The Promise and Challenges of Large-Scale Computational Science and Engineering

Douglass Post
Chief Scientist
DoD High Performance Computing Modernization Program
(IPA from CMU Software Engineering Institute)

12th workshop on the Use of High Performance Computing in Meteorology, 30 Oct - 3 Nov 2006,
European Centre for Medium-Range Weather Forecasts (ECMWF)
Reading, UK
Exponential Growth In Supercomputer Speed And Power Is Making It A “Disruptive” Technology.

Enable paradigm shift
- Potential to change the way problems are addressed and solved
- Make reliable predictions about the future
- Superior engineering & manufacturing
- Enable research to make new discoveries
- A vastly more powerful solving methodology!

Computer power comes at the expense of complexity!
Computational Tools are becoming widely used in Science and Science

<table>
<thead>
<tr>
<th>Past</th>
<th>Present</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory</td>
<td>Pencils, paper; slide rules</td>
<td>New: symbolic math; computational solutions</td>
</tr>
<tr>
<td>Experiments</td>
<td>Physical hardware; notebooks; chart recorders; polaroid film, ...</td>
<td>New: computerized data collection &amp; analysis; little V&amp;V of computations; simple simulations &amp; experimental design</td>
</tr>
<tr>
<td>Eng. Design</td>
<td>Pencils, paper; slide rules</td>
<td>New: CAD-CAM; computational design analysis</td>
</tr>
</tbody>
</table>
Computational Science and Engineering (CSE) is a uniquely powerful tool for studying the interaction of many different natural effects.

Science-based: laws governing individual interactions are known

1. Scientific discovery
2. Experimental analysis and design
3. Prediction of operational conditions
4. Scientific design and analysis
5. Engineering design and analysis

Heuristic-based: laws governing individual interactions are heuristic and/or empirical

6. Data collection, analysis & mining
   • Social sciences, medicine, education, research
7. Heuristic simulations and decision tools (economic forecasts, war and strategy simulations,..)
Computational Science and Engineering is becoming ubiquitous in science and engineering.
Computational Science and Engineering contributes today

- Computational Science and Engineering is making major contributions today
- DoD—Stores separation, weather and ocean prediction, materials, armor penetration, RF antenna and radar signatures, aircraft and ship design and analysis, bio-warfare countermeasures, and many more
- DOE, NSF, NIST, NASA, NOAA, EPA,…—high energy and nuclear physics, nuclear weapons design, controlled fusion, materials, nuclear reactor, fuel efficiency, geophysics, astrophysics, space physics, and many more
- Industry—Crash design (GM,…), Tire design (Goodyear), chip design (Intel,…), consumer products (P&G,…), aircraft (Boeing, Airbus), structural design, drug design and data searches (Merck,…), oil exploration, and many more
**Stores Integration & Certification**

**Supercomputing Improves the Test Process**

**Old Way - Flight test it**

**Today’s Way - Computationally simulate the test and run much reduced flight test**

<table>
<thead>
<tr>
<th>Benefits:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster</td>
</tr>
<tr>
<td>Cheaper</td>
</tr>
<tr>
<td>More technical insight</td>
</tr>
<tr>
<td>Safer</td>
</tr>
</tbody>
</table>

**Quick reaction process must have validated models and tools ready when need arises**

**Future Goals – Move the test simulation into the Design Process:**

- Provide first model quicker, better, cheaper
- Continuously improve models thru collaboration
- Speed response to warfighter – less test & better design
PetaFlop computers are coming

NSF, DOE Science and Defense Systems, NASA, NOAA, DoD (DARPA) all plan petaflop computers for 2008-2012
• DARPA High Productivity Computing Systems a bright light
  • Faster computing but also
  • Higher bandwidth and lower memory latency (64k GUP/s)
  • Flatter memory hierarchy with globally addressable memory
  • + many more

But are we ready to use them?
To succeed, Computational Science and Engineering faces immense challenges*

- **Scientific and Engineering:**
  - Calculate the trade-off of many different strongly interacting effects across many orders of magnitude of multiple time and distance scales
  - Verify and validate highly complex applications
  - Develop problem generation and setup methods for larger and more complex problems
  - Analyze and visualize larger and more complex datasets
- **Project:**
  - Evolve from small code development teams to large teams
  - Successfully deploy multi-disciplinary and multi-institutional code development teams
- **Programming:**
  - Develop codes for computers that don’t yet exist.
  - Develop codes for computers that will be $10^2$ to $10^4$ faster and contain $10^2$ to $10^3$ times more processors than today
  - Develop codes with adequate performance levels
  - Cope with relatively immature tools for developing and running massively parallel applications

---

Lessons Learned are the way forward!!!

- 4 stages of design maturity for a methodology to mature—Henry Petroski—*Design Paradigms*
- Suspension bridges—case studies of failures (and successes) were essential for reaching reliability and credibility
- The Scientific Method!

**Tacoma Narrows Bridge buckled and fell 4 months after construction!**

- Case studies conducted after each crash.
- Lessons learned identified and adopted by community
- Computational Science is at stage 3
What do CSE applications look like?

Surveyed DoD and other codes to verify characterizations of CSE codes.

- Identify general characteristics
- Preamble (anonymity guaranteed)

Questionnaire asked for:

- Contact information
- Code purpose
- Team size, number of users
- Domain Science area and sponsor
- Code size (slocs)
  - Total and for each language
- Code history
  - How long did the code take to develop and how old is it now?
- Platforms
- Degree of parallelism
- Computer time usage
- Memory requirements
- Algorithms
What kind of codes are we talking about?
We surveyed our Large, Diverse DoD HPC Community to characterize our codes

- 587 projects and 2,262 users at approximately 144 sites
- Requirements categorized in 10 Computational Technology Areas (CTA)
- DoD HPCMP has about 20 computers with ~240 TFlops/s peak (circa 2006)
We sent surveys to our top 40 codes (ordered by time requested), with 15 responses so far.

<table>
<thead>
<tr>
<th>Application Code</th>
<th>Hours</th>
<th>Application Code</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTH (SNL)</td>
<td>93,435,421</td>
<td>DMOL</td>
<td>5,200,100</td>
</tr>
<tr>
<td>HYCOM (30% DoD)</td>
<td>89,005,100</td>
<td>ICEM (commercial)</td>
<td>4,950,000</td>
</tr>
<tr>
<td>GAUSSIAN (Commercial)</td>
<td>49,256,850</td>
<td>CFD++ (commercial)</td>
<td>5,719,000</td>
</tr>
<tr>
<td>ALLEGRA (SNL)</td>
<td>32,815,000</td>
<td>ADCIRC (DoD + academia)</td>
<td>4,100,750</td>
</tr>
<tr>
<td>ICEPIC (100% DoD)</td>
<td>26,500,000</td>
<td>MATLAB (commercial)</td>
<td>4,578,430</td>
</tr>
<tr>
<td>CAML (100% DoD)</td>
<td>21,000,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANSYS (Commercial)</td>
<td>17,898,520</td>
<td>Loci-Chem</td>
<td>5,500,000</td>
</tr>
<tr>
<td>VASP (U.ofVienna)</td>
<td>18,437,500</td>
<td>GAMESS (Iowa State)</td>
<td>5,142,250</td>
</tr>
<tr>
<td>Xflow (Commercial)</td>
<td>15,165,000</td>
<td>STRIPE</td>
<td>4,700,000</td>
</tr>
<tr>
<td>ZAPOTEC (SNL)</td>
<td>12,125,857</td>
<td>USM3D</td>
<td>4,210,000</td>
</tr>
<tr>
<td>XPATCH (DoD commercial)</td>
<td>23,462,500</td>
<td>FLUENT (commercial)</td>
<td>3,955,610</td>
</tr>
<tr>
<td>MUVES</td>
<td>10,974,120</td>
<td>GASP</td>
<td>4,691,000</td>
</tr>
<tr>
<td>MOM</td>
<td>18,540,000</td>
<td>Our DNS code (DNSBLB)</td>
<td>2,420,000</td>
</tr>
<tr>
<td>OVERFLOW (NASA)</td>
<td>8,835,500</td>
<td>ParaDis</td>
<td>4,000,000</td>
</tr>
<tr>
<td>COBALT (commercial)</td>
<td>14,165,750</td>
<td>FLAPW</td>
<td>4,050,000</td>
</tr>
<tr>
<td>ETA</td>
<td>11,700,000</td>
<td>AMBER</td>
<td>4,466,000</td>
</tr>
<tr>
<td>CPMD (MPI &amp; IBM)</td>
<td>5,975,000</td>
<td>POP (LANL)</td>
<td>3,800,000</td>
</tr>
<tr>
<td>ALE3D (LLNL)</td>
<td>5,864,500</td>
<td>MS-GC</td>
<td>3,500,000</td>
</tr>
<tr>
<td>PRONTO (SNL)</td>
<td>5,169,100</td>
<td>TURBO</td>
<td>3,600,600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Freericks Solver</td>
<td>2,600,000</td>
</tr>
</tbody>
</table>
Characteristics aren’t surprising.

- Even now, codes are developed by teams
- Most codes have more users than just the development team
- Codes are big
- 58% of the codes are written in Fortran.
- New languages with higher levels of abstraction are attractive, but they will have to be compatible and inter-operable with Fortran with MPI.

<table>
<thead>
<tr>
<th>Team size FTEs</th>
<th># users</th>
<th>Total sloc(k)</th>
<th>SLOC Fortran 77 (k)</th>
<th>SLOC Fortran 90, 95 (k)</th>
<th>SLOC C (k)</th>
<th>SLOC C++ (k)</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>38</td>
<td>5,038</td>
<td>820</td>
<td>24%</td>
<td>34%</td>
<td>17%</td>
<td>13%</td>
</tr>
<tr>
<td>Median</td>
<td>6</td>
<td>27</td>
<td>275</td>
<td></td>
<td></td>
<td></td>
<td>13%</td>
</tr>
</tbody>
</table>
Further data isn’t surprising either.

<table>
<thead>
<tr>
<th></th>
<th>Total project age</th>
<th>age version</th>
<th>total number of different platforms</th>
<th>Largest Degree of Parallelism</th>
<th>Typical minimum # of processors</th>
<th>Typical Maximum # of processors</th>
<th>Is memory a limitation?</th>
<th>Memory processor GBytes /proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>19.8</td>
<td>15.1</td>
<td>6.9</td>
<td>1000 to 3000</td>
<td>225</td>
<td>292</td>
<td>Sometimes</td>
<td>0.75-4</td>
</tr>
<tr>
<td>Median</td>
<td>17.5</td>
<td>15.5</td>
<td>7.0</td>
<td>1000 to 3000</td>
<td>128</td>
<td>128</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Most codes are at least 15 years old
- Most codes run on at least 7 different platforms
- Most codes can run on ~1000 processors, but don’t
- Most users want at least 1 GByte / processor of memory.
HPCMP TI-05 Application Benchmark Codes perform differently on different platforms.

- **Aero** – Aeroelasticity CFD code (Fortran, serial vector, 15,000 lines of code)
- **AVUS** (Cobalt-60) – Turbulent flow CFD code (Fortran, MPI, 19,000 lines of code)
- **GAMESS** – Quantum chemistry code (Fortran, MPI, 330,000 lines of code)
- **HYCOM** – Ocean circulation modeling code (Fortran, MPI, 31,000 lines of code)
- **OOCore** – Out-of-core solver (Fortran, MPI, 39,000 lines of code)
- **CTH** – Shock physics code (SNL) (~43% Fortran/~57% C, MPI, 436,000 lines of code)
- **WRF** – Multi-Agency mesoscale atmospheric modeling code (Fortran and C, MPI, 100,000 lines of code)
- **Overflow-2** – CFD code originally developed by NASA (Fortran 90, MPI, 83,000 lines of code)
Performance depends on the computer and on the code.

- Normalized Performance = 1 on the NAVO IBM SP3 (HABU) platform with 1024 processors (375 MHz Power3 CPUs) assuming that each system has 1024 processors.

- GAMESS had the most variation among platforms.

---

Substantial variation of codes for a single computer.

---

Relative code performance

—SC 2005 panel Tour de HPCylces
Also did detailed case studies of first 6 large US federal agency CSE codes and then another set of 5 large-scale CSE codes.

5 CSE codes (academia and lab)
## Use of Higher-Level Languages

<table>
<thead>
<tr>
<th>Application Domain</th>
<th>Falcon</th>
<th>Hawk</th>
<th>Condor</th>
<th>Eagle</th>
<th>Nene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Duration</td>
<td>~10 years (since 1995)</td>
<td>~6 years (since 1999)</td>
<td>~20 years (since 1985)</td>
<td>~3 years</td>
<td>~25 years (since 1982)</td>
</tr>
<tr>
<td>Number of Releases</td>
<td>9 Production</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>Staffing</td>
<td>15 FTEs</td>
<td>3 FTEs</td>
<td>3-5 FTEs</td>
<td>3 FTEs</td>
<td>~10 FTEs+100s of contributors</td>
</tr>
<tr>
<td>Customers</td>
<td>&lt;50</td>
<td>10s</td>
<td>100s</td>
<td>Demonstration code</td>
<td>~100,000</td>
</tr>
<tr>
<td>Nonimal Code Size</td>
<td>~405,000</td>
<td>~134,000</td>
<td>~200,000</td>
<td>&lt;100,000</td>
<td>750,000</td>
</tr>
<tr>
<td>Primary Languages</td>
<td>F77 (24%), C (12%)</td>
<td>C++ (67%), C (18%)</td>
<td>Fortran 77 (85%)</td>
<td>C++, Matlab</td>
<td>Fortran 77 (95%)</td>
</tr>
<tr>
<td>Other Languages</td>
<td>F90, Python, Perl, ksh/csh/sh</td>
<td>Python, Fortran 90</td>
<td>Fortran 90, C, Slang</td>
<td>Java Libraries (~70%)</td>
<td>C (1%)</td>
</tr>
<tr>
<td>Target Hardware</td>
<td>Parallel Supercomputers</td>
<td>Parallel Supercomputers</td>
<td>PCs to Parallel Supercomputers</td>
<td>Embedded App</td>
<td>PCs to Parallel Supercomputers</td>
</tr>
<tr>
<td>Status</td>
<td>Production</td>
<td>Production ready</td>
<td>Production</td>
<td>Demonstration code</td>
<td>Production</td>
</tr>
<tr>
<td>Sponsors</td>
<td>DOE</td>
<td>DoD</td>
<td>DoD</td>
<td>DoD</td>
<td>DoD, DOE, NSF</td>
</tr>
</tbody>
</table>
Nine Cross-Study Observations

1. Once selected, the primary languages (typically Fortran) adopted by existing code teams do not change.

2. The use of higher level languages (e.g. Matlab) has not been widely adopted by existing code teams except for "bread-boarding" or algorithm development.

3. Code developers in existing code teams like the flexibility of UNIX command line environments.

4. Third party (externally developed) software and software development tools are viewed as a major risk factor by existing code teams.

5. The project goal is scientific discovery or engineering design. "Speed to solution" and "execution time" are not highly ranked goals for our existing code teams unless they directly impact the science.

6. All but one of the existing code teams we have studied have adopted an "agile" development approach.

7. For the most part, the developers of existing codes are scientists and engineers, not computer scientists or professional programmers.

8. Most of the effort has been expended in the "implementation" workflow step.

9. The success of all of the existing codes we have studied has depended most on keeping their customers (not always their sponsors) happy.
Developing a large, multi-scale, multi-effect code takes a lot of people a long time, and development continues through the entire life cycle of the code.
The process is complex!
Computational Science Workflow

1. Requirements
2. Design
3. Code
4. Test
5. Run

Not the WaterFall Model!

Upgrade existing code or develop new code
Computational Science and Engineering has at Least Four Major Elements.

<table>
<thead>
<tr>
<th>Computers</th>
<th>Codes</th>
<th>V&amp;V</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Making enormous progress but at cost of complexity, particularly memory hierarchy</td>
<td>More complicated models + larger programming challenges</td>
<td>Harder due to inclusion of more effects and more complicated models</td>
<td>Use tools to solve problems, do designs, make discoveries</td>
</tr>
<tr>
<td>Need to reduce programming challenge</td>
<td><strong>Greatest bottleneck</strong></td>
<td>Inadequate methods, need paradigm shift</td>
<td>Users make connections to customers</td>
</tr>
</tbody>
</table>

- We need to develop a total capability to solve problems, not just build codes or computers.
What are the needs of CSE Application Codes?

- Developers and production users want and need:
  - Fast integer and floating point arithmetic (with fast divides)
  - Fast, global addressable, reliable memory and data storage with low latency
  - Stable, long-lived and reliable platforms and architectures
  - Stable, long-lived and reliable software development and production tools that provide the needed capability and are simple and easy to use
  - Developers want something like a Unix/LINUX or Mac workstation development environment or better

**Summary:** Users and developers want to solve their scientific or engineering problem and not worry about the details of computers
What are they getting?

- Distributed memory with only very slowly improving memory bandwidth
- Slowing rate of processor speed growth

- Distributed processor and memory systems linked together in ever more complex networks
- Rapid turnover in machines and machine architectures (2-4 years)
- Unreliable parallel file systems
- Unstable development and production environment
- Highly complex programming environment and challenges
  - Complex architectures—>Complex programming
  - Performance that is poor (a few % of peak) and hard to optimize
  - Frequent and challenging ports to new platforms
Three Challenges
- Performance Challenge
- Programming Challenge
- Prediction Challenge

Where case studies are important

Case Studies are needed for success
- The Scientific Method

Paradigm shift needed
- Computational Science moving from few effect codes developed by small teams to many effect codes developed by large teams

Similar to transition made by experimental science in 1930—1960
- Software Project Management and V&V need more emphasis

Issues summarized in January 2005 Physics Today Article.*

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.

Douglas E. Post and Lawrence G. Votta

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 sorting difficult nonlinear problems in magneto-hydrodynamics; atomic, molecular, and nuclear structure; fluid turbulence; shock hydrodynamics; and cosmological structure formation.

Beyond such well-established uses to theorists and experimenters, the exponential growth of computer power is now launching the new field of computational science. Meanwhile, computational physicists 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 supernovae explosions (see figure 1 and 2), interstellar dust particles, nuclear explosions (see the box on page 28), assisted impacts (figure 3), and the effect of space weapons 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 struggling immensely. We point out three distinct challenges that computational scientists 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. This 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.

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. Much remains to be done to meet the programming challenge. Not only is it important 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 nuclear codes developed by small teams to many 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 many codes, and the problem of linking all the effects within the codes. This often results in codes that are not sufficiently robust and reliable to be the basis of important decisions facing society. The growth of code size and complexity, and its attendant problems, force some resemblance to the transition from small to large scale by experimental physicists in the decades after World War II.

A comparative case study of six large-scale scientific code projects, by Richard Korenblit and one of us (Post), has yielded three important lessons. Verification, validation, and quality management, we found, are essential to the success of a large-scale code-writing project. Although some computational science groups—those illustrated by figures 1-4, for example—satisfy all three requirements, many others current and planned projects give them insufficient attention. In the absence of any one of these requirements, one does not have the assurance of independent assessment, confirmation, and repeatability of results.


Email post@ieee.org to get a copy.
Code Development will be (is) the major bottleneck in the future (now).

• Codes need to scale to many, many thousands of processors.
• Low-hanging fruit has been gathered (porting of serial codes to parallel computers).
• Exciting opportunities to remedy present deficiencies:
  – Better spatial and temporal resolution
  – More accurate models
  – Inclusion of a more complete set of effects
    • Strongly-coupled, multi-scale effects
  – Codes that can model a whole system
  – Codes that can get answers in minutes to hours rather than days to weeks to months
• The greatest opportunities include integrated codes that couple many multi-scale effects to model a complete system.
• Success often requires large (10 to 30 professionals), multi-disciplinary, multi-institutional teams and 5 to 10 years of development time.
• It’s exciting, it’s challenging and it’s risky.
Predictive Risk is even more serious than Programming Risk.

- Programming Risk is a matter of efficiency
  - Programming for more complex computers takes longer and is more difficult and takes more people, but with enough resources and time, codes can be written to run on these computers.
- But the Predictive Risk is a matter of survival:
  - If the results of the complicated application codes cannot be believed or if the right codes are not developed and used effectively, then there is no reason to support the development and deployment of platforms or supporting software.
  - Pretty pictures are not necessarily consistent with the laws of nature!
  - Computational scientists and engineers have to be aware of all the issues:
    - Development of the application codes takes time and resources, often tens of people for tens of years plus resources for validation and testing and productions runs.
    - If the right codes are never developed, they cannot be used to solve problems.
    - If they are developed and give wrong answers, they cannot be used to solve problems.
    - If they are developed and not utilized effectively to solve problems, then the problems won’t be solved.
Proto-FALCON Workflows were initially serial

Historic Contractor A Code Development Workflow (Serial Development)

calendar time (years)

5  10  15  20  25  30  35

Conservation Equations Effect A

V&V, Production Use

Effect B

Effect C

Effect D

User Requirements (modified as program needs dictate)

Prior generation of Contractor A simulation codes (prototypes)
Ambitious schedule required parallel development with no contingency.

Delayed delivery of Package with Effect C led to missed milestones.

Package C failed to be delivered in working form
Computational Science and Engineering is making the same transition that experimental science made in 1930 through 1960.

- Computational Science and Engineering moving from “few-effect” codes developed by small teams (1 to 3 scientists) to “many-effect” codes developed by larger teams (10, 20 or more).
- Analogous experimental science transition made in 1930-1960 time frame
- Small-scale science experiments involving a few scientists in small laboratories —> “big science” experiments with large teams working on very large facilities.
- “Big Science” experiments require greater attention to formality of processes, project management issues, and coordination of team activities than small-scale science.
- Experimentalists were better equipped than most computational scientists to make the transition and they had more time to make the transition.
  - Small scale experiments require much more interaction with the outside world than small-scale code development.
  - Experimentalists had ~20 years, while computational scientists are doing the transition much more quickly.

![Early 1930’s](image1) ![Late 1930’s](image2) ![CERN 2000](image3)
We studied 6 federal agency projects to identify the “Lessons Learned*”

The Successful projects emphasized:
- Conservative approach - Minimize Risks!
  - Building on successful code development history and prototypes
  - Better physics and computational mathematics over better “computer science”
  - The use of proven Software Engineering rather than new Computer Science
    - Don’t let the code project become a Computer Science research project!
- Sound Software Project Management - Plan and Organize the Work!
  - Highly competent and motivated people in a good team
  - Development of the team
  - Software Project Management: Run the code project like a project
  - Determining the Schedule and resources from the requirements
  - Identifying, managing and mitigating risks
  - Focusing on the customer
    - For code teams and for stakeholder support
  - Software Quality Engineering: Best Practices rather than Processes
- Verification and Validation – Correct Results are Essential!
  - Need for improved V&V methods became very apparent

The unsuccessful projects didn’t emphasize these!

Verification and Validation

- Customers want to know why they should believe code results
- Codes are only a model of reality
- Verification and Validation are essential
- Verification
  - Verify equations are solved correctly
  - Regression suites of test problems, convergence tests, manufactured solutions, analytic test problems, code comparisons and benchmarks
- Validation
  - Ensure models reflect nature, check code results with experimental data
  - Specific validation experiments are required
    - Federal sponsor is funding multi-billion dollar validation experiments for V&V,…
- V&V experience with these and other codes indicates that a stronger intellectual basis is needed for V&V
- More intense efforts are needed in both types of V&V if computational science is to be credible

Many things can be wrong with a computer generated prediction.

- Experimental and theoretical science are mature methodologies but computational science is not.
- Hatton study* indicates that Scientific codes have ~ 6 defects per 1000 lines of code.
- Code could have bugs in either the models or the solution methods that result in answers that are incorrect.
  - e.g. 2+2=54.22, \( \sin(90^\circ) = 673.44 \), etc.
- Models in the code could be incomplete or not applicable to problem or have wrong data.
  - E.g. climate models without an ocean current model.
- User could be inexperienced, not know how to use the code correctly.
  - CRATER analysis of Columbia Shuttle damage.
- Many examples: Columbia Space Shuttle, Sonoluminescence, Fusion

It’s risky. Software failures are not just in the IT industry.

- While software failures are commonly acknowledged in the IT industry*, not much is heard about them in the technical HPC community.
- But they exist.

Nov.25, 2004 Economist
Computer codes not delivering!

Jan., 1997 IEEE Computer
Software errors crash Ariane launch.

Nov., 2004 IEEE Spectrum
Software failure takes LA FAA controllers off the air.

Jan., 2005 ComputerWorld
“FBI trying to salvage $170M software package”
Perspective:
Requirements are important after all

- Often said that computational science and engineering software doesn’t have requirements in same sense as the IT industry
- Computational science and engineering does have highly rigid requirements
  - The laws of nature
- Computational science and engineering code development can’t be planned in detail because it involves discovery of how to accurately simulate those laws
Perspective: Software Engineering and Computer Science are different and each is important

- Every successful code project needs software engineering, not computer science
- Software engineering involves the implementation of proven methodologies for code development
- Computer science involves exploration, research and development of new methodologies and concepts
- Computer science is an essential activity, but it should be an independent activity
CREATE Focuses on Design and Engineering for Acquisition

Goal—Make design and engineering a more effective contributor to acquisition

- CREATE will develop advanced computational engineering tools to optimize the design and testing of:
  1. Military aircraft (i.e., structures & aerodynamics)
  2. Naval vessels (i.e., structures & hydrodynamics)
  3. Integration of RF sensors and C4ISR antennas with platforms (i.e., electromagnetics & signatures)

- Each project: $10M/year for 10 years; total $300M

- Result:
  - Faster and more effective acquisition process
  - Better, faster and more effective design and validation
  - Fewer problems discovered in testing
  - Fewer costly delays and rework to fix flaws
Recap: What do you do you need to succeed?

Case studies* of existing computational science and engineering project indicate that increased emphasis is crucial for:

• Verification and Validation
  – Accurate, reliable results, are needed and not just pretty pictures!

• Software Project Management
  – Single investigator paradigm doesn’t work for large teams
  – Large teams need a project orientation to organize and coordinate the work

• Software Engineering
  – Software development is a highly technical process for producing a complex system
  – Success requires effective methods and tools that balance the need for structured development with the required degree of flexibility and agility.

Observations on Weather prediction

• Validation is a challenge
  – Few controlled experiments
• Who is the code architect? Where is the conceptual integrity? And who enforces it?
• All codes involve trade-offs between accuracy and time to problem completion.
  – I’m not sure that many weather/climate codes enforce the trade-off to ensure practical run times.
• Example: ASCI academic alliances:
  – Multi-physics codes, each module with the “best physics”
  – Result: Initially could only simulate 6 s of a 20 minute fire, 2 s of a 120 s rocket burn,…
Reductionism and Emergence

• Weather and climate codes include 100s of effects
  – Problem is reduced to its constituents
  – Answer depends on trade-off of many competing effects
• Robert Laughlin (Nobel Prize, 1999) and others have been pointing out that solving complex problems by calculating the trade-off of all of the detailed effects (reductionist) is an NP incomplete problem
• They claim that we only solve problems where there are a set of overarching or “emergent” principles (e.g. conservation laws, symmetry, thermodynamic principles,…)
  – We use hydrodynamics to calculate ocean flow, not molecular dynamics
• How can we sure that weather models correctly capture the relevant emergent principles?
  – Validation is the best way to ensure that the emergent principles are captured
The Future

• We live in “exciting times”
• CSE offers tremendous promise to address and solve important problems
  – The potential to tackle and solve problems that we couldn’t before now
• CSE faces many challenges just like every other new problem solving methodology has faced
• It will take time and a lot of hard work
• But if we face and overcome the challenges we can do great and important things