies from academia, industry, and a complex of federal departments and agencies as part of the High Productivity Computing Systems program of DARPA—the Defense Advanced Research Projects Agency. That program addresses all three challenges defined at the beginning of this article. Among the DARPA program's goals are developing a 2-petaflop (2×10^{15} floating-point operations per second) computer by 2010, improving high-performance computing software infrastructure, and increasing scientific research productivity. The ASCI case study[1]was the program's first case study effort.

When computational science can consistently make credible predictions, society will have acquired a powerful methodology for addressing many of theits pressing problems.

Douglass Post is a computational physicist at Los Alamos National Laboratory and an associate editor-in chief of Computing in Science and Engineering. **Lawrence Votta** is a Distinguished Engineer at Sun Microsystems Inc in Menlo Park, California. He has been an associate editor of *IEEE Transactions on Software Engineering*.

References

1. D. E. Post, R. P. Kendall, *Internat. J. High Perf. Comput. Appl.* **18**(4), 399 (2004).
2. L. Hatton, A. Roberts, *IEEE Trans. Software Eng.* **20**, 785 (1994).
3. R. Laughlin, *Comput. Sci. Eng.* **4**(3), 27 (2002).
4. H. Petroski, *Design Paradigms: Case Histories of Error and Judgment in Engineering*, Cambridge U. Press, New York (1994).
5. R. L. Glass, *Software Runaways: Monumental Software Disasters*, Prentice Hall, Upper Saddle River, NJ (1998); J. Jezequel, M. Meyer, *Computer* **30**(1), 129 (1997); A. G. Stephenson et al., *Mars Climate Orbiter Mishap Investigation Phase I Report*, NASA,

Washington, DC (1999), available at http://klabs.org/richcontent/Reports/MCO_report.pdf.
6. H. W. Gehman et al., *Report of* Columbia *Accident Investigation Board*, NASA, Washington, DC (2003), available at http://www.nasa.gov/columbia/home/CAIB_Vol1.html.
7. R. P. Taleyarkhan et al., *Science* **295**, 1868 (2002) [INSPEC]; D. Shapira, M. Saltmarsh, *Phys. Rev. Lett.* **89**, 104302 (2002) [SPIN].
8. J. Glantz, *Science* **274**, 1600 (1996) [CAS].
9. P. J. Roache, *Verification and Validation in Computational Science and Engineering*, Hermosa, Albuquerque, NM (1998).
10. T. Trucano, D. E. Post, *Comput. Sci. Eng.* **6**(5), 8 (2004).
11. S. Traweek, *Beamtimes and Lifetimes: The World of High Energy Physicists*, Harvard U. Press, Cambridge, MA (1988).
12. T. DeMarco, *The Deadline*, Dorset House, New York (1997); R. Thomsett, *Radical Project Management*, Prentice Hall, Upper Saddle River, NJ (2002).
13. K. Ewusi-Mensah, *Software Development Failures: Anatomy of Abandoned Projects*, MIT Press, Cambridge, MA (2003).
14. T. DeMarco, T. Lister, *Waltzing with Bears: Managing Risk on Software Projects*, Dorset House, New York (2003).
15. A. C. Calder et al., http://arXiv.org/abs/astro-ph/0405162.
16. J. M. Blondin, A. Mezzacappa, C. DeMarino, *Astrophys. J.* **584**, 971 (2003) [SPIN].
17. G. Gisler et al., *Comput. Sci. Eng.* **6**(3), 46 (2004).
18. T. I. Gombosi et al., *Comput. Sci. Eng.* **6**(2), 14 (2004).

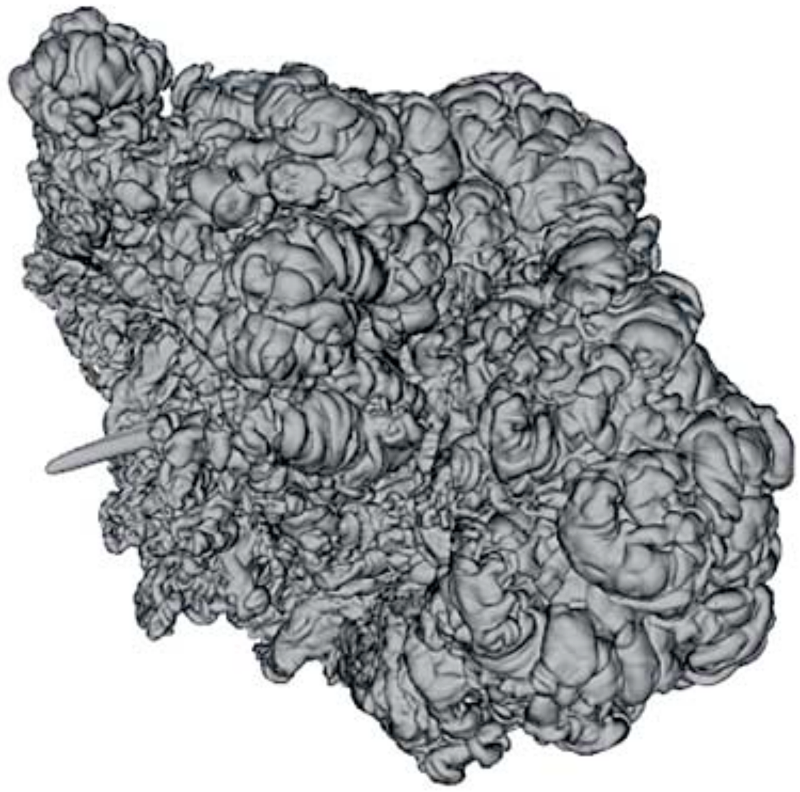


Figure 1. Just before a white dwarf blows up as a type Ia supernova, the volume inside the turbulent, expanding burn front shown in this simulation of the star's interior contains hot carbon– oxygen fusion products created in the fraction of a second since ignition at an off-center point near the fingerlike structure at left. Outside the front is the 96% of the star's C and O still unburnt. A few seconds later, the front may break through the star's surface and spread over it, giving birth to the supernova. The simulation was done at the University of Chicago's ASCI Flash Center. (Adapted from ref. 15.)

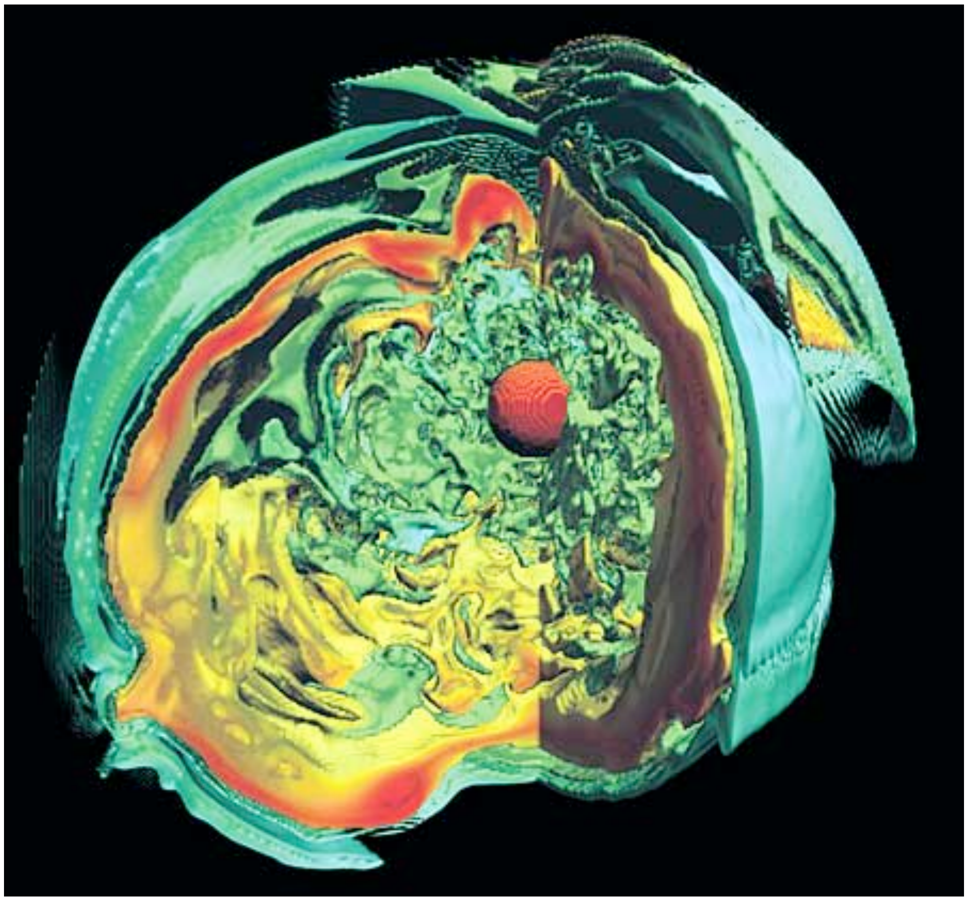


Figure 2. For a star 10 times heavier than the Sun, about to explode as a type II supernova, the collapse of the star's iron core creates a hot fluid of nucleons. This model simulation shows the instability and turbulent flow of the hot nucleon fluid a fraction of a second after core collapse. Colors show the fluid's entropy: Red indicates the highest-entropy, hottest fluid; cooler, lower-entropy fluid is shown green. The outer surface of the nucleon fluid is just inside the expanding shock front that formed when the collapsing core rebounded off the superdense proto-neutron star (the small blue sphere). At the instant shown, the front is stalled at a radius of about 150 km. Reenergized by fluid instability, neutrino flux, and other mechanisms, the shock front will reach the star's surface in a few hours, thus creating the visible supernova. This image is based on a three-dimensional simulation[16]performed at Oak Ridge National Laboratory's Center for Computational Sciences.

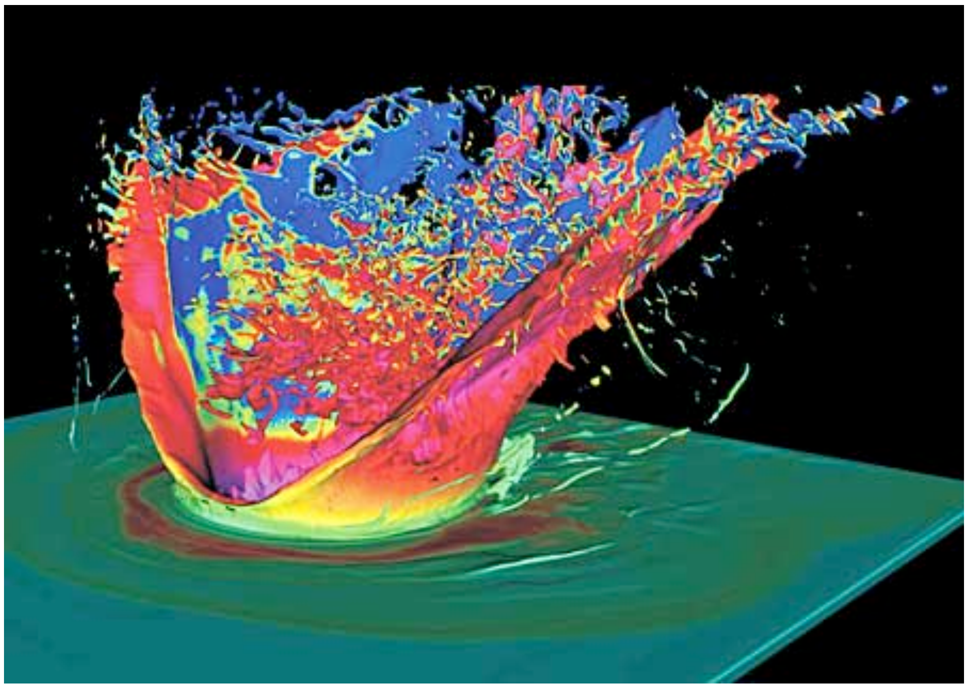


Figure 3. Simulated impact of the 10-km- diameter asteroid that struck the Yucatan peninsula 65 million years ago and presumably triggered the worldwide extinction of the dinosaurs and many other taxa. Shown here 42 seconds after impact, the expanding column of debris from the asteroid and crater is about 100 km high. Colors indicate temperature: The hottest material (red) is at about 6000 K, and the coolest (blue) has returned to ambient temperature. The simulation uses the SAGE code developed at Los Alamos National Laboratory. (Adapted from ref. 17.)

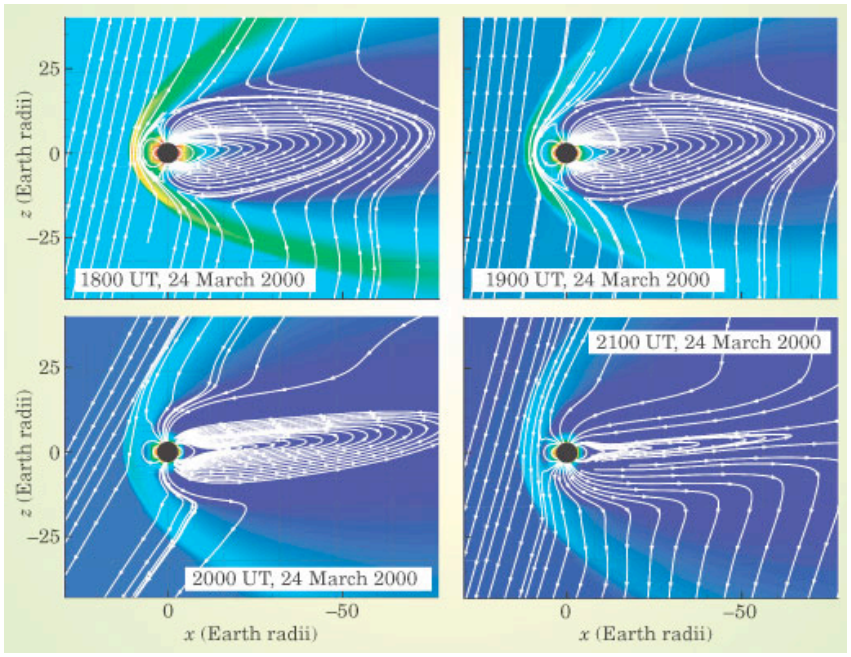


Figure 4. Effect on Earth's magnetosphere of the arrival of plasma and associated magnetic field from a strong mass ejection off the Sun's corona three days earlier,

simulated by a group at the University of Michigan. Color intensity indicates pressure and the white lines are projections of the magnetic field on the plane that cuts Earth's noon and midnight meridians. The *x*axis marks the direction of the Sun, off to the left. (Adapted from ref. 18.)

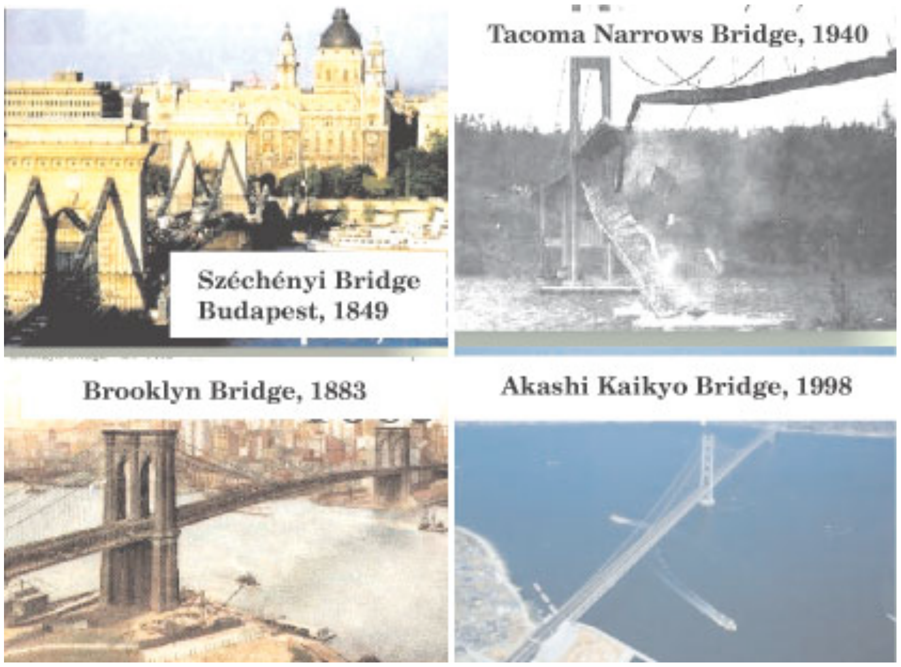
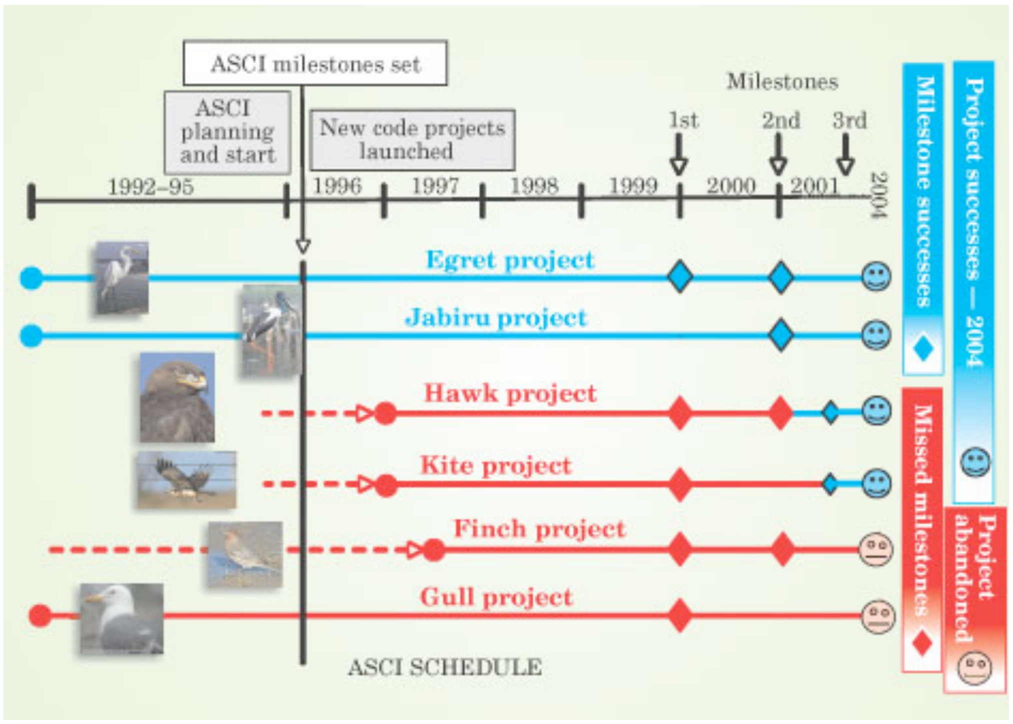


Figure 5. Four suspension bridges illustrate typical stages on the way to maturity for an engineering technology. The two 19th-century bridges, spanning the Danube at Budapest and New York's East River, are still in service. Lessons learned from the spectacular 1940 collapse of the newly built Tacoma Narrows Bridge in Washington State have contributed to the confident design of ambitious modern structures like the Akashi Kaikyo Bridge built near Kobe in a region of Japan prone to earthquakes. Its central span is 2 km long.



Lessons Learned from ASCI

Our conclusions in this article were derived from our many years of experience in the computational-physics and engineering communities. In addition, these conclusions have been validated by a detailed case study of six projects in the Department of Energy's Accelerated Strategic Computing Initiative (ASCI).[1]

DOE launched ASCI in 1996 at the Livermore, Los Alamos, and Sandia national laboratories. Its aim was to develop computer infrastructure and application codes that would serve to certify the reliability of the US stockpile of nuclear weapons in the absence of testing.

The program included the development of hardware and software for massively parallel computers as well as the creation of application codes. Livermore and Los Alamos had extensive experience simulating nuclear weapons going back to the 1950s, and these new efforts were well funded.

The six ASCI projects provided an almost unique opportunity for comparative case study. The projects had almost identical requirements and goals, but different approaches to mitigate risk. Detailed project data were available to Richard Kendall and one of us (Post) over a six-year period. ASCI management's plan was to reduce the number of projects to three or four when it became clear which were the most promising. ASCI benefited from the lessons learned through the case study and continual retrospective assessment of its experience.

The initial success rate for the ASCI codes is typical of large code projects: About a third of the ASCI projects succeeded as planned, another third succeeded later than planned, and work on the remaining third was eventually abandoned (see the figure).[13]That success rate was more than adequate for the program. For most other large computational science projects now starting, the success rate is likely to be even lower. Almost all the new simulation projects involve institutions and staff with much less experience in large-scale simulation than Livermore or Los Alamos, and they have fewer resources.

In addition to helping ASCI, the Post–Kendall case study was designed to extract lessons that could increase the success rate of computational-science projects in general. The case study results contributed to the revision of ASCI's schedule and milestones in 2002, with the result that four of the original six projects reached their goals by 2004. The study identified two key requirements for success: sound software-project management and minimization of risk. The latter means that goals and methods need to be conservative and realistic.

The history of the six ASCI projects, as charted in the figure, illustrates the importance of those principles. To preserve anonymity, the chart uses avian pseudonyms. Senior management at DOE and the laboratories developed the first set of project milestones for the year-ends of 1999, 2000, and 2001. Quantitative analysis of the history of the six projects indicated that it takes about eight years for a staff of 15–30 to develop a massively parallel three-dimensional weapons simulation. Those projects that did not have a five-year head start or did not follow sound management principles did not meet their first milestones.

The Egret and Jabiru projects, which met all their milestones, did have full head starts and well-led and strongly supported teams. The Hawk and Kite projects didn't get started until mid-1996. After they failed to meet their initial milestones, their goals were revised to be more realistic. Now, eight years after they began, they are beginning to succeed.

The Finch project attempted to integrate three existing codes. It didn't succeed, largely because the challenge of the multiscale integration problem had been underestimated. Gull had been turned into an overly ambitious computer research project. Work on both Finch and Gull has stopped and their resources were reallocated to the remaining four projects.

Failure of large-scale code projects is often erroneously blamed on poor team performance. But the ASCI case study indicated that missing milestones is often due to overly ambitious goals and schedules and to lack of support. Those are generally issues over which the code teams have no control. These findings are consistent with the experience of the broader information-technology industry.[14]

The ASCI program recognized the importance of verification and validation from the beginning. For that purpose, it established an explicit program and launched major initiatives to build experimental facilities. But even with such support, the code teams found detailed verification and validation difficult to accomplish. The conclusion was that better methods are urgently needed.