INTERNATIONAL WORKSHOP ON SCALABLE ENGINEERING SOFTWARE

JUNE 2-3, 2010

FINAL REPORT

November, 2010

David B. Nelson
# TABLE OF CONTENTS

Executive summary .................................................................................................................................................... 1
Introduction................................................................................................................................................................... 2
Purpose of the workshop ......................................................................................................................................... 3
Structure of the workshop ....................................................................................................................................... 3
Areas addressed........................................................................................................................................................... 3
Session I: Descriptions of european scalable engineering codes ............................................................ 5
Panels I - IV .................................................................................................................................................................... 7
Scalable Algorithm advance and barriers ......................................................................................................... 7
Development approaches, including verification and validation, ease of use, and workflow factors........... 8
Funding models............................................................................................................................................................ 9
Intellectual property models ................................................................................................................................10
Code maintenance and upgrade models ..........................................................................................................11
User community models.........................................................................................................................................12
Conclusions..................................................................................................................................................................13
Appendix A. Agenda .................................................................................................................................................14
Appendix B. Participants........................................................................................................................................16
Appendix C. Presentations.........................................................................................................................................18
EXECUTIVE SUMMARY

With support from the DoD High Performance Computing Modernization Program (HPCMP) and the Networking and Information Technology Research and Development Program (NITRD), WTEC2, a sister company to the World Technology Evaluation Center (WTEC), convened a workshop on June 2-3, 2010 to examine factors that encourage the development of scalable engineering software. The workshop featured six presentations from European organizations that have developed widely used scalable engineering codes in the disciplines of biology/chemistry/materials, finite element analysis, weather/climate/ocean modeling, fluid dynamics, and integrated code suites. These presentations were complemented by four panel sessions that discussed factors affecting the development of scalable software and the current situation in the U.S. The workshop concluded with two summarizing presentations.

The workshop was motivated by previous studies that documented the critical role of simulation in engineering and the inability of most engineering software to utilize the power of large computers because of the scaling barrier in engineering software: most engineering codes in current use do not scale well beyond at most 100 processors, and some do not scale at all. These prior studies also suggested that some European codes in general use have achieved good scaling on realistic problems, in some cases up to several thousand processors.

Discussions at the workshop provided strong evidence that scalable engineering codes for realistic problems can be developed that scale to hundreds, or in some cases up to thousands of processors. However, for success it appears that several important factors must be favorably aligned. These include an expert and committed development team that responds to user needs, stable funding that is provided over more than a decade for development, support and upgrades, use of scalable physics models and algorithms, availability of suitable scalable software libraries and middleware, code licensing terms that make it cost-effective for users to scale up their problems, regular releases that include bug-fixes, improvements, and upgrades to the code, and a large and vibrant developer-maintained user community that provides endorsement, support and feedback.

The European presentations provided a striking insight regarding funding: European government agencies commonly provide long-term stable funding to companies to develop and maintain scalable engineering codes deemed important to the country. The European codes presented at the workshop were originally developed with government funding at universities or government laboratories. Several of the development teams have since moved to companies, and they have continued to receive government funding combined with private funding. All of these codes are freely available to users who meet certain criteria (such as nationals of the country that developed them or their collaborators.) Several are freely available to anyone as downloadable open source. The U.S. government does not typically fund code development in companies other than for specific government purposes, and these codes are not usually available to general users.
In general the licensing terms for European codes developed under government funding seem more liberal than in the U.S., commonly using the open-source Gnu Public License. The free availability of these codes may enhance their widespread acceptance and use.

**INTRODUCTION**

Recent studies have documented the critical importance of physics-based modeling and simulation to the design, development, and use of engineered systems in industry and government. However, these studies have also shown that most engineering software suffers from a scaling barrier that seriously limits our ability to use the full power of modern highly parallel computers to model complex engineered systems. For example, a recent study conducted by the Council on Competitiveness under government sponsorship documented the scaling barrier and showed that most U.S. commercial engineering codes do not scale to use more than about one hundred processors, and in some important cases do not use parallel processing at all.\(^1\)\(^2\)\(^3\) Findings by that study lead to several plausible reasons for this scaling barrier, including lack of R&D funds for Independent Software Vendors (ISVs) to improve their codes, lack of trained personnel, licensing and business models that discourage use of highly scaled codes, and lack of access to highly parallel systems.

The study pointed out that research codes developed at universities and national laboratories in several disciplines of science and engineering have achieved much greater scale-up by employing novel scientific algorithms and improved computer science. However, for a variety of economic, technical and organizational reasons very few of these codes have become commercialized or seen wide use. Their existence shows that it is possible to break the scaling barrier, but their lack of widespread use shows that, practically speaking, the scaling barrier continues. The study concluded that federal funding patterns have changed in recent years to discourage the types of long-term effort required to commercialize research codes. This conclusion is supported by a National Academies study of High Performance Computing that concluded “...from the committee’s visits to DOE sites, members got the clear impression that there are no incentives for the transfer of codes developed at those sites to industrial use and no significant funding to facilitate the transfer.”\(^4\)

A more recent WTEC study, *International Assessment of Research and Development in Simulation-Based Engineering and Science*, found that in some areas of science and engineering the U.S. may be lagging in developing scalable codes, especially by comparison with Europe and Japan.\(^5\)

---


A WTEC2 study conducted in preparation for this workshop, *Identification of Successful International Models for Scalable Engineering Software*, confirmed this finding and provided additional information about scalable engineering codes.6

**PURPOSE OF THE WORKSHOP**

The purpose of the International Workshop on Scalable Engineering Software was to better understand how to produce scalable engineering codes by identifying factors that are important for their development and maintenance. To do this the workshop examined the factors that have led to scalable European codes and compared these with current practice in the United States. Among the areas considered were: development models including ease-of-use and workflow factors, funding approaches, intellectual property issues, code maintenance and upgrade approaches, and user community involvement.

**STRUCTURE OF THE WORKSHOP**

The agenda for the workshop is presented in Appendix I. Workshop participants are listed in Appendix II. The agenda of the workshop included a keynote talk, five invited talks describing European codes with good scaling properties, four panels that discussed issues in developing scalable codes, and two summary talks. The workshop included substantial open discussion among participants; where possible this is captured in the report.

**AREAS ADDRESSED**

The workshop addressed the following technical areas:

1. Characterization of code scalability: measures include benchmark runs on specific systems, graphs showing scalability as number of processors increases, discussion of whether the code displays weak or strong scalability up to a certain number of processors, sustained rate in TFLOPs, or other.

2. What technical advances were required in order to achieve good scalability? These could include better physical models, better algorithms, better computer science, etc.

3. What was the business model used in developing the code? Business models include factors such as funding model for code development (private funding, government funding, volunteer effort), organizational model (private company, university, government center, volunteer community, individual), leadership model (charismatic leader, appointed leader, steering committee, etc.), intellectual property model (proprietary, open source, public domain, informal), sustainment model (license fees, contributions, sustained government funding, sustained private funding, and embedding in an ongoing research group).

---

4. Measures regarding the acceptance and adoption of codes within the engineering community: these could include discussion of reference accounts, estimates of number of users, engineering areas where the code is most used, or other.
SESSION I: DESCRIPTIONS OF EUROPEAN SCALABLE ENGINEERING CODES

This session included six presentations of European codes chosen from the disciplines of computational biology, chemistry, and materials science, computational finite element analysis including structural mechanics, climate/weather/ocean modeling, computational fluid dynamics, and integrated code suites. Each of the chosen codes is widely used and has relatively good scaling properties; their developers release frequent upgrades and maintain good user relations:

- Elmer – Multiphysics Open Source FEM Package, Thomas Zwinger, IT Center for Science Ltd., Espoo, Finland
- The Met Office Unified Weather/Climate Model, Paul Selwood, UK Met Office, United Kingdom
- HPC for Industrial Use: Code_Aster and Salome_Meca, Christophe Durand, EDF R&D, France
- Parallelization and Scalability in OpenFOAM, Hrvoje Jasek, Wikki Ltd, United Kingdom
- DL_POLY: Software Solutions in Molecular Dynamics, I.T. Todorov & W. Smith, Daresbury Laboratory, United Kingdom
- The Quantum ESPRESSO Distribution, Paolo Giannozzi, Universita di Udine, Italy

The speakers were asked to address six questions in their talks. The summaries of the talks are organized in terms of the questions.

1. Please characterize the scalability of the code.

In each case scalability depends in part on the details of the simulation, such as complexity of the included models and features such as free surfaces. However, for most of the presented codes good scalability has been achieved on realistic problems for up to a few thousand processors. Code_Aster scales to 100 processors, consistent with other structural mechanics codes.

2. What technical advances were required in order to achieve good scalability?

Each of the codes relies on commonly available, external, highly optimized routines and libraries such as MPI and OpenMP, FFT, MUMPS, BLAS, and LAPACK. Beyond that, each code has relied on mathematical advances to increase parallelization, carefully tuned coding, and user feedback regarding scaling success. The codes also rely on externally developed analytical tools to identify bottlenecks and guide tuning for scalability. Some codes have used the multiple threads capabilities of the hardware to improve scalability, and a few of them are exploring the use of graphical processing units (GPUs) to increase parallelism. FORCHECK is used by some for checking validity of FORTRAN code.
3. What was your business model for developing this code? (Business model includes why you developed the original code and why you decided to improve the scalability, targeted customer base, management, schedule, team organization, funding, intellectual property, user input, etc.)

In each case the respective government provided the initial funding for code development, either through grants to universities or funding for R&D centers. In most cases the initial impetus for the code was either to conduct academic research or to develop improved codes for specific computational research projects. Continuing funding for maintenance and improvements is mostly governmental, although some of the codes benefit from commercial co-development activities on a project basis. This funding has been long term and usually stable, a factor that seems important for the success of these codes. Some of the codes have been spun out from their academic origins into companies, with continued government funding.

Commercial co-development projects provide interesting examples of a “mixed development strategy.” In these projects the developers work on a proprietary basis with the funding company to apply and/or enhance the code specifically for simulation that improves the competitive position of the funder. The developers also benefit from the project through the experience of extending their code into new regimes. In many cases the project terms call for release of newly developed code into the standard distribution after a negotiated period of time.

All of the codes are free to external users, in some cases with restrictions. The most common license is the GPL (Gnu Public License), an open source license, and the code is freely available for download. The user communities for these codes are typically quite large, numbering in the thousands. Aside from the UK Met Office and EDF, the team that maintains the code is small and often loosely organized. Yet each of these codes (and their predecessors) has endured for over a decade, and shows no sign of atrophying or disappearing. The synergy between freely downloadable open source and large user communities seems to be positive. Users assist with quality control and bug identification; they also provide suggestions for improvements and through their numbers and visibility endorse the value of the respective codes to funding organizations. The codes are given regular updates, usually at least annually, and sometimes much more frequently.

4. Please discuss any metrics you may have regarding the acceptance and adoption of this code within the engineering community.

The metric most commonly cited was the number of downloads and users. In several cases the user communities are organized via user forums and other Internet facilities, training sessions, annual conferences, etc. Users also contribute results from the code as testimonial to its value. In some cases the user community develops and contributes back advances or extensions to the codes. The code maintainers seem to take seriously the job of helping and cultivating the user communities.

5. Please describe the approach you have taken with regard to verification and validation of your code.

For several of the GPL codes verification and validation appears to come mainly from user runs compared with other codes or experimental data. For these codes there does not seem
to be a rigorous methodology such as test suites or regression analysis. However, the existence of large user communities helps to integrate verification/validation with bug reports and other benefits of large scale use. EDF does internal V&V, consistent with the stringent requirements of nuclear design. Elmer has a built-in suite of about 135 tests to be run after each update. These can be triggered automatically along with an analysis of the results.

PANELS I - IV

During the four panels the panelists were requested to address fifteen questions relating to scalable code development, usage, and maintenance organized into four groups (panels). The summaries of the panels are presented in terms of these questions.

SCALABLE ALGORITHM ADVANCE AND BARRIERS

1. What recent algorithmic advances have been important for improving scalability of codes in your area of application? What future broadly-applicable algorithm advances would improve scalability in your area? What are the major algorithmic barriers to scalability in your area?

For most codes advances in parallel algorithms that reduce communications and operations count for the specific problem have been very important. The panelists agreed that optimized standard parallel libraries and profiling/debugging tools were also important for rapid progress in scalability. For many of the codes, scaling above a few thousand processors on realistic problems is proving very hard. DL_POLY has been scaled to 16K processors on an IBM Blue Gene and 64K on a Cray XT6 before hitting I/O bottlenecks. Parallel mesh generation is difficult, and some of the linear algebra packages have scaling limitations. Generally the communications latency in parallel computers makes it hard to scale unstructured problems or implicit techniques. Advances in communications speed or the discovery of new algorithms with less need for communication could help scaling. Some codes are exploring the use of GPUs for further speedup. However this makes the programming model even more complicated and tends to tie the code to one architecture.

Future needs include better ways to handle I/O that is becoming a major bottleneck for some codes, easier or more automatic unstructured grid generation and data layout techniques, and hybrid or multi-level programming models to better handle massive parallelism. Some developers are exploring the possibility of parallelization in the time domain to increase scalability. For example, hyperbolic problems that are local in space and time might be amenable to computing "patches" of space-time in parallel.

2. What tools and middleware (e.g. compilers, memory management tools such as MPI, data layout, code optimization, performance-monitors, memory and cache use analyzers, profilers, debuggers, etc.) have been most helpful in improving scalability of codes in your area? What future improvements in this class of tools would be most helpful?

Developers and users rely heavily on optimizing compilers, profilers and performance analysis tools, debuggers and other middleware to optimize their code and extend its scalability. For most codes the main parallelizing standard is MPI. For some codes OpenMP is
also used for parallelization on multicore shared-memory nodes that offer multiple execution
threads. The most common compiled languages used for scalable engineering codes include
F77, F90, C and C++, with some use of interpreted languages such as Python at the executive
level. Charm++, a parallel processing extension of C++, is used to aid load balancing and
communication optimizing. Compilers in common use for these languages include PGI, GNU,
Intel, Cray, IBM, and Lahey. Examples of middleware in common use include valgrind
(debugging/run-time check), as well as profilers including gprof (GNU), HPM Xprofiler (IBM),
craypat (Cray) and vtune (Intel.) Scalasca was mentioned as a profiler for MPI. One speaker
noted the absence of an OpenMP profiler that would show detailed statistics on performance
and overhead of OpenMP threads.

DEVELOPMENT APPROACHES, INCLUDING VERIFICATION AND VALIDATION, EASE OF
USE, AND WORKFLOW FACTORS

3. What models for code development are most suitable for your area of application?
Factors could include size and skill mix of team, management structure, and time
required for code development and maturation.

Several different models were discussed by participants, but the differences appeared to
depend more on type of institution, importance of the application to the institution, and on
available funding than on area of application. In academic settings, teams of 2-5 developers
are common, whereas in companies and national laboratories several dozen developers may
be working on large codes or suites of codes. A few of the successful codes were initially
produced by one developer, often a graduate student, who worked for several years before
releasing the first version of the code. For some academic codes researchers from different
universities may join forces to produce a common code that embodies their individual
research. In every case for large successful codes the development time is at least several
years, and often decades. Some of the most successful codes have gone through several
formal upgrades, with total project lifetime of twenty years or more.

The skill mix and management structure appear to vary substantially. Some of the academic
codes were produced by discipline scientists who picked up the requisite computer science
along the way. In many cases these codes were the brain-children of one or more committed
developers, who pushed the project along for many years through research grants or
institutional funding. The corporate and laboratory codes were usually developed with
structured development teams, more formal management, and funding arrangements to meet
internal commitments. Many of the commercial, proprietary codes were originally produced
in academic settings and spun off into various forms of profit-making company.

Several of the developers acknowledged that they had given insufficient attention to
computer science issues in designing their codes, largely because the teams tended to be led
by discipline scientists and engineers who were not computer science or software
engineering experts.

4. How do you determine the needs of users when developing codes in your area,
including insuring that the codes are easy to use by engineers?

The successful open-source codes typically have user communities of several hundred to
thousand users, not necessarily all active. The developers typically maintain online user
forums, manage email help services, hold user meetings, and conduct training programs to
assist these users. These user communities provide frequent and pointed feedback on bugs or deficiencies. They also provide ideas for upgrades, and less frequently submit code for inclusion in the package. User comparisons with other codes or with experimental data provide ongoing informal verification and validation. The developers justify the effort of supporting their user communities, often at no cost, as proof to the code funders that their support is worthwhile. Developers of proprietary codes typically (but not always) pay more attention to ease-of-use issues than do the open source developers.

5. Please describe the approach you have taken with regard to verification and validation of your code.

A wide range of models seem to be in use, partly depending on the institutional and funding arrangements. Proprietary code developers maintain suites of test cases and work with users to verify their codes. Many customers of commercial codes feel that they are paying (in part) for verification and validation and therefore expect such codes to “just work” when given a valid model. Expectations are often lower for non-commercial codes.

Where the new releases are aimed at higher efficiency or greater parallelization, the verification may measure the extent to which the code gives the same answers as the previous version. Validation is typically based on comparison of simulation data against published experimental data. Industrial and national laboratory developers may compare different codes running the same data for verification. They may also rely on in-house experimental data for validation. Small developer groups, especially for open source codes, rely significantly on their user bases for verification and validation. The larger the user base, the greater the diversity of scientific problems that the code will be used for, leading to a greater likelihood of users uncovering problems. Many of their academic users publish code results as part of their research papers, including cross-code and experimental comparisons, that the developers quote to support their claims for accuracy and correctness.

FUNDING MODELS

6. How are development and maintenance of codes in your area funded? Discuss factors such as funding source, duration and stability, funder milestones, and funding to sustain and maintain the codes after initial release.

All of the scalable codes showed evidence of patient, long term funding support. For the smallest development teams the patience may come from the tenacity of one or two developers who cobble together funding from different sources over many years to improve and maintain their code. In several cases the funding was for research projects rather than code development, but the developers were able to continue code development as part of their research support. For the larger teams the patient funding came either from in-house sources to develop corporate resources or from government sources that funded the code as an important community resource. All of the successful codes had been funded for at least a decade or more.

Code maintenance also required patient funding. Several code developers stated that the existence of large, satisfied user communities using the code for important projects helped to justify funding for code maintenance and improvement, even if no users actually helped fund the code.
For proprietary codes the market place determined funding rate and stability. Again, successful codes required long-term funding based on acceptance of the code in the market place. One difference that was observed is that the boundaries between government and corporate entities and funding sources are much less rigid in Europe than in America. It is easier for for-profit companies to obtain direct government funding to support software research and development, and easier for both commercial and non-profit institutions to leverage both government and commercial funding to contribute to a software project. For example, companies used government funding to maintain the core team and augmented it with commercial funding for specific projects, some of which led to additional code being open-sourced. In many cases the codes were open source, and the government funded them as national resources.

7. How should development and maintenance of codes in your area be funded?

All of the code developers emphasized the importance of patient, long-term funding. For codes developed by national laboratories and companies to meet internal needs long-term internal funding is vital. Rapidly changing priorities and funding demoralize the group and delay or kill the code. For codes developed to meet community needs the most common model seemed to be sustained core funding from a long-term committed sponsor, augmented by occasional short-term funding for specific projects, sometimes from private sources. For proprietary codes the funding stream is from ongoing licensing fees, and the challenge is to create sufficient value early in the development process to build a sufficient revenue stream. (Many of the proprietary codes were originally developed in academic settings, and code development was justified for particular research.)

Maintenance funding appears to be as important as developmental funding for successful codes, and maintenance is as important as initial development for long-term code success. Maintenance includes coding and algorithm improvements to enhance performance and accuracy and extend the range of validity, providing new capabilities for users, porting to new platforms, finding and fixing bugs, and testing before releasing new versions. Several developers of community codes pointed to the value of good relationships with their user communities in convincing their sponsors to continue funding. For proprietary codes maintenance funding is essential to maintain license revenue.

Several participants pointed out a difference between open source and proprietary codes that is partly based on user needs and resultant funding. Open source codes tend to put more attention on performance and scalability, with less attention to ease of use issues and error checking. Proprietary codes are often designed to be easy to use by bench engineers doing projects under tight deadlines. As a result user interfaces are designed to be intuitive, data setup is automated where possible, and input data is checked for consistency.

INTELLECTUAL PROPERTY MODELS

8. What type of license is appropriate for codes in your application area?

Workshop participants had very different views in this area. Those whose codes are offered under an open source license, usually GPL, maintained that this was best because it encourages broad use, with attendant discovery and elimination of bugs, demonstration to funders that the code was in demand, and in some cases improvements in the code that are placed in the code base. Those who offered commercial proprietary licenses maintained that
this is the only way to maintain control of the code and to ensure a continuing revenue stream. Concerns were also expressed about the export control issues attendant in freely-downloadable source code. One industrial participant pointed out that in most cases it is the data that should be closely controlled, not the code. The data embodies trade secrets, national security concerns, and other intellectual property more than does the code. Some industrial participants pointed out that they were not very good at selling code, so getting out of that business helps to focus on their real work. For some small companies the revenue from supporting an open-sourced code exceeds that which they would otherwise receive from licensing a proprietary code. By open-sourcing the code they build larger user communities, create more name-recognition, and develop more funding opportunities.

9. Is binary-only availability suitable or should the source be available to users?

Some participants preferred binary-only availability because it lets the developer/maintainer preserve control over the code. For proprietary codes this helps to preserve the economic value of the code, and even for non-commercial (freely distributed) codes it avoids the possibility that user modifications to the code might introduce bugs that are attributed to the developer. Especially for proprietary codes the black-box nature of binary-only distribution helps to preserve trade secrets that may be present in the algorithms or coding details.

10. What is or should be the role of open-source intellectual property and licensing (e.g. gpl, bsd) for codes in your application area?

Participants who develop or use open-source codes agreed that this licensing model can be very appropriate in some settings. Open source licensing encourages widespread usage and can quickly build the reputation of codes in cases where licensing revenue is not a prime consideration. Developers who offer their codes under open source licenses derive revenue from government or corporate sponsors, consulting or maintenance contracts and paid collaborative agreements. Open source removes the need for manpower to determine and enforce licensing terms, especially for developers who do not expect to derive much revenue from their codes. Several participants pointed out that the legal enforcement of terms in open source licenses is as yet little tested in courts. Some developers use a hybrid approach, offering the alternative of a free open source license or a paid proprietary license. The advantage of the latter is that the licensee can extend the code to produce a proprietary version without having to open-source the code it develops.

CODE MAINTENANCE AND UPGRADE MODELS

11. How should codes be maintained and/or upgraded in your application area?

Participants generally agreed that it is important for developers to maintain close contact with their users, whether the code is open source or proprietary. Especially for open source codes Internet-based tools such as forums and Wikis are widely used for this purpose. User meetings and training sessions are also valuable sources for user feedback. Participants also agreed that code maintenance, including interactions with users, should receive strong attention. Many developers offered regular releases or upgrades, often several times annually. User feedback on problem areas and user requests for enhancements can help to guide the maintenance and upgrade priorities. Especially for codes intended to be scalable, it
is important to incorporate as quickly as possible discipline discoveries of more easily parallelized physical models or computer science improvements in parallelization techniques. Code developers must be prepared to decide that an existing code has become so unwieldy that it should be rewritten from scratch, even if this entails a major development cycle.

12. What is the role of user input in code maintenance and upgrades, including ease-of-use factors?

This question has been covered in several previous questions. In summary, developers should offer several easy means for users to provide input into code usability and enhancement, using both face-to-face and Internet-mediated techniques. Users will point out shortcomings of the code as used on their problems, will occasionally report bugs, but in general will not contribute code.

13. How do you decide whether to continue incremental improvements vs. complete rewrite with substantial code changes?

This decision is based on developer resources, competitive pressures, user demands, and technical feasibility. Developers tend to resist complete rewrites because they disrupt normal activities. The emergence of radically improved new computational techniques (such as in materials codes) may mandate rewrite if the code is to remain competitive. Developers often respond to demands from users to run larger problems by incremental tuning of the code to permit use of more processors, but this route usually leads to poorer efficiency. The development of new processors, such as graphics processing units, may stimulate substantial rewrite to achieve higher performance.

USER COMMUNITY MODELS

14. What forms of user involvement are most helpful in code development, testing, maintenance, and upgrades?

This question has also been covered previously. The most commonly mentioned forms of user involvement include bug reports, supplying verification/validation data based on their use of the code, and suggestions for improvements and upgrades. Large, active user communities for open source codes also provide powerful endorsement of the value of the code to the organizations that fund the developers and may enhance the ability of the developers to attract funding for special projects or collaborations. Especially for proprietary codes that cater to bench engineers, user experiences can provide ease-of-use data for improving user interfaces. For internal corporate or laboratory codes the close involvement of developers with their users was also considered beneficial. This involvement could come through training courses, help desks, planning meetings, and feedback sessions. Each developer should spend some time in these activities.

Another distinctive characteristic of feature of the European environment is the stronger connection between academia and industry. It is not uncommon for PhD-level students to carry out their dissertation work at a large company, where they are paid as employees and contribute to the development of software products. In the other direction, several speakers discussed initiatives (often at least partly government supported) to introduce engineering software into the educational curriculum. This is seen as having both educational advantages and workforce development benefits.
15. **What types of user community organizations are most helpful to insure that codes in your application area meet user needs and are modified as necessary to meet those needs?**

No particular type of organization was preferred, but developers agreed that vibrant user involvement was desirable. Especially for open source codes with large user communities a multi-level approach was commonly recommended. This includes an online user forum in which users can help each other, an email help facility, a Wiki containing manuals, tutorials and other discussions, user meetings when new releases become available and otherwise periodically, and user training courses (perhaps held at professional meetings.)

**CONCLUSIONS**

The workshop provided strong evidence that scalable engineering codes for realistic problems in several disciplines can be developed that scale to hundreds, and in some cases up to thousands, of processors. However, doing so appears to require that several important factors be favorably aligned. These included an expert and committed development team that pays attention to user needs, stable funding that is provided over more than a decade for development, support and upgrades, use of scalable physics models and algorithms, availability of suitable scalable software libraries and middleware, access to large computers, code licensing terms that make it cost-effective for users to scale up their problems, regular releases that include bug-fixes, improvements, and upgrades to the code, and a large and vibrant developer-maintained user community that provides endorsement, support and feedback.

The European presentations provided a striking insight: European government agencies commonly provide long-term stable funding to companies to develop and maintain scalable engineering codes deemed important to the country. The European codes presented at the workshop were originally developed with government funding at universities or government laboratories. Several of the development teams have since moved to companies, and they have continued to receive government funding combined with private funding. All of these codes are freely available to users who meet certain criteria (such as nationals of the country that developed them or their collaborators); several are freely available to anyone as downloadable open source. The United States government does not typically fund code development in companies other than for specific government purposes, and these codes are not usually available to general users.

Another insight from the discussion is that European funders and developers are comfortable with very liberal licensing terms, in many cases making their codes freely downloadable under the GPL open source license. Even Électricité de France, a government corporation that designs, builds and operates that country's nuclear reactors, has made its production suite of engineering software freely available under the GPL. Some of these codes were developed with partial funding from the French Commissariat à l'Energie Atomique. One of the workshop participants indicated that EDF wished to make the codes more available but didn’t want to be in the business of selling proprietary codes. As to the wisdom of releasing the company's design tools, he replied that the real value is in the data; this is not released.
APPENDIX A. AGENDA

June 2

9:00 Keynote: Why are we here, what are the issues, what we hope to learn
Douglass Post, DoD HPC Mod Office

9:30 Session I: Descriptions of scalable engineering codes
David Nelson (Chair)

Computational Structural Mechanics
Thomas Zwinger, CSC Finland (Elmer)

Climate/Weather/Ocean Modeling
Paul Selwood, Met Office UK (Unified Model)

Integrated Engineering Codes
Christophe Durand, EDF R&D France (Code Aster) (Presented by Bernholdt)

Computational Fluid Dynamics
Hrvoje Jasak, Wikki Ltd. UK (OpenFOAM)

Computational Biology, Chemistry, and Material Science
I.T. Todorov, STFC Daresbury Laboratory UK (DL_POLY)
Paolo Giannozzi, Università di Udine Italy (Quantum ESPRESSO)

1:05 Panel I: Scalable algorithm advances and barriers; development models, including
ease-of-use and workflow factors
Carl Dyka, NSWC – Dahlgren (Chair)

Bert de Jong, PNNL (NWChem)
Paolo Giannozzi, Università di Udine Italy (Quantum ESPRESSO)
Loren Miller, DataMetric Innovations

2:20 Panel II: Funding approaches; intellectual property models and issues
Joseph Gorski, Navy-Carderock, (Chair)

Marvin L. Alme, LANL
Robert Meakin, US Army (CFD)
Thomas Zwinger, CSC Finland

3:55 Panel III: Code maintenance and upgrade; user community involvement
Alex Larzelere (Chair)

Christopher Atwood, SNP (CREATE-AV)
Andre Ribes, EDF France (SALOME)
Gene Poole, CD-ADAPCO
June 3

9:00 Summary: What did we learn yesterday about successful scalable software
   David Bernholdt, ORNL

9:45 Panel IV: Current U.S. situation (algorithmic advances and barriers; development
   models, including ease-of-use and workflow factors; funding models; intellectual
   property models including export controls; maintenance and user community)
   Steven Payne, Navy - CNMOC (Chair)
   Peter Cummings, Vanderbilt U.
   Charbel Farhat, Stanford U.
   Joe Jung, Sandia National Labs
   Laxmikant (Sanjay) Kale, University of Illinois
   Scott Morton, Eglin AFB

11:20 Lessons for creating and maintaining scalable engineering codes
   Doug Kothe, ORNL
APPENDIX B. PARTICIPANTS

Marvin Alme, LANL

Christopher Atwood, CREATE-AV / Scaled Numerical Physics LLC

Ben Benokraitis, WTEC

David Bernholdt, Oak Ridge National Laboratory

Robert Bohn, NCO/NITRD

Clark Cooper, National Science Foundation

Peter Cummings, Vanderbilt U./ORNL

Wibe (Bert) DeJong, Pacific Northwest National Laboratory

Carl Dyka, Naval Surface Warfare Center - Dahlgren

Charbel Farhat, Stanford University

Patricia Foland, WTEC

Paolo Giannozzi, U. of Udine and IOM-CNR

Joseph Gorski, Naval Surface Warfare Center, Carderock

Matthew Grismer, Air Force Research Laboratory

Luanne Handley-Blair, HPCMP CREATE

Angela Harris, HPCMP CREATE

Cray Henry, HPCMP/DOD

Thuc Hoang, DOE NNSA

Geoff Hodlridge, NNCO

Taimak Holland, WTEC

Myles Hurwitz, DoD HPCMP

Leland Jameson, National Science Foundation

Hrvoje Jasak, Wikki Ltd UK

Pat Johnson, Johnson Edits

Joseph Jung, Sandia National Laboratories
Appendix B. Participants

Laxmikant (Sanjay) Kale, U. of Illinois
Douglas Kothe, Oak Ridge National Laboratory
Alex Larzelere, U.S. Department of Energy (NE-71)
Robert Meakin, Pax River NAVAIR
Loren Miller, Data Metric Innovations, Goodyear (retired)
Eduardo Misawa, NSF
Scott Morton, Eglin AFB
David Nelson, WTEC
Ruth Pachter, Air Force Research Laboratory
Steven Payne, Navy
Gene Poole, CD-ADAPCO
Douglass Post, DoD HPCMO
André Ribes, EDF R&D
Thomas F. Russell, NSF
Paul Selwood, Met Office UK
Duane Shelton, WTEC
William Tang, Princeton U.
Ilian Todorov, STFC Daresbury Laboratory
Ralph Wachter, ONR
Thomas Zwinger, CSC - IT Center for Science
APPENDIX C. PRESENTATIONS

International Workshop on Scalable Engineering Software

Dr. Douglass Post
Chief Scientist, DoD High Performance Computing Modernization Program
NSF, Arlington, VA, June 2-3, 2010

Workshop Goals

• What are the workshop goals?
  • Develop a better understanding of how to produce scalable engineering codes, especially engineering codes for the DoD
  • What lessons can be learned from codes developed abroad

• What are the issues?
  • Scalable algorithms—advances and barriers
  • Development Models—ease of use, workflow factors
  • Funding approaches—business model
  • Market Opportunities and requirements
  • User community involvement
  • Intellectual property
  • V&V approach
  • Code maintenance and support

• What are the lessons learned?
Next Generation Computers
Unparalleled Power for Problems

- Next generation computers (2020) will enable us to develop and deploy codes that are much more powerful than present tools:
  - Utilize accurate solution methods
  - Include all the effects we know to be important
  - Model a complete system
  - Complete parameter surveys in hours rather than days to weeks to months
- In ~ 10 years, workstations will be as powerful as today's supercomputers
- Greatest opportunities for 2020 (and 2010) include large-scale codes that integrate many multi-scale effects to model a complete system

A Paradigm Shift in Engineering Is Underway

- Past:
  - Repeated Design→Build→Test Cycles

- Present:
  - Occasionally Augment Past with Limited Single-Physics Analysis with Research Codes on Laptops

- Future:
  - Design Through Analysis, Multi-Physics Design and Analysis with Supercomputer Power
  - Repeated CAD→Mesh→Analyze Cycles followed by a few Design→Build→Test Cycles
Present Systems Engineering
Iterated Design → Build → Test Cycles

Requirements → Design → Build → Test → Market

(Many) Design iterations

- Requires many lengthy and expensive design/build/test iteration loops
- Process converges slowly, if at all
- Design flaws discovered late in process
- \( \rightarrow \) Long time to market

Modern Systems Engineering:
Computational Design Through Analysis

Requirements → Design and mesh Virtual Product → Analyze and Test Virtual Product → Build and Test Physical Product → Market

- Reduced time to market from 3 years to 9 months
- Increased new products delivery from 1 every 3 years to 5 per year
- Saved the company
Physics-based Engineering Software Helped The US Win Cold War.

- Nuclear weapons are complex, expensive, and hard to test
  - ~ 5 to 10 tests per system
- DOE NNSA uses computational tools for:
  - Design development, optimization, & analysis
- DOE NNSA labs own the biggest supercomputers

Testing + Computer Power + Computational Design ➔ Weapon Capability

<table>
<thead>
<tr>
<th>Year</th>
<th>Computer Power</th>
<th>Testing</th>
<th>DOE NNSA uses computational tools for:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1945</td>
<td>0.000000001 GigaFlops/s</td>
<td>Air Tests</td>
<td>Design, development, optimization &amp; analysis</td>
</tr>
<tr>
<td>2010</td>
<td>1,000,000 GigaFlops/s</td>
<td>Underground Testing</td>
<td>DOE NNSA labs own the biggest supercomputers</td>
</tr>
</tbody>
</table>

Increasing Computational Design Capability

- Spatial resolution
- Temporal resolution
- Geometric fidelity
- Physics models

- MIRV (even lighter, smaller)
- SLBM (even lighter, smaller)
- ICBM (lighter, smaller)
- Heavy Hydrogen Bombs

Improved yield to weight

It Takes a Village!

- **Sponsors** ➔ Provide mission, resources and support
  - **Designers**—End-users
    ➔ Engineers to use the tools to design products
  - **Codes**
    ➔ Takes a good team ~ 10 years and ~ $100M to develop a complex code
  - **V&V**
    ➔ Dedicated experiments and tests
  - **Computers**
    ➔ Capability to develop codes and run the problems quickly and conveniently
Developing These Codes Takes A Large Team A Long Time

Begin with a few engineering codes developed abroad

Session I: Descriptions of Scalable Engineering Codes (David Nelson chair)

Computational Structural Mechanics
  - Thomas Zwinger, CSC Finland (Elmer)

• Climate/Weather/Ocean Modeling
  - Selwood, Met Office UK (Unified Model)

• Integrated Engineering Code Suite (given by D. Bernholdt)
  - SALOME_MECA/Code Aster: Christophe Durand, EDF

Computational Fluid Dynamics:
  - Hrvoje Jasak, Wikki Ltd. UK (OpenFOAM)

Computational Biology, Chemistry, and Material Science
  - I.T. Todorov, STFC Daresbury Laboratory UK (DL_POLY)
  - Paolo Giannozzi, Università di Udine Italy (Quantum ESPRESSO)
Assess specific issues

Panel I: Scalable algorithm advances and barriers; Development models, including ease-of-use and workflow factors—(chair: Carl Dyka, Dahlgren, Navy)
- Bert deJong (NWChem)
- Paolo Giannozzi (Quantum ESPRESSO)
- Loren Miller (Goodyear, retired)

Panel II: Funding approaches; Intellectual property models and issues—(chair Joseph Gorski, Carderock, Navy)
- Marvin L. Alme (LANL)
- Robert Meakin (CREATE-AV)
- Thomas Zwinger (CSC)

Panel III: Code maintenance and upgrade; User community involvement—(chair: Alex Larzelere, DOE NE)
- Christopher Atwood (CREATE-AV)
- Andre Ribes (SALOME_MECA)
- Gene Poole (CD-ADAPCO)

Summarize State of US and Draw Conclusions

Summary Talk: What did we learn yesterday about successful scalable software--David Bernholdt, ORNL

Panel IV: Current U.S. Situation—chair: Steven Payne, Navy, NAVO
- Peter Cummings (Vanderbilt)
- Charbel Farhat (Stanford)
- Joe Jung (Sandia Labs, Sierra code suite)
- Laxmikant (Sanjay) Kale, (University of Illinois)
- Scott Morton (CREATE and Eglin AFB)
- (Algorithmic advances and barriers; Development models, including ease-of-use and workflow factors; Funding Models; Intellectual Property Models including export controls; V&V; Maintenance and User community)

Lessons for Creating and maintaining scalable engineering codes--Doug Kothe (ORNL)
Workshop Goals

• Why are the workshop goals?
  • Develop a better understanding of how to produce scalable engineering codes, especially engineering codes for the DoD
  • What lessons can be learned from codes developed abroad

• What are the issues?
  • Scalable algorithms—advances and barriers
  • Development Models—ease of use, workflow factors
  • Funding approaches—business model
  • Market Opportunities and requirements
  • User community involvement
  • Intellectual property
  • V&V approach
  • Code maintenance and support

• What are the lessons learned?
Elmer – Multiphysics Open Source FEM package

Dr. Thomas Zwinger
CSC – IT Center for Science Ltd.
Espoo, Finland

Contents

• A brief history of Elmer
• Elmer users
• Elmer features
• Elmer Modules:
  – ElmerGUI, ElmerSolver, ElmerPost
• Parallel performance
• Verification/Validation
A brief history of Elmer

• Elmer development started in 1995 (National CFD program)
• 9/2005 published under GPL-license
  • ~300 000 lines of code
  • IP remains with CSC
  • Development driven by projects
• 10/2007 Elmer version control put under sourceforce.net ~1000 downloads/month
Elmer features

- Fluid Mechanics: RANS, VMS, Reynolds, free surfaces
- Structural Mechanics: non-linear elasticity, plates
- Heat Transfer: phase change
- Electro-Magnetics
- Acoustics: Helmholtz equation
- Quantum Chemistry: DFT

Elmer modules

ElmerGUI  ElmerSolver  ElmerPost
Elmer Scalability

Parallel Runs: using MPI; interfaces to external libraries: Hypre, MUMPS, Pardiso, SuperLU

Verification/Validation

Comparison between Elmer full-Stokes simulation (left) and mass-balance velocity plot (Bamber, et al. 2001) of present day Greenland ice sheet
Elmer applications

CFD + Heat transfer

Parallel simulation of silicon meltflows using stabilized finite element method (5.4 million elements).

Simulation Juha Ruokolainen, animation Matti Gröhn, CSC

Elmer applications

MEMS

Simulation of the damping oscillations of a perforated micromechanical membrane

P. Råback et al., Hierarchical finite element simulation of perforated plates with arbitrary hole geometries, MSM 2003
Elmer applications
Acoustics

Simulation Mikko Lyly, CSC

Sound waves solved from the Helmholtz equation

Elmer applications
Microfluidics

Electrokinetically driven separation chip

Elmer applications

Hemodynamics

• Cardiovascular diseases leading cause of deaths in western countries
• Calcification reduces elasticity of arteries
• Modeling of blood flow = fluid-structure-interaction


Elmer applications

• Glaciology: Ice sheets
• Answers to open question in IPCC report
• Widely used tool in the area
  – British Antarctic Survey
  – University of Grenoble
  – University of Hokkaido
• Simulations of continental ice sheets very demanding

About the Met Office

• Trading Fund owned by the UK Secretary of State for Defence
• Required to operate on a commercial basis and to meet agreed Key Performance Targets
• Financed by income from its wide mix of customers
• Weather Forecasting and Climate Research
• Operational service and scientific research
• Outputs are subject to Crown Copyright

© Crown copyright Met Office
Unified Model - Technical

- Hours to Centuries, Regional to Global, Simple Forecast to full Earth-System Model
- Atmosphere
  - Lat-Long grid, Non-hydrostatic, semi-Lagrangian, semi-implicit, Arakawa C grid, Charney-Phillips vertical coordinate
- Atmosphere approx 1 million lines of code
- Highly flexible choice of scientific options
- Hybrid MPI/OpenMP parallelisation
- Mostly Fortran 90
- Constant development since early 1990s

UM Evolution
Validation & Verification

- Validation
  - Large suite of community-standard idealised test cases (e.g. shallow water, aqua-planet, vertical slices, standard profiles, etc)
  - Cross comparison against high resolution specific process models, e.g. cloud resolving LEM

- Verification
  - Every forecast has key weather outputs verified against observations and analyses
  - Global inter-centre comparisons of certain parameters
  - Climate verifies against long historical records and ability to reproduce important features e.g. ENSO

Strong Scaling
Appendix C. Presentations

Parallelisation and Scalability in OpenFOAM

Hrvoje Jasak
h.jasak@wikki.co.uk

Wikki Ltd, United Kingdom

International Workshop on Scalable Engineering Software
National Science Foundation, 2 June 2010

Background

Objective
- Review the state and prospects for massive parallelisation in CFD codes, with review of implementation in OpenFOAM

Topics
1. Introduction: Parallelisation of CFD simulation work-flow
2. Components
3. Parallel algorithms
4. Parallelisation and efficiency of the linear algebra toolkit
5. "Auxiliary algorithms" and parallel efficiency
6. Points of interest and summary
Appendix C. Presentations

Parallelisation in CFD

Massively Parallel Computing
- Today, most large-scale CFD solvers rely on distributed-memory parallel computer architectures for all medium-sized and large simulations
- Parallelisation of CFD solvers is complete. If the algorithm does not parallelise, it is not used. Example: wall distance calculation
- Complete simulation workflow is still not parallelised! Bottle-necks:
  - Parallel mesh generation is missing or under development
  - Scaling issues related to the linear algebra toolkit: solver technology
  - Parallel efficiency of “auxiliary algorithms” is sometimes very poor
- User expectation: linear scaling to thousands of CPUs

Parallel Computer Architecture
- Parallel CCM software operates almost exclusively in domain decomposition mode: a large loop (e.g. cell-face loop in the FVM solver) is split into bits and given to a separate CPU. Data dependency is handled explicitly by the software
- Using distributed memory machines with communications overhead; architecture dictates speed of communications and limits scalability
- Handling of multi-core processors sometimes questionable: memory access bandwidth is the limiting factor in serial execution speed

OpenFOAM

OpenFOAM: Computational Continuum Mechanics Library
- OpenFOAM is a free-to-use Open Source numerical simulation software with extensive CFD and multi-physics capabilities. Current source code size: approx 990k lines of object-oriented C++. Parallelisation and performance consideration central to development, from the initial design
- Development Paradigm and Leadership Model
  - Bulk of code written between 1993 and 2004, with two co-authors
  - Currently two branches: closed development by OpenCFD and open source development model supported by Wikki Ltd. (public steering committee and contributions; open Workshop and training events, interest groups)
  - Two leadership models: Single reference (OpenCFD) or public committee with contributors and review by trusted individuals
- Validation and verification of OpenFOAM: Mirroring Commercial Verification Effort
  - Numerics kernel is stable and under constant verification (e.g. tet mesh)
  - Bulk of physical models developed under PhD project: validation and tutorials
  - In practice, validation and verification is done on client side
  - Work-in-progress: public regression test loops and model validation in automated manner, governed by Special Interest Groups (SIG)
Appendix C. Presentations

Components

Parallel Components and Functionality
1. Parallel communication wrapper
   - Basic information about the run-time environment: serial or parallel execution,
     number of processors, process IDs etc.
   - Passing information in transparent and protocol-independent manner
   - Optimised global gather-scatter communication operations
2. Mesh-related operations
   - Mesh and data decomposition and reconstruction
   - Global mesh information, e.g. global mesh size, bounding box etc.
   - Handling patched pairwise communications
   - Processor topology communication scheduling data
3. Discretisation support
   - Operator and matrix assembly: executing discretisation operations in parallel
   - Data updates across processor boundaries: data consistency
4. Linear equation solver support (highest impact on solver performance!)
5. Auxiliary operations, e.g. messaging or algorithmic communications, non-field
   algorithms (e.g. particle tracking), data input-output, solution sampling and
   acquisition of (point, line, surface) post-processing data

Parallel Algorithms

Zero Halo Layer Approach in Discretisation
- Traditionally, FVM parallelisation uses the halo layer approach: data for cells next
  to a processor boundary is duplicated. Halo layer covers all processor boundaries
  and is explicitly updated through parallel communications calls: prescribed
  communications pattern, at pre-defined points
- OpenFOAM operates in zero halo layer approach: flexibility in communication
  pattern, separate setup for FVM and FEM solvers
- FVM and FEM operations “look parallel” without data dependency: perfect scaling

![Diagram of subdomains and decomposition](chart.png)
Appendix C. Presentations

**Linear Algebra Toolkit**

Matrix Assembly and Solution
- Performance of linear solvers practically dictates parallel scaling in CFD
- In terms of code organisation, each sub-domain creates its own numbering space; locally, equation numbering always starts with zero and one cannot rely on global numbering; it breaks parallel efficiency
- Processor interfaces are updated separately, involving communications
- Explicit codes/operations scale well: no data dependency
- Implicit algorithms are \( \approx 100 \) times faster but involve parallelised linear algebra

Choice of Linear Equation Solvers
- As a rule, Krylov space solvers parallelise naturally: global updates on scaling and residual combined with local vector-matrix operations: global sum
- Algebraic Multigrid (AMG) performs much worse due to coarse level hierarchy; balance of work and communications at coarse levels
- ... but AMG is intrinsically 3 times faster in the number of operations
- Currently, all algorithms assume uniform communications performance across the machine; improvement is needed but may require complete algorithmic rewrite
- **Outlook:** an (unknown) new approach is needed to achieve performance scaling

---

**Auxiliary Algorithms**

Efficiency in Auxiliary Algorithms
- Overall, parallelisation efficiency is satisfactory on low 000s of CPUs for field-based operations: explicit field algebra, matrix assembly and linear solvers
- Parallelisation of some components is bound to domain decomposition but operations are not field-based: surface physics, particle tracking, patch integrals
- Massive load imbalance or inappropriate parallelisation limits performance

Example: Lagrangian Particle Tracking in Parallel
- Particle tracking algorithm operates on each segment of decomposed mesh by tracking local particles. Load balancing issues: particles are not fields!
- Processor boundary interaction: a particle is migrated to connecting processor
# Points of Interest

**Current Status: Performance Scaling**
- Implicit general purpose CFD solvers work reasonably well up to low 000s of CPUs
- Current approach is inappropriate for inhomogeneous communication speed and an order of magnitude increase in computer size
- Communication speed and memory access are critical; clusters must be co-located with fast inter-connect; moving jobs around is not practical
- Limiting factors in performance scale-up:
  - Parallelisation of complete CFD process, from geometry to post-processing
  - Iterative sparse matrix solver algorithms
  - Non-field operations in CFD algorithms: spray, radiation, surface physics

**Summary**
- Current CFD technology involving implicit solvers is close to its limit
- Some necessary algorithms are badly parallelised due to requirement of method-to-method coupling
- New approach is needed: how to do massively parallel linear algebra?
- We may remain at the level of 000s of CPUs for a long time, looking for other avenues of improvement: multi-core CPU, handling inhomogeneous communications speed, fast memory access, complete migration to GPU
DL_POLY: Software Solutions in Molecular Dynamics

I.T. Todorov & W. Smith
ARC Group & CC Group

CSED, STFC Daresbury Laboratory, Daresbury
Warrington WA4 1EP, Cheshire, England, UK

Where is Daresbury?

In the realm of Alice's Wonderland (1865)

Lewis Carroll

Charles Lutwidge Dodgson
DL_POLY Project Background

- **General purpose parallel (classical) MD simulation software** *(VERIFICATION)*
  - It was conceived to meet needs of CCP5 - The Computer Simulation of Condensed Phases (academic collaboration community)
    - 1994 – now: DL_POLY_2 (RD) by W. Smith & T.R. Forester (funded for 6 years by EPSRC at DL) *(FUNDING MODEL)*
    - 2003 – now: DL_POLY_3 (DD) by I.T. Todorov & W. Smith (funded for 4 years by NERC at Cambridge) *(FUNDING MODEL)*
  - Over 7,500 licences taken out since 1994 *(WORKSHOPS & SUMMER SCHOOLS)*
  - Over 850 registered FORUM members since 2005
  - Available free of charge (under licence) to University researchers (provided as code) and at cost to industry

DL_POLY Licensing and Support

- Online Licence Facility at [http://www.ccp5.ac.uk/DL_POLY](http://www.ccp5.ac.uk/DL_POLY)
  - The licence is
    - To protect copyright of Daresbury laboratory
    - To reserve commercial rights
    - To provide documentary evidence justifying continued support by UK Research Councils
  - It covers all DL_POLY software
  - Registered users are entered on the DL_POLY e-mailing list
  - Support is available (under CCP5 and EPSRC service level agreement to CSED) *only* to UK academic researchers
  - For the rest of the world there is the online user FORUM
  - Last but not least there is a detailed, interactive, self-referencing PDF (LaTeX) user manual
Appendix C. Presentations

**DL_POLY Licence Statistics - I**

![](chart1)

**DL_POLY Licence Statistics**

![](chart2)
DL_POLY Licence Statistics

DL_POLY Licences 2008 by Countries

- UK: 11%
- China: 13%
- USA: 19%
- France: 4%
- India: 9%
- Iran: 5%
- Other: 40%

- Japan: 3%
- Germany: 3%
- Spain: 3%
- Italy: 3%
- Brazil: 1%
- Algeria: 1%
- Turkey: 1%
- Russia: 2%
- Canada: 2%
- Mexico: 2%
- South Korea: 1%
- Greece: 1%
- The Netherlands: 1%
- Australia: 1%
- Brazil: 1%
- Taiwan: 2%
- Sweden: 1%
- Poland: 1%
- Australia: 1%
- Singapore: 1%
- Japan: 3%
- France: 4%

DL_POLY Licences 2009 by Countries

- USA: 19%
- China: 17%
- UK: 11%
- India: 8%
- Iran: 5%
- Other: 42%

- Japan: 3%
- France: 4%
- Brazil: 1%
- Mexico: 1%
- Canada: 1%
- Taiwan: 2%
- Russia: 2%
- South Korea: 2%
- Spain: 2%
- Italy: 2%
- Germany: 3%
- Australia: 1%
- Singapore: 1%
- Poland: 1%
- Sweden: 1%
- Japan: 3%
- France: 4%
- REST OF WORLD: 14%
Written in modularised free formatted F90 (+MPI) with rigorous code syntax (FORCHECK and NAGWare verified), no external library dependencies

**· DL_POLY_2 (version 20)**
- **Replicated Data** parallelisation, limits up to ≈30,000 atoms with good parallelisation up to 64 (system dependent) processors (running on any processor count)
- Full force field and molecular description
- Hyper-dynamics, Temperature Accelerated Dynamics, Solvation Dynamics
- Free format reading with somewhat rigid semantics

**· DL_POLY_3 (version 10)**
- **Domain Decomposition** parallelisation, limits up to ≈2.1 × 10^9 atoms with inherent parallelisation (any high processor count, 2^k restriction applies for SPME)
- Full force field and molecular description but no rigid body description (available in DL_POLY_4, release in July 2010)
- Free format semantically approached reading with some fail-safe features and basic reporting (but fully fool-proofed)
Appendix C. Presentations

**DL_POLY_2:**
- Hautman-Klein 2D Ewald
- Truncated octahedral & Rhombic dodecahedral PBC
- Temperature Accelerated Dynamics
- Bias Potential Hyperdynamics
- Solvation features
  - Molecular energy decomposition
  - Thermodynamic integration (free energy)
  - Solvent induced spectral shifts

**DL_POLY_3:**
- Radiation damage defects analysis, boundary thermostats, volumetric expansion, replay history
- Variable time step algorithm
- Langevin, Andersen and MTK ensembles
- Parallel I/O
### DL_POLY_3 Development Statistics

- **DL_POLY_3.00**
- **DL_POLY_3.04**
- **DL_POLY_4.00**

**Graph Details:**
- Y-axis: Number of lines of code (x1,000)
- X-axis: Years since 2003
- Key points:
  - Released
  - Referenced

### Performance Weak Scaling on IBM p575 (2005-2010)

- **Solid Ar (32'000 atoms per CPU)**
- **NaCl (27'000 ions per CPU)**
- **SPC Water (20'736 ions per CPU)**

**Graph Details:**
- Y-axis: Speed Gain
- X-axis: Processor Count
- Key points:
  - Perfect parallelisation
  - Good parallelisation
  - 33 million atoms
  - 28 million atoms
  - 21 million atoms
  - Max load per 1GB/CPU:
    - 700'000 atoms
    - 220'000 ions
    - 210'000 ions

---

Appendix C. Presentations
Rigid Bodies versus Constraints
450,000 particles with DL_POLY_4

Scaling

ICE7
ICE7_CB

Benchmarking BG/L Jülich
2007

14.6 million particle Gd$_2$Zr$_2$O$_7$ system

Perfect
MD step total
Link cells
van der Waals
Ewald real
Ewald k-space

Processor count

Speed Gain
Benchmarking Main Platforms

![Graph showing evaluations vs. processor count for different platforms]

- CRAY XT4 SC
- CRAY XT4 DC
- CRAY XT3 SC / IBM P6+
- CRAY XT3 DC
- BG/L
- BG/P
- IBM p575
- 3GHz Woodcrest DC

3.8 million particle Gd₂Zr₂O₇ system

HPC Evolution & Challenges

- PIII 833 MHz 0.8 GF/s (year 2000)
- Woodcrest 3.8 GHz 31.2 GF/s (year 2010)

~ 40 fold performance increase in 10 years, which almost follows Moore's Law. Only the emergence of multi-core chips 2007-2008 saved Moore's Law per chip but not per processing element as the power and memory per chip decreased. Pressure on programming models!

- Myrinet 18 μs 0.14 GB/s (year 2000)
- Infiniband 0.2 μs 40 GB/s (year 2010)

~ 25 fold improvement with the interconnect.

Memory and RAID card performance have also increased by a factor of 20 over the same period of time.
However, MD simulation sizes have increased 1000 fold, something that the I/O infrastructure cannot cope with by traditional means (MPI-I/O). Memory & I/O bottlenecks. Emergence of I/O subsystems.

“Raw computational power has increased more than any other HPC system parameters”. This makes application scaling and comparison to old benchmarks more difficult today.

However, it is the emergence of the heterogeneous platforms with Cells and/or GPUs that really pushed the change of conceptual models of programming. This makes codes structure alienate from scientific methodology and consequently impose a toll on verification and further development.

Proof of Concept on IBM p575 2006

300,763,000 NaCl with full SPME electrostatics evaluation on 1024 CPU cores

- Start-up time \( \approx 1 \) hour
- Timestep time \( \approx 68 \) seconds
- FFT evaluation \( \approx 55 \) seconds
- Shut-down time \( \approx 3 \) hours

In theory, the system can be seen by the eye. Although you would need a very good microscope – the MD cell size for this system is 2\( \mu \)m along the side and as the wavelength of the visible light is 0.5\( \mu \)m so it should be theoretically possible.
### I/O Weak Scaling on IBM p575

**2005-2008**

- **Solid Ar**
- **NaCl**
- **SPC Water**

Dashed lines show shut-down times, solid lines show start-up times.

**Time [s]**

**Processor Count**

### Parallel I/O on 250,000 NaCl

- **3.09: Compute Only**
- **3.09: Compute + Write**
- **3.10: Compute Only**
- **3.10: Compute + Write**

**Timesteps/s**

**Number Of Cores**
### Writing Performance for 216,000 Ions

<table>
<thead>
<tr>
<th>Cores</th>
<th>I/O Procs</th>
<th>Time/s</th>
<th>Time/s</th>
<th>Mbyte/s</th>
<th>Mbyte/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>32</td>
<td>143.30</td>
<td>1.27</td>
<td>0.44</td>
<td>49.78</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>48.99</td>
<td>0.49</td>
<td>1.29</td>
<td>128.46</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>39.59</td>
<td>0.53</td>
<td>1.59</td>
<td>118.11</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>68.08</td>
<td>0.43</td>
<td>0.93</td>
<td>147.71</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>113.97</td>
<td>1.33</td>
<td>0.55</td>
<td>47.60</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>112.79</td>
<td>1.20</td>
<td>0.56</td>
<td>52.47</td>
</tr>
<tr>
<td>2048</td>
<td>512</td>
<td>135.97</td>
<td>0.95</td>
<td>0.46</td>
<td>66.39</td>
</tr>
</tbody>
</table>

### Reading Performance for 216,000 Ions

<table>
<thead>
<tr>
<th>Cores</th>
<th>I/O Procs</th>
<th>Time/s</th>
<th>Time/s</th>
<th>Mbyte/s</th>
<th>Mbyte/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>16</td>
<td>3.71</td>
<td>0.29</td>
<td>17.01</td>
<td>219.76</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>3.65</td>
<td>0.30</td>
<td>17.28</td>
<td>211.65</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>3.56</td>
<td>0.22</td>
<td>17.74</td>
<td>290.65</td>
</tr>
<tr>
<td>256</td>
<td>32</td>
<td>3.71</td>
<td>0.30</td>
<td>16.98</td>
<td>213.08</td>
</tr>
<tr>
<td>512</td>
<td>64</td>
<td>3.60</td>
<td>0.48</td>
<td>17.53</td>
<td>130.31</td>
</tr>
<tr>
<td>1024</td>
<td>64</td>
<td>3.64</td>
<td>0.71</td>
<td>17.32</td>
<td>88.96</td>
</tr>
<tr>
<td>2048</td>
<td>128</td>
<td>3.75</td>
<td>1.28</td>
<td>16.84</td>
<td>49.31</td>
</tr>
</tbody>
</table>
Reading Performance for 1,728,000 Ions

<table>
<thead>
<tr>
<th>Time/s</th>
<th>Mbyte/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.15</td>
<td>437.24</td>
</tr>
<tr>
<td>1.02</td>
<td>494.68</td>
</tr>
<tr>
<td>1.22</td>
<td>414.61</td>
</tr>
<tr>
<td>1.56</td>
<td>323.24</td>
</tr>
<tr>
<td>3.64</td>
<td>138.73</td>
</tr>
<tr>
<td>5.64</td>
<td>89.48</td>
</tr>
</tbody>
</table>

Future Plans

The Future - DL_POLY_4

- GPU CUDA port. (ICHEC)
- MS Windows PC port. (Microsoft)
- dCSE funded by EPSRC & NAG Ltd.
  - Rigid Bodies (allowing for frozen sites)
  - 3D FFT for small powers of 3 and 5
  - Uncoupling cutoff from DD
  - Dynamic load balancing
Load Balancing

Non uniform density

Unrestricted
Cut off

Appendix C. Presentations

DL_POLY Spin-offs

• DL_MULTI – Distributed multipoles (polimorphs search) (M. Leslie rtd., J. Purton DL_MONTE – Monte Carlo)

• DL_FILED – AMBER AND CHARMS to DL_POLY FIELD generator (Chin Yong)

• DL_MESO – DPD (M. Seaton)

• DL_PROTEIN – Bio-simulations (Roma – Cozzini & Melchionna)

• DL_PATH – Path Integral (prototype)
Thanks to

- WTEC Organisers
- Bill Smith (STFC)
- Ian Bush (NAG Ltd.)
- Mike Ashworth (STFC)
- Laurence Ellison (STFC)

http://www.ccp5.ac.uk/DL_POLY/
Quantum ESPRESSO

Paolo Giannozzi
Università di Udine Italy

The Quantum ESPRESSO distribution

QUANTUM ESPRESSO (opEn-source Package for Research in Electronic Structure, Simulation, and Optimization): an integrated suite of software for atomistic simulations based on electronic structure, using density-functional theory, a plane-wave basis set, pseudopotentials (norm-conserving, ultrasoft, or PAW). License: GNU GPL.

Usage: academic research in materials science, nanotechnology, solid-state physics; also theoretical and biological chemistry, engineering, geophysics. Estimated usage:
\( \sim 800 \) downloads since the latest release (v.4.2, 10 May 2010)
\( \sim 1100 \) (true!) subscribers to the mailing list (average: > 10 posts a day)
\( \sim 100 \) citations of the paper documenting QUANTUM ESPRESSO in 9 months.
Used as benchmark in European supercomputing initiatives DEISA and PRACE

Development: coordinated by a small (\( \sim 10 \)) informal group of scientists, with the help of a larger community of contributors and users

Support: mainly from SISSA-ISAS and from Democritos National Simulation Center (Trieste, Italy) via a dedicated research position and a technical staff. The CINECA supercomputing center provides support for massive parallelization and porting. Informal support from several other research groups worldwide.
Appendix C. Presentations

Rationale for push towards increased scalability

- increase the size of the systems that can be studied with first-principle methods (i.e. from electronic structure) \( \implies \) more RAM and CPU needed
- increase the accuracy and the predictive power of first-principle methods \( \implies \) new methods requiring more complex calculations, longer simulation times

In the beginning \( (1985 \div 1994) \): development of basic methods, iterative techniques, optimization with libraries and with vectorization – uncoordinated efforts

The dawn of Parallel Age \( (1994 \div 1997) \): first coordinated efforts towards scalable parallelization, with parallel (MPI) distributed load-balanced 3D FFT

Public Release \( (1998 \div 2002) \): more effective coordination of efforts between scientists in different groups and their codes - beginning of Quantum ESPRESSO

Parallelism everywhere \( (2002 \div 2010) \): additional parallelization levels, more memory scalability via distribution of more arrays, mixed MPI-OpenMP parallelization

---

Scalability for “small” systems

Scalability strongly depends upon the kind of system under examination! Max number of processors for satisfactory scalability limited by system “size”, i.e. number of atoms, number of electrons, dimensions of the simulation cell. Coarse, i.e. “easy” parallelization of loosely-coupled, semi-independent calculations, not considered here.

Typical speedup vs number of processors:

128 Water molecules (1024 electrons) in a cubic box 13.35Å side, \( \Gamma \) point. Pwscf code on a SP6 machine, MPI only.
\( ntg=1 \) parallelization on plane waves only
\( ntg=4 \) also on electron states

CPU time is mostly spent in FFT and in linear algebra operations
Appendix C. Presentations

Scalability for “medium-size” systems

PS/WAT: Thiol-covered gold surface and water, 4 k-points, 10.59 x 20.53 x 32.66 Å³ cell, 587 atoms, 2552 electrons. Pilact code on CRAY XT4, parallelized on plane waves, electron states, k-points. MPI only.

CNT(1): nanotube functionalized with porphyrins, Γ point, 1532 atoms, 5232 electrons. Pilact code on CRAY XT4, parallelized on plane waves and electron states, MPI only.

CNT(2): same system as for CNT(1). CP code on a Cray XT3, MPI only.

3D parallel distributed FFT is the main bottleneck. Additional parallelization levels (on electron states, on k-points if available) allow to extend scalability.

Mixed MPI-OpenMP scalability

Two models of graphene on Ir surface on a BlueGene/P using 4 processes per computing node. Execution times in s, initialization + 1 self-consistency step.

<table>
<thead>
<tr>
<th>N cores</th>
<th>T CPU (wall)</th>
<th>T CPU (wall)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>2561(2914)</td>
<td>2561(2914)</td>
</tr>
<tr>
<td>512</td>
<td>5081(2914)</td>
<td>5081(2914)</td>
</tr>
<tr>
<td>1048</td>
<td>10261(2914)</td>
<td>10261(2914)</td>
</tr>
<tr>
<td>16384</td>
<td>441(515)</td>
<td>441(515)</td>
</tr>
<tr>
<td>16384</td>
<td>887(1028)</td>
<td>887(1028)</td>
</tr>
<tr>
<td>16384</td>
<td>1333(1351)</td>
<td>1333(1351)</td>
</tr>
<tr>
<td>16384</td>
<td>1780(1780)</td>
<td>1780(1780)</td>
</tr>
</tbody>
</table>

Fragment of an Aβ-peptide in water containing 838 atoms and 2312 electrons in a 22.1x22.9x19.9 Å³ cell, Γ-point. CP code on BlueGene/P, 4 processes per computing node.

(the large difference between CPU and wall time is likely due to I/O)
HPC for industrial use

EDF’s software policy for structural mechanics

International Workshop on Scalable Engineering Software
June 2-3, 2010

Christophe Durand EDF R&D - France

Code_Aster in several words

- An all-purpose FEA software for structural mechanics:

- Plugged in a user-friendly inter-operable environment: Salome_Meca
Code_Aster in several figures

60 releases each year

1,250 documents freely available (17,000 pages)

15 PHD for the 8 last years

250 new features in the code in 2008

2,300 tests runned for each release

CODE_ASTER as a GPL free software since 2001:

50,000 downloads, more than 10,000 hits each week

Mechanics: characterization of scalability

Domains of massive scalability (>500 procs):
- CFD
- Explicit structural mechanics
- Implicit structural mechanics for 3D models in linear elasticity

Moderate scalability (50-100 procs):
- Implicit structural mechanics
- Models mixing 3D and structural elements
- Non linear behaviour
- Contact
- Multi-point constraints and rigid body elements

Code_Aster purpose
Scalability of Code_Aster: common use

- **Parallelism for all users**
  - Switching a key-word (SOLVEUR) + choosing the number of procs
  - Distribution of FEM elemental contributions
  - Parallel resolution of linear systems
  - Iso-fonctionality with the sequential usage

- **MUMPS v4.9.2**
  - Numerous features
    - Real, complex, simple or double precision
    - Analysis, factorization, solving
    - Out-of-core
    - Null pivot detections

- **Parallelism is becoming into common use**
  - 40% of EDF’s computations in mechanics are

Scalability of Code_Aster

- **Example: a model of a pressure vessel**
  - 0.86 M dof with 11000 MPC
  - 71 M non-zero terms in the matrix, \(\text{cond}(K) \sim 10^9\)
  - Speed-up of 10 on 32 procs
  - 40 minutes CPU reduced to 4 minutes
Scalability of Code_Aster

- Other solutions for experimented users
  - FETI-1
  - DD solver with rigid body modes detection and lumped preconditioning
  - PETSc
    - Library: iterative solvers and parallel preconditioners

- Not a popular success (yet)
  - More difficult for the average user
  - Non compatible with all the (numerous) code features
  - Lack of robustness

- But good performances
  - polycristal: 3.3 M dof
  - MUMPS 1proc: 11h
  - PETSC 1proc: 55 min
    - 4 proc: 13 min

Improving software performances: 1st need

- When even small models are costly: highly non-linear, complex phenomenons
  - Behaviour laws, coupling different physics, contact, models with rich mechanical content
  - Long and fine transients (seismic simulation, plane crash): drastic multiplication factor
  - Challenge: from academic models and sketches to usable simulation for industry

Simulation of seismic load

Hujeux law: shear behaviour of clays and sands
22 parameters for the law

50,000 DOF 98% of CPU spent in behaviour resolution
Improving software performances: 2nd need

- When you cannot idealize (too much) the structure: big FEM models
  - The valuable information is at a fine local scale (e.g., tracking crack growth)
  - The problem is implacably multiscale. Substructuring is impossible or too simplifying: global loads, local interest, feedback from local to global
  - Challenge: brute performance on big meshes, parallel solvers, confinement of non-linear models in the zone of interest.

Improving software performances: 2nd need

- When you cannot idealize (too much) the structure: big FEM models

Elastoplasticity with creep and swelling induced by irradiation
From 4 meters to 20 millimeters
Unilateral contact on each of 200 screws

6.7 M DOF without contact: OK
6.7 M DOF with 50000 contact nodes: challenge
0.8 M DOF with 15000 contact nodes: OK
Free software distribution: intents and motivations

- Recognition by the use
  > 50,000 downloads, living community structured by the forum: industry, research, teaching. Multiplication of uses by number and variety.

- Qualification
  Benchmarks about performances and abilities. Feedbacks in a bug-tracker.

- Contributions
  Modular architecture making constitutive laws and finite elements easy to code and plug.

- Spreading of competences
  Academic and industrial partners, providers, students and PhDs.

But most of all:

- An open-source base for cooperations
  Developments at shared costs. Easier capitalization for PhD works and researches. Software base for industrial and scientific partnerships.

For more information and download:

www.code-aster.org

2006 french award: best open-source project for industry.
EDF : a world leader nuclear utility with strong R&D involvement

- European leader of CO2-free energy production, 58 standardized nuclear units, industrial architect, responsible for nuclear safety
- Technical and scientific involvement through large engineering and R&D Divisions: 1M€/day invested in research.

Simulation challenges guided by industrial needs

**Permanent objectives**
- guarantee safety
- improve performances/costs
- maintain assets

**Changing operating conditions**
- face unexpected events, ageing issues, maintenance
- improve performance through new technologies, new operating modes and system wide optimization
- adapt to evolving set of rules (safety, environment, regulatory)

**In-house technical backing**
- expertise: strong Engineering and R&D Divisions
- physical testing and simulation are key tools from the outset

« Living » systems

40 -100 years operation
NWChem: Pushing the scientific envelope on large computing platforms

Bert de Jong
High Performance Software
Molecular Science Computing Facility

What is NWChem?

- Provides major modeling and simulation capability for molecular science
  - Broad range of molecules, including biomolecules, nanoparticles and heavy elements
  - Electronic structure of molecules (non-relativistic, relativistic, ECPs, first and second derivatives)
  - Extensive solid state capability (DFT plane-wave, CPMD)
  - Molecular dynamics, molecular mechanics

[Images of molecular structures]
**NWChem Design and Distribution**

- Designed and developed to be a highly efficient and portable Massively Parallel computational chemistry package
  - Uses Global Arrays Toolkit as parallel infrastructure
  - Data centric model

- Provides computational chemistry solutions that are scalable with respect to chemical system size as well as MPP hardware size

- Emphasis on modularity, portability, and integration
  - i.e., developers can work independently

- Portable – runs on a wide range of computers
  - Supercomputer to Mac or PC with Windows
  - Now runs efficiently on IBM BlueGene, Cray XT, InfiniBand

- Freely available, world-wide distribution
  - 70% is academia, rest government labs and industry

---

**NWChem Software Refresh Strategy**

- Evolving Science Drivers. Input gathered from:
  1) Feedback from users
  2) EMSL Greenbook

- Implementation of the science needs. Porting to new architectures, Performance tuning on the hardware. Support/maintenance phase.

- New SW features/methods made available to users through a yearly release process. Testing phase preliminary to the distribution phase.
Validation & Verification

- Validation
  - New capabilities:
    - Comparison to other codes
    - Comparison of parts with other known properly working code pieces
  - Release cycle:
    - QA suite testing (almost) all functionality of released capabilities
    - New capabilities require addition to QA suite with reference data

- Verification
  - Benchmarking of approximate methods to accurate methods at small scale
  - Comparison to and prediction of experimental data

- Documentation, documentation, documentation

What is next for NWChem

- Scientific drivers:
  - Kinetics and dynamics of chemical transformations
  - Chemistry in condensed phase and at interfaces
  - Spanning longer times for biological systems

- Extreme scale (beyond petascale):
  - Software infrastructure and architecture (re)design
    - 100,000s of complex compute nodes
      - Each with multiple (heterogeneous) cores
    - Computing paradigm is changing drastically
    - Actor based models for efficient load balancing
    - Applications need fault resilience and/or tolerance

- Sustained funding model is essential!
Achieving scalability takes a large team

- Extreme scaling requires a collaborative multidisciplinary development team
  - Computational chemists
  - Computer scientists
  - Mathematicians

- PNNL’s eXtreme Scale Computing Initiative leads the way
  - Scalable computational chemistry methods
  - Fault tolerance at the application level
  - Scalable software infrastructure

What is next for NWChem

- Math:
  - Eigen solvers
    - Scalable
    - Converging (with more linear dependency)
    - Preconditioning
  - Bridging multiple scales
  - Real space methodologies

- Computer science:
  - Stateless task-based compute models needed to achieve performance
  - Fault tolerant software infrastructure
  - Heterogeneous computing
**Scalability needs**

- **Math:**
  - Eigen solvers
    - Scalable
    - Converging (with more linear dependency)
    - Preconditioning
  - Bridging multiple scales
  - Real space methodologies

- **Computer science:**
  - Stateless task based compute models
  - Fault tolerant software infrastructure
  - Heterogeneous computing

---

**NWChem: Going Open-Source**

- NWChem consortium to deliver code and infrastructure for computational chemistry community to build upon
- Greatly expands user base through ease of access by students
- Expanding developers base by leveraging development and resources of universities and national labs
- Establish more collaborative development environment
- License will be Educational Community License (ECL)
Environmental Molecular Sciences Laboratory

A national scientific user facility integrating experimental and computational resources for discovery and technological innovation

www.emsl.pnl.gov

EMSL | Office of Science | Pacific Northwest National Laboratory
Quantum ESPRESSO Development

Paolo Giannozzi
Università di Udine Italy

Quantum ESPRESSO development

- QUANTUM ESPRESSO is mostly developed by scientists with little computer-science sophistication; even the most computer-oriented developers are scientists active in the field of electronic-structure computation

- Help from computer-science people via CINECA: mixed MPI-OpenMP a recent important contribution (CINECA + a computer science master student) is

- Same codes from single PCs to BlueGene – no specialized codes for massively scalable calculations. QUANTUM ESPRESSO is general-purpose software

- Contributions from the community of users are welcome and encouraged
Innovation, validation, verification

- **Verification** – a scientific problem that will always be open: QUANTUM ESPRESSO aims at innovation (i.e. new theories and new methods). Verification is a task for the scientific community at large, not only for specialists of computation: help from non-specialists, e.g. experimentalists desiring to understand their measurements, is important $\implies$ ease of use is an important goal.

- **Validation** – Open-source approach helps a lot. All sources are available, including those under development that can thus be tested before a release. Also important:
  - **Portability**: potential hidden problems are more easily revealed by compilation and execution on many different architectures.
  - **Interoperability** with other codes: gives the possibility to reproduce our calculations with different software $\implies$ data and file exchange.

Tools/middleware used in development

- **Multiple compilers / compilation options**

- **Various analysis tools have proven useful:**
  - valgrind: debugging/run-time check
  - gprof (GNU), HPM, Xprofiler (IBM), craypat (cray): profilers
  - vtune (Intel): profiler, will be tested soon
  - Scalasca (developed at Jülich): MPI profiler

- **Something that would be useful:**
  - an OpenMP profiler, i.e. showing detailed statistics on performances and overhead of OpenMP threads
Appendix C. Presentations

Advances in scalability - important factors

Important algorithmic advances (in conjunction with methodological advances):

- FFT + iterative techniques
  - iterative diagonalization
  - iterative solution of self-consistent equations
  - global minimization (e.g. Conjugate Gradient)
- linear algebra (matrix-vector, matrix-matrix multiplication, matrix diagonalization)

Important algorithmic and technical advances for scalability:

- 3D load-balanced parallel FFT on distributed arrays
- parallel matrix-matrix multiplication, matrix diagonalization (ScaLapack)
- MPI + fast communication hardware
- Mixed MPI-OpenMP

Scalability bottleneck and a look at the future

Current bottlenecks for scalability:

- parallel FFT, subspace diagonalization
- Input-output is quickly becoming a limiting factor
- in general: complex data structure and workflow

Future developments leading to better scalability:

- methodological advances: order-N algorithms based on Wannier functions or real-space approaches? "multiple-scale" (⇒ interoperability with other codes)?
- non-blocking algorithms overlapping calculations and communications
- faster interconnects, especially with reduced latency time
- usage of various kinds of floating-point accelerators (e.g. GPU)
The Goodyear Story

- Working in collaboration with Sandia National Laboratories, The Goodyear converted from prototype-based to simulation-based engineering design over 12 years.
- SBE provides Goodyear significant competitive advantage.
  - Reduced development time-to-market from three years to less than one.
  - Reduced prototype build/test costs by $100 million annually.
  - Dramatically increased rate of new product introductions.
Establish the Need for Change

• Are management and staff convinced that there’s a problem that must be solved?
• Personal and organizational issues can override the merits of major change efforts.
  – “If that’s such a good idea, we’d already be doing it.”
  – “I know modeling and simulation are coming, but I hope I retire before we have to implement them.”
  – “You must use my analysis tool...”

Document Existing Process Flows

Identify Points of Maximum Leverage

• What are the most time-consuming, expensive, personnel intensive/boring, or error prone process steps?
• What software is already available?
  – Does it fit with your architecture, languages, procedures, data structures,…?
  – Why not use it?
• What are the inherent advantages/disadvantages and risks of custom vs. off-the-shelf software as a solution to this business problem?
• Do a decision analysis on the alternatives.

Insulate the User

• Examine in detail the way users interact with your prototype software.
  – Do they “intuitively” go to the “wrong” place?
  – If so, consider moving menu picks to the places where they consider the menu picks should be.
• Insulate users from details they don’t need.
  – For standardized tests, enter the corresponding menu picks automatically.
  – For non-standard tests by advanced users, permit manual overrides.
• Monitor time per step and failure rates throughout the process.
  – Correct problems before the production release.
  – Use statistics to guide changes for the next production release.
• Train, train, train,...
Risk Reduction: V&V

- Bugs will always exist. Manage their impact.
- Standardize the analysis process.
  - Multilevel interface
  - Monitor
- Verification was the responsibility of Goodyear and Sandia code developers.
  - Mathematical accuracy was assessed with respect to experimental data and previously validated codes.
  - Bugs were corrected through vigorous use and reporting of code behavior. (Monitor the process.)
- Validation was the responsibility of the “physics group.”
  - Primarily experimentalists with PhD’s in engineering mechanics
  - Skeptical, but capable of assessing whether code represented reality
- Credibility tipping point: when the software revealed a previously unknown physical phenomena
I’m on the panel because

- **Antero Project Leader, 2001 – 2003**
  - One of four large ASC Code development projects at LANL
  - Terminated (four to two down selection) in 2003

- **ASC Integrated Codes Program Manager, 2004-2008**
  - The other principal in the Antero Project, Bob Webster, became the Group Leader of the Code Group in 2004
  - The Antero Project was gone, but the Antero Management team was leading the ASC Code Development

- **Thermonuclear Burn Initiative Program Leader, 2006-2009**
  - Discovery Simulations at very high resolution
    - First 2D, now 3D (hydro-centric)
    - Focus has turned to higher fidelity physics models
What are the most important comments/observations I can make to this audience in the next 10 minutes?

- Edward Teller and his definition of “experts” - Alme is one of them
- The ASC Program is now a decade and a half old
- The resources available to the program have been significant
- The weapons performance codes tackle an extremely complex problem (regard this as a peek at the future)
  - Multiple time scales
  - Macro timescales run from microseconds to nanoseconds
  - Micro timescales can vary by six orders of magnitude
  - Three 3D phase spaces at every point in physical space
  - Extremely non-linear interactions of important physical mechanisms

- Major changes in the programming models are upon us, and our code bases are getting “long in the tooth”

Lessons

- Need to be cognizant of architecture and structure of existing code base. Knitting existing codes together has not been very successful.
- For a multi-physics code, the problem state and the problem mesh should be central.
- Staff training and evolution in the large ASC code projects are significant problems. After a decade, many staff members have worked on only one or two major physics packages.
- With our physics focus, we have inadvertently “discriminated” against software professionals.
- Only a few people at the Labs are capable of running the true capability simulations. Even fewer are capable of analyzing the large data sets that result.
Predictive Capability Is Multi-faceted

- Predictive capability is important, particularly for complex problems when one is going outside the phase space of established experience.
- But, codes that accurately interpolate within the phase space of established experience can be very important.
- In today’s world, it’s an iterative trio of
  - Theory
  - Experiment
  - Simulation
- Iteration with experiment is always important (validation)
- Often though, we’re computing for insight
  - Validation doesn’t mean agreement within 3%

What do we need?

- Conversion tools to migrate/merge mixed language code bases (to C++?)
- Tools to migrate the existing code bases to the emerging heterogeneous architectures
- Plotting and analysis tools need to improve
  - How do we effectively visualize, analyze 3D data?
  - The data sets are becoming much larger.

- Be careful about what you incentivize. The ASC cost model is irrational (in Alme’s opinion) for the final users, the designers. The designers do not see any impact of the actual cost of the simulations run. There was pressure to reduce the code run times until the run times fit the designer work patterns. (overnight runs, etc.) With a few man years of code optimization, we could have reduced current run times several-fold, but there was nothing in the system to make this a priority.
Elmer Business Model

Dr. Thomas Zwinger
CSC – IT Center for Science Ltd.
Espoo, Finland

Contents

• License and IP
• User Community
• Elmer services
• Funding/Maintenance schemes
License and IP

• **GPL 2** (or later)
  – Internal R&D and consultancy, no problem
  – GPL “viral effect”: derived versions also GPL

• **CSC owns the code**:
  – Commercial licensing allowing third party non-GPL software development
  – Due to Qt library ElmerGUI may only be licensed under GPL

License and IP cntd.

• **CSC has also withheld some software modules related to joint projects**
  – Parts may be added to GPL once the novelty value is used if agreed by all parties

• **Maintenance of full ownership of code**
  – External contribution: Equal ownership
  – Contributor Agreement → access to SVN
  – Flexibility for CSC in future/double licensing
User Community

Strategic alliances:
Nokia, Okmetic,…

EU level users
environmental sciences

Open source users

User Community cntd.

Use the force!
- elmerfem.org
  * Wiki, Forum
- www.csc.fi/elmer
  * Main Elmer page
- sourceforge.net
  * Main distribution channel
- www.nic.funet.fi
  * Manuals and binaries
Elmer services
(aka. Business models)

- **Daily support**: forums, web, emails, ...
  → *street credibility*
- **CSC’s mission**: computational science
- **Collaboration**: give and get
- **Consulting**: consulting agreements
- **Courses**: tailored courses
- **HPC**: Combining CSC’s computational capacities with Elmer

---

**Funding/Maintenance**

- **Unit price / mm**
  - Consultancy aka Engineering Office
  - Private collaboration, strategic
  - Tekes-project
  - EU-project
  - SA-project (national)
  - Courses
  - Academic aid, basic dev.

- **Cumulative extent / mm**
  - Private money
  - Public funding
  - IP & PR ?
Platform SALOME: maintenance, upgrade and users

Dr. André Ribes – EDF R&D – SALOME project
02 June 2010

The SALOME Platform (EDF and CEA project)

- SALOME is an open-source (LGPL) pre/post processing & solver coupling integration platform for the numerical simulation
  - Gather improvements for pre and post processing in a common platform for numerical simulation
  - Facilitate the integration of solvers coupling in a distributed heterogeneous environment
  - Distributed at http://www.salome-platform.org

The SALOME Platform (EDF and CEA project)

- SALOME is an open-source (LGPL) pre/post processing & solver coupling integration platform for the numerical simulation
  - Gather improvements for pre and post processing in a common platform for numerical simulation
  - Facilitate the integration of solvers coupling in a distributed heterogeneous environment
  - Distributed at http://www.salome-platform.org
A specialization of SALOME: SALOME-Meca platform (Structural Mechanics)

Geometry + Meshing → Data setting → Code_Aster® computation → Post processing

Mechanical computation on a primary circuit coolant pump of nuclear plant

Distributed in open-source at http://www.code-aster.org

Different types of users

- SALOME faces many different types of users:
  - “Simple” users of generic modules: geometry, meshing, ...
  - Integrator users:
    - Want to create their own specialization platform on top of SALOME
    - Want to use some modules connected to their code: data assimilation as example
  - Coupling users:
    - Want to use SALOME platform module for code coupling application

- Each user wants to be able to:
  - Learn how to use the platform from his point of view
  - Tell us about bugs and ease of use
  - Talk about improvements
  - Contribute? No time for this 😞!
Tools to exchange with users

- Bugtracker
- Hotline
- Forum
- SALOME’s days and meetings
- Newsletters
- Trainings

Now what organization?
- Day by day
- Month by month

Month by month

SALOME’s days
- Each year: user day
  - New projects on SALOME
  - New usage or features on SALOME
  - And sure users talk about their problems
- Each new stable version: feature day
  - New features on SALOME
  - More technical than user day

Trainings
- Basic: presentation of all modules
- Advanced: geometry and meshing
- Visualization
- Code coupling and programming model

Each SALOME developer is involved in one or two integration projects
What size of project this organization manage ?

- **SALOME platform:**
  - 250 Users (at EDF and CEA)
  - 25 000 downloads / year ([http://www.salome-platform.org](http://www.salome-platform.org))
  - 200 engineers.years
  - 3 000 tests
  - 500 corrections and evolutions / year

- **Available physics platforms:**
  - SALOME-MECA ([http://www.code-aster.org](http://www.code-aster.org))
  - Structural Mechanics
  - PLEIADES (AREVA, CEA and EDF)
  - Nuclear fuel
  - ALLIANCES (ANDRA, CEA)
  - Nuclear waste storage
  - PANTHERE V2 (EDF)
  - Radiation protection
  - NURISP (EUROPEAN UNION) ([http://www.nuresim.com](http://www.nuresim.com))
  - Nuclear reactor simulation
  - …
What to learn?

- Different types of users means different needs
  - From geometry module usage to parallel code coupling integration...

- A lot of tools are needed to communicate with user community
  - A lot of energy and effort day after day

- Developers have to be in contact with users to respond to their needs
  - To see real problems and talk with real users

- It’s difficult to avoid a “heavy” organization
  - Multi-level organization
Scalable Engineering Software Workshop
May 2-3, 2010

Gene Poole

Outline

- Practices
- Priorities
- Problems
Practices

- Code maintenance and QA are closely coupled
- QA is driven by customer demand
  - Customers don't care about maintenance
  - Customers DO care about bugs, reliability and useability
- Legacy code is maintained and incrementally changed
  - Here QA maintains a “status quo”
  - Experienced user base demands continuity and reliability
- Modern software is adaptable and can change radically
  - Here QA is part of a continual feedback loop
  - Users demand the ability to interact with the software

Practices at CD-adapco

- Maintenance tasks are part of each design process
- Code cleanup is ongoing and enforced by higher level than original developer
  - Code may become locally cluttered when features or new algorithms are added but is cleaned as soon as new becomes permanent
- Short release cycle supports regular maintenance
  - New features are incremental to overall design
  - Maintenance is integral to the process
Priorities

- Code maintenance is not the main priority
  - Features, performance, reliability are rewarded
  - Maintenance is (an important) means to the end
- Legacy Code
  - Maintenance mostly maintains capability
  - Major change is risky and difficult
  - Focus is to support existing users and solve outstanding issues that limit the code performance or competitiveness
- Modern architecture (modular, object oriented)
  - Maintenance can be ongoing and localized to meet specific needs
  - Major features or changes can be made within the design framework quickly without starting over

Problems

- Tyranny of the release cycle
- New software complexity adds to the challenge
  - We don’t write all of the software
  - We can’t always predict what users will try with the flexible, adaptable interfaces we give them
- Parallel processing
  - Every new computer is a parallel computer
  - How do we maintain robust QA testing for an unknown/ever increasing number of processors?
- QA/maintenance for modern software needs a different focus than the old status quo system of the past.
  - Old system – 1000’s of short, accumulated test jobs
  - New system – thoroughly test all paths on carefully designed test cases
What Have We Learned About Successful Scalable (Engineering) Software?

David E. Bernholdt
Computer Science and Mathematics Division
Oak Ridge National Laboratory

Ground Rules

• What follows is my distillation of the discussions yesterday
• Please correct errors of fact!
• Feel free to disagree with things I say
• Please add your own observations
• Ample time for discussion
  – I hope someone is taking notes!
Outline

• Diverse Origins
• Scalability
• Verification and Validation
• Funding Models
• Relationship Between Government and Industry
• Intellectual Property
• Code Management and Maintenance
• User Community

Diverse Origins

• Most of the codes were created to address a scientific or engineering need
  – Mostly “purpose built” (maybe using existing components)
  – Quantum ESPRESSO unified independent research efforts of long standing
• Size of code teams ranges from 2 - ~dozens
• Originated in garages, academia, government labs, industrial labs, ISVs
Scalability

- Only NWChem and OpenFOAM expressed scalability as a primary goal
  - NWChem: scalable in both computer hardware and chemical system

- Mostly to thousands of processors
  - Some in the hundreds (some problems harder)
  - NWChem tens of thousands
  - Actual scalability often leads user needs

- No algorithmic magic

- Reliance on external packages for key computational needs (i.e. solvers)
  - OpenFOAM owns everything itself

- I/O most cited scaling bottleneck

Verification & Validation

- Yes, we do it (or at least aspire to do it)
  - Experimental data, simplified/idealized test problems, comparison with other codes
  - Thousands of tests, run nightly or pre-release
  - Many codes lack comprehensive suite of tests (unit, integration, regression, validation)

- V&V is (part of) what you pay for in commercial codes
  - Lower expectations for freely available codes
  - Users are good at finding and reporting bugs
  - Reproducibility is also important (the same wrong answers)

- Combinatorial explosion of tests for multicore/parallel
  - Can’t test all processor counts, too many corner cases
  - Need to shift focus from discrete test cases to testing all paths through the code
Funding Models

- Diverse
- Much direct govt funding
- Private funding for support, customization, consulting
- Institutional support
  - Provide key positions
- Funding for scientific research vs funding for software as a product?

Relationship between Industry and Government

- Different approaches to government support of industry in Europe and US
- Europe...
  - Fuzzier lines between public and private entities
  - Direct govt support of industrial software dev
  - Expectation that public funding is "seed" to benefit industry, get more private funding
- US...
  - Little direct govt support for industrial software dev
  - Tech transfer challenges from govt labs
  - Govt labs not allowed to compete with US industry
  - Cooperative agreements hard to negotiate
Intellectual Property

- Open source model widely used
  - EDF: we do software, but we’re not a software company
  - Acknowledged issues with US export control, national security constraints
  - GPL most commonly mentioned license
  - Make money from support, consulting
- Issues of how to measure usage/impact in an open/free environment
  - Cited as a reason for non-open licensing in some cases
  - Downloads, mailing list membership as proxies for user counts
- “Street credibility” with users, facilitated by easy access to software, seen as valuable

Code Management & Maintenance

- Use of “gatekeeper” to control code base
- Use of “change control board” to prioritize and plan developments
  - Formal or informal
  - Internal or may include stakeholder reps
- Rule of thumb: 10 years to take code/capability from conception to production
- Release cycles typically 3-4/yr
- Binary code distribution easiest, but often impractical
User Community

- User connection is central to overall approach
  - Drive requirements
  - Help sell sponsors on new funding

- Users don’t contribute code
  - Users are mostly *not* developers (different from many science codes)
  - Do find and report bugs

- Multiple levels of engagement
  - Online communication tools
  - User group meetings, trainings, …
  - Embedding/consulting with users

- Acceptance of modeling and simulation by engineers?
  - If culture/background is experimental, M&S may be a hard sell
  - Goodyear, DoD CREATE, nuclear engineering, …

What Did I Miss?
**Education and Workforce Development**

- Strong culture in Europe of industrial support for academic research, PhD students
  - Direct industrial support of (academic) research far more common
  - Students often sited at industrial sponsor location
- Workforce development
  - EDF example of encouraging use of SALOME in college courses
  - Industry recruits from students they support

---

**Competitive Advantage, Information Assurance**

- Concerns over competitive advantage manifest differently in different organizations
  - Academia wants to publish
  - Industry wants unique products to sell
  - Defense seeks advantage over other nations
- Often methods are well known, instead, it is a particular workflow/composition, or results that are considered “proprietary”
  - Makes open distribution of code easier
  - Protection of (industrial) data is expected
Computational Nanoscience and Results from 3-Year Study on Simulation-Based Engineering and Science

Peter T. Cummings

Department of Chemical and Biomolecular Engineering
Vanderbilt University
and
Center for Nanophase Materials Sciences
Oak Ridge National Laboratory

International Workshop on Scalable Engineering Software
National Science Foundation, Arlington VA
June 2-3, 2010

Introduction

- Theory, modeling and simulation (TMS)
  - Expected to play key role in nanoscale science and technology
    - "Nanotechnology Research Directions: IWGN Workshop Report, Vision for Nanotechnology Research and Development in the Next Decade," edited by M.C. Roca, S. Williams, P. Alivisatos, Kluwer Academic Publisher, 2000
      - Also available on-line at http://www.wtec.org/layol/nanos/
      - IWGN Research Directions
      - Discusses issues and examples
      - Published by DOE
      - Also available on the web at http://www.er.doe.gov/bes/reports/files/TMN_rpt.pdf
      - Published by DOE and NNI
      - Also available on the web at http://www.er.doe.gov/bes/reports/files/NREN_rpt.pdf
    - BESAC, "Opportunities for Discovery: Theory and Computation in Basic Energy"
Appendix C. Presentations

Computational Nanoscience

- Background:
  - Founder, Nanomaterials Theory Institute (NTI), Center for Nanophase Materials Sciences (CNMS) at Oak Ridge National Laboratory (ORNL)
  - Theory group for CNMS
    - Joint with Computer Science and Mathematics Division
  - Thomas Schulthess-led NTI won 2008 and 2009 Gordon Bell prizes for performance
  - CNMS is user facility (~50%) and research center (~50%)
    - Entered full-scale operation in 2005
    - ~400 users per year, ~200 projects, about 20% are theory/simulation
    - NTI is bridge between users and HPC/USC
    - Focus on codes that scale well on capacity and capability platforms
      - E.g., LAMMPS for molecular dynamics, Paul Kent's version of VASP for first principles calculations
      - Cluster, NERSC allocation, INCITE allocation

Computational Nanoscience

- Trends in software for computational nanoscience
  - Every computational subfield reaches point where need for control gives way to need for capability
    - Instead of every group having its own home-grown code, community-based codes emerge
      - E.g., LAMMPS/DL_POLY for molecular dynamics (MD), GAMESS/VASP/Siesta/INCHOM for first principles, etc
    - Focus changes from what can be done to solving problems as accurately as possible
      - E.g., for MD community, quality of forcefields and analysis of trajectories replaces time spent on extending in-house codes
    - Reducing codes to libraries that can be optimized on emerging platforms by computational science experts will allow computational nanoscientists to focus on coding for scientific insight, not computational efficiency
  - Keep in mind that today's scientific codes will form core of tomorrow's engineering software
    - Transition from nanoscience to nanotechnology
Appendix C. Presentations

Simulation-Based Engineering and Science

- **Components**
  - **International comparative study** [http://www.wtec.org/sbes](http://www.wtec.org/sbes)
    - Identify innovation overseas and mechanisms for achieving potential collaboration with U.S. R&D programs; clarify research opportunities and needs for promoting progress in field
  - **Schedule**
    - Kick off meeting: 10 July 2007
    - Baseline workshop: 1-2 November 2007
    - Visits to Asia: 3-7 December 2007
    - Visits to Europe: 25-29 February 2008
    - Final workshop: 25 April 2008
    - Final report released: 22 April, 2009
  - **Strategic directions report** [http://www.wtec.org/sbes-vision/](http://www.wtec.org/sbes-vision/)
    - Strategic investments in SBE&S are needed in order to achieve promise of SBE&S; time frame and format of these investments
    - **Schedule**
      - Strategic directions workshop: April 22-23, 2009 at National Academies
      - Report: Available today

---

Simula

- **Components**
  - **International**
    - Identify innovation and need
    - **Schedule**
      - Kick
      - Baseline
      - Event
      - Finale
  - **Strategic direction**
    - Strategic
    - SBE&S
    - **Schedule**
      - Start
      - Rep
Appendix C. Presentations

Simulation-Based Engineering and Science

- **Structure of International Comparative Study**
  - **Primary thematic areas**
    - Materials
    - Life sciences and medicine
    - Energy and sustainability
  - **Core cross-cutting issues**
    - Next-generation algorithms and high performance computing
    - Multiscale simulation
    - Simulation software
    - Validation, verification, and quantifying uncertainty
    - Engineering systems
    - Big data and data-driven simulations
    - Education and training
    - Funding, organization, and collaboration
Appendix C. Presentations

Findings from International Comparative Study: Materials

- Computational material science and engineering is changing how new materials are discovered, developed and applied
- World-class research in all of the above-mentioned areas in the U.S., Europe and Asia
  - However...
    - Algorithm innovation primarily in US and Europe, some in Japan
    - Almost none in China (primarily applications)
- Rapid ramping-up of activities in some countries
  - China
  - Extraordinary revival of science in Germany

Findings from International Comparative Study: Materials

- Much greater collaboration among groups in code development in Europe
  - US tenure process and academic rewards systems mitigate against collaboration
  - How much credit is given to junior collaborator on large code development?
- Development of simulation tools is not considered high-impact science
  - Numerous counter-examples
  - Impact often not clear for decade or more
Findings from International Comparative Study: Materials

- Much greater collaboration among groups and development in Europe
  - US tenure process and academic rewards stand against collaboration
    - How much credit is given to junior collaborators on code development?
- Development of simulation tools is not having much impact science
  - Numerous counter-examples
  - Impact often not clear for decade or more

Findings from International Comparative Study: Materials

- Quality of leading researchers in U.S. is comparable to that of leading researchers in Europe and Japan
- Funding models to support ambitious, visionary, long-term research projects is better in Europe and Asia than in the U.S.
  - Baseline direct funding
  - EU frameworks
  - In U.S., long-term support of any science is eroding***
    - NWChem was in construction budget of EMSL
    - NIRT projects last just long enough to get people working together
    - Commercialization is considered success
- Funding models and infrastructure to support multi-investigator collaborations between academia and industry
  - Have long tradition in Japan
  - Are rapidly developing in Europe
    - IP issues with U.S. universities

***BUT!
Appendix C. Presentations

Strategic Workshop and Report

- Research Directions: Vision for Research and Development in Simulation-Based Engineering and Science in the Next Decade
  - National Academy of Science Building, April 22-23, 2009
- Day 1: Framing talks by 21 experts on importance of SBE&S and potential for transforming US scientific and industrial competitiveness
  - Climate change, earthquake simulation/prediction, subsurface transport, industrial applications (Goodyear), chemical process industry, econometrics, bioinformatics, cancer modeling, software frameworks, materials, nanoscience, turbulent combustion, molecular theory/chemistry, high energy physics
- Day 2: Breakout sessions, drafting of recommendations in focus areas
  - Discovery, engineering innovation, software, big data, education

Recommendations from Strategic Workshop Report

- Recommendations relevant to software
  - Provide long-term (5+ years) single and small group grants
  - Create long-term team grants that support interdisciplinary collaborations among domain scientists, computational scientists, and mathematicians
  - Build large-scale virtual institutes/centers tasked with developing and stewarding community codes for specific SBE&S domains
  - Establish 20+ multi-investigator SBE&S interdisciplinary research institutes, with broad research programs in specific SBE&S problem domains, data and software
  - Establish 40+ multi-investigator SBE&S interdisciplinary research centers, with more focused research efforts in SBE&S problem domains, data and software
  - Award several hundred innovator grants to individuals or small teams conducting high-risk, high-reward transformative research in data and software
Scalable Numerical Engineering Software

Charbel Farhat
Stanford University

RESEARCH AND DEVELOPMENT

Research

Scalable Numerical Engineering Software

Development

Computer Science

Engineering Disciplines
SCALABILITY

* “The ability of the software to run efficiently on large numbers of parallel processors or clusters in order to study large or complicated problems”

* The ability of the software to run $n$-times faster on an $n$-times larger parallel machine

  \[ \text{speed-up} \begin{cases} \text{- conventional (fixed-size problem)} \\ \text{- scaled (scaled-size problem)} \end{cases} \]

  \[ \text{parallel scalability} \]

* The ability of the software to solve an $n$-times larger problem on an $n$-times larger parallel machine in the same amount of CPU time

  \[ \text{numerical scalability} \]

NUMERICAL SCALABILITY

* Computational complexity: $C n^\beta$

* Equation solvers (implicit computational technology)

  - direct methods (skyline, frontal, sparse, …) $\Rightarrow \beta > 2$
  - optimal iterative methods $\Rightarrow \beta \sim 1$

  \[ \text{robustness for complicated problems} \]

* Global operations
PARALLEL SCALABILITY

* A parallel-wise scalable but suboptimal algorithm is an algorithm which runs $n$-times less slowly on an $n$-times larger parallel machine

* Load balance

* Data locality at multiple levels

* Communication bandwidth → hardware → fabric

ENGINEERING VS SCIENCE

Science Engineering Science Engineering

knowledge-driven cost/time-driven

routine analysis design, control assistive technologies
Scalable Engineering Software Workshop: Current US Situation

June 3, 2010

Joseph Jung, Manager
Sierra/SolidMechanics-StructuralDynamics
Sandia National Laboratories

Panel 4 Opening Remarks

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

Current Situation Assessment

- Players
  - National Labs
  - Universities & Super Computing Centers
  - Commercial codes
  - Large Companies (special in-house apps)
  - DoD Labs

- Science Drivers
  - HPC software and hardware offers ‘unparalleled’ opportunity to address:
    - Multi-physics events
    - Multi-length scale understanding (space and time)
    - Intrinsic UQ

- Economic Drivers
  - Commercial HPC software will focus on moderately complex problems rather than high fidelity problems
    - For largest impact, most users want more runs quicker vs. few really high fidelity runs
  - Large cost associated with supporting scalability
    - Access to machines on a regular basis for testing
    - Debugging is more difficult and often takes much longer
    - New algorithm development

- People Drivers
  - People are still asked to solve real problems
    - How much headache can they stand?
    - Model setup becomes more difficult for complex problems
    - Results analysis (e.g. visualization becomes more complex)
Current Situation Assessment

- Development Models
  - Team development environments are essential
  - Need cross-functional teams with computer science, software engineering, and mechanics domain experts
- Testing, Testing, Testing
- Hardware & System Software Issues
  - Large machines are few and usually have enough uniqueness to them to make supporting all of them more time consuming
    - e.g. For SNL's new 18K core machine the vendor went from their standard 2D grid to a 3D torus and new interconnect cabling that caused months of support issues with the development teams to track down the source of the errors – hardware, system software (MPI), or the application code.
    - Variations in the feature set of light-weight kernals on the compute nodes
  - Memory access is big issue
  - I/O performance is big issue
  - Lack good tools for developers
    - Debugging at scale
    - Performance profilers
    - Testing at scale
  - Robustness of MPI implementation
    - Seems like the application development teams are expected to do all the testing for new versions
  - Usually when a system software stack (compilers, MPI, etc) becomes stable, there is a desire to freeze it and not make future updates (due to the large support costs to retest everything) which can cause compatibility issues with the application software as system software stacks continue to evolve on commodity HPC platforms.

What Advances are Needed

- We need to look at the whole user Software Stack
  - Problem setup, prep, and execution
    - Still very user time intensive
    - Problem geometry needs to be on solver machine
    - Meshing needs to be done in parallel on solver machine
    - Initial domain decomposition needs to be done in parallel on solver machine
    - Intrinsic UQ
  - Post simulation analysis
    - Output and post processing very time consuming
    - Remote Viz is tough
- Robustness of Hardware and Software as a system
  - Fault tolerance HW/SW
  - Error handling
  - Machine availability for “at scale” testing
- Algorithms Needs
  - AI for Automated output detection and load imbalance
  - Parallel in Time
  - Multi-scale in time/space
  - Any O(ln #Processor) algorithms
    - Solvers
    - Search algorithms
  - Communication avoiding algorithms
Scalable Application Development

Laxmikant (Sanjay) Kale

http://charm.cs.uiuc.edu

Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana Champaign

Why am I here: my story

• Computer Science Professor, did parallel prolog and AI until tenure, oriented to practical parallel apps around 1992

• Guiding principles:
  – No magic
    • i.e. no Parallelizing compilers
  – Seek an optimal division of labor between the system and the programmer
  – Design abstractions based solidly on use-cases
  – Application-oriented yet computer-science centered approach

NAMD: A Production MD program

**NAMD**
- Fully featured program
- NIH-funded development
- Distributed free of charge (~37,000 registered users)
- Binaries and source code
- Installed at NSF SC centers, desktops, …
- User training and support
- Large published simulations

My collaboration started in 1992, 1994: v1 of namd
1996 current NAMD was born, from scratch design
Parallelization using Charm++

Initial design based on asymptotic iso-efficiency analysis, has withstood the test of time.


<table>
<thead>
<tr>
<th>Language</th>
<th>files</th>
<th>blank</th>
<th>comment</th>
<th>code</th>
<th>scale</th>
<th>3rd gen. equiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>169</td>
<td>13706</td>
<td>14253</td>
<td>71335 x</td>
<td>1.51 =</td>
<td>107715.85</td>
</tr>
<tr>
<td>C</td>
<td>90</td>
<td>5945</td>
<td>8909</td>
<td>25832 x</td>
<td>0.77 =</td>
<td>19890.64</td>
</tr>
<tr>
<td>C/C++ Header</td>
<td>249</td>
<td>6363</td>
<td>7431</td>
<td>24026 x</td>
<td>1.00 =</td>
<td>24026.00</td>
</tr>
<tr>
<td>Tcl/Tk</td>
<td>36</td>
<td>857</td>
<td>1034</td>
<td>3183 x</td>
<td>4.00 =</td>
<td>12732.00</td>
</tr>
<tr>
<td>make</td>
<td>6</td>
<td>227</td>
<td>536</td>
<td>863 x</td>
<td>2.50 =</td>
<td>2157.50</td>
</tr>
<tr>
<td>C Shell</td>
<td>3</td>
<td>62</td>
<td>2</td>
<td>447 x</td>
<td>3.81 =</td>
<td>1703.07</td>
</tr>
<tr>
<td>Bourne Shell</td>
<td>2</td>
<td>59</td>
<td>118</td>
<td>143 x</td>
<td>3.81 =</td>
<td>544.83</td>
</tr>
<tr>
<td>Perl</td>
<td>1</td>
<td>13</td>
<td>13</td>
<td>42 x</td>
<td>4.00 =</td>
<td>168.00</td>
</tr>
</tbody>
</table>

SUM: 556 27232 32296 125871 x 1.34 = 168937.89
Performance of NAMD

![Graph showing performance of NAMD with various number of cores and time per step.]

OpenAtom
Car-Parinello ab initio MD
NSF ITR 2001-2007, IBM

Molecular Clusters:
- G. Martyna (IBM)
- M. Tuckerman (NYU)
- L. Kale (UIUC)

Nanowires:

Semiconductor Surfaces:

3D-Solids/Liquids:
New OpenAtom Collaboration (DOE)

- Principle Investigators
  - G. Martyna (IBM TJ Watson)
  - M. Tuckerman (NYU)
  - L.V. Kale (UIUC)
  - K. Schulten (UIUC)
  - J. Dongarra (UTK/DOE)

- Current effort focus
  - QMMM via integration with NAMD2
  - ORNL Cray XT4 Jaguar (31,328 cores)

- A unique parallel decomposition of the Car-Parinello method.
- Using Charm++ virtualization, we can efficiently scale small (32 molecule) systems to thousands of processors.
OpenAtom code-base

8+ years of development as a charm project (without prior piny efforts)
12 contributors in last 5 years
160 commits in last 150 days (2010 Jan-May)

500 source code files
190K lines in project
95K Sloc (pure source. no comments, blank lines etc)

PARALLEL code characteristics:
240 entry methods
30 parallel compute entities (chare arrays and groups. excluding map groups)
50 message types
20 modules (in ci files. logically, there are more)

Appendix C. Presentations

Scaling Water on Blue Gene/L
ChaNGa

- Collaborative project (NSF, NASA)
  - with Tom Quinn, Univ. of Washington
- NSF ITR, 2002-2008, on life-support via NASA now
- Gravity, gas dynamics
- Barnes-Hut tree codes
  - Oct tree is natural decomp
  - Geometry has better aspect ratios,
  - But is not used because it leads to bad load balance
  - Assumption: one-to-one map between sub-trees and PEs
  - Binary trees are considered better load balanced

With Charm++: Use Oct-Tree, and let Charm++ map subtrees to processors

ChaNGa

- 8 years of development (not counting imported code from PkdGrav and others)
  - Somewhat interrupted mode for the past 2-3 years.. After NSF ITR funding stopped
- Code: (Charm++, C++)
  - 120 source files
  - 55k lines (total in files)
  - 40k SLOC
Charm++ and Adaptive MPI

- Development since 1988 (may be 1992)
- Almost never directly funded
  - (1988-1990 NSF was the exception)
- Slow evolution
  - It's like a piece of art that we are sculpting in our basement slowly
  - Except, its engineering piece in use by parallel apps all along
    - They provided most of the funding
Charm++ 6.2 Release and LOC

<table>
<thead>
<tr>
<th>Language</th>
<th>files</th>
<th>blank</th>
<th>comment</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>686</td>
<td>35131</td>
<td></td>
<td>186972</td>
</tr>
<tr>
<td>C</td>
<td>782</td>
<td>28796</td>
<td></td>
<td>144946</td>
</tr>
<tr>
<td>C/C++ Header</td>
<td>908</td>
<td>19328</td>
<td></td>
<td>72169</td>
</tr>
<tr>
<td>Bourne Again</td>
<td>251</td>
<td>2603</td>
<td>2895</td>
<td>19965</td>
</tr>
<tr>
<td>make</td>
<td>266</td>
<td>3038</td>
<td>878</td>
<td>7041</td>
</tr>
<tr>
<td>Fortran 77</td>
<td>27</td>
<td>697</td>
<td>799</td>
<td>3385</td>
</tr>
<tr>
<td>Perl</td>
<td>8</td>
<td>273</td>
<td>240</td>
<td>1848</td>
</tr>
<tr>
<td>Assembly</td>
<td>20</td>
<td>357</td>
<td>698</td>
<td>1817</td>
</tr>
<tr>
<td>Fortran 90</td>
<td>15</td>
<td>184</td>
<td>44</td>
<td>1426</td>
</tr>
<tr>
<td>Java</td>
<td>17</td>
<td>765</td>
<td>172</td>
<td>1300</td>
</tr>
<tr>
<td>XML</td>
<td>1</td>
<td>18</td>
<td>5</td>
<td>117</td>
</tr>
<tr>
<td>Bourne Again Shell</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>83</td>
</tr>
<tr>
<td>C Shell</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>XML</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>CSS</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>44</td>
</tr>
<tr>
<td>Pascal</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>SUM</td>
<td>3029</td>
<td>91112</td>
<td>91808</td>
<td>444836</td>
</tr>
</tbody>
</table>

- Slowly evolved code
- Daily build and tests on dozen+ machine configurations
- Interconnects/ APIs (just those still autobuilt)
  - Ethernet
  - InfiniBand
  - Myrinet
  - Cray SeaStar
  - Blue Gene
  - LAPI
  - VMI
  - SMP, PXSHM, SYSVSHM
- Compilers
  - GCC
  - ICC
  - Sun CC
  - PGI

4055 text files.
3854 unique files.
913 files ignored.

5452 Gengbin Zheng
1559 Orion Lawlor
1035 Milind Bhandarkar

Log-tail distribution of contributions (showing Number of checkins:
May be distorted, because pre-1995 numbers unavailable, and also, doesn't track the magnitude of change checked in. So, don't take the numerical data too seriously, but the log-tail nature of it)

<table>
<thead>
<tr>
<th>Contributors</th>
<th>Number of Checkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lukasz Wesolowski</td>
<td>9</td>
</tr>
<tr>
<td>Gunavardhan Kakulapati</td>
<td>9</td>
</tr>
<tr>
<td>Parthasarathy Ramachandran</td>
<td>9</td>
</tr>
<tr>
<td>Sameer Paranjpye</td>
<td>8</td>
</tr>
<tr>
<td>Robert Blaize</td>
<td>12</td>
</tr>
<tr>
<td>Yogeesh Mahta</td>
<td>12</td>
</tr>
<tr>
<td>Jonathan Lefter</td>
<td>2</td>
</tr>
<tr>
<td>Filip Giaochoin</td>
<td>7</td>
</tr>
<tr>
<td>Parthasarathy Ramachandran</td>
<td>7</td>
</tr>
<tr>
<td>Sameer Paranjpye</td>
<td>7</td>
</tr>
<tr>
<td>Robert Blaize</td>
<td>7</td>
</tr>
<tr>
<td>Yogeesh Mahta</td>
<td>7</td>
</tr>
<tr>
<td>Jonathan Lefter</td>
<td>7</td>
</tr>
<tr>
<td>Nickel Lawlor</td>
<td>7</td>
</tr>
<tr>
<td>Sameer Paranjpye</td>
<td>7</td>
</tr>
<tr>
<td>Robert Blaize</td>
<td>7</td>
</tr>
<tr>
<td>Yogeesh Mahta</td>
<td>7</td>
</tr>
<tr>
<td>Jonathan Lefter</td>
<td>7</td>
</tr>
</tbody>
</table>

Log-tail distribution of contributions (showing Number of checkins:
May be distorted, because pre-1995 numbers unavailable, and also, doesn't track the magnitude of change checked in. So, don't take the numerical data too seriously, but the log-tail nature of it)
Collaboration Models

- A central (bottom?) team of people with parallel programming expertise
- Collaborating with multiple application teams
  - Each application team has at least one full-time member of the CS team, and fractions of other people’s time
- Allows us to pool expertise
- Transfer useful insights, tools, libraries across apps

Interdisciplinary communication

- Need to develop mutual respect and understanding
- Communication:
  - Scientist: Something-Sham psuedo-potential
  - CS person: you mean array A
- Takes time to develop a mutual vocabulary
Parallel Code development

• Sorry, no “parallelize this” flag that works
• Must design the parallel structure from scratch
  – BUT: you can import code from previous generation products
    • PINY → OpenAtom, PKDGrav → ChaNGa
  – Just that the top level has to be from-scratch parallel design
  – Some shoehorning needed (its called “refactoring”)

Obstacles

• Transient population of personnel
  – Students
  – Staff
  – Academic salaries
• Steep learning curves
  – MS students pose a challenge
  – Progress reports become an art
    • (can’t say “student spent the semester trying to learn X, before he could contribute to the project”)• Community attitudes: innate conservatism, “platonic” CS
Role of infrastructure and tools

• Charm++, with its adaptive runtime, helped scaling, and provided a medium for tech transfer across projects
• Targeted development of modules
• Targeted development of performance analysis tools (*Projections, CharmDebug*):
  – Produce a view/tool as you see the need
  – Evaluate across apps
Licensing

• Charm++ and NAMD have been distributed, since 1992 or so, with a license:
  – Freely available in source-code form, for non-commercial use
    • Commercial use: requires negotiated license
  – I found out that this is not “Open Source”
  – Open source has been designed for “free-rider” commercial exploitation
    • Except for GPL, but it has other problems
  – I can elaborate, but please consider alternative models
    • E.g. what about charging a fee for usage to fund support, for academic sw? we don’t do this, but some others do, and it should be respected
  – Collaborators can contribute modules in any license they want
• OpenAtom and ChaNGa are traditional Open-source

Funding Models

• Recognize the importance of software as a thing-by-itself (rather than ancillary to science)
  – Community software
• NIH “resource” as a model
  – NSF’s new SSI-SI2 program is welcome step, and DOE has recognized this in many ways
• Community software needs to be nurtured
  – Needs evaluation but needs long-term funding
• Consider “life-support” grants
  – 1 grad student to keep it (sort of) going, during fallow periods of no-big grant
Summary

• Code development models:
  – From scratch parallel algorithm, with importing

• Interdisciplinary collaboration
  – Is challenging but needed
    • The model that worked for us: a team of parallelizing experts working with multiple apps

• Recognize diversity of licensing models

• Need a funding model that can nurture software over a decade or two
Lessons for creating and maintaining scalable engineering codes
Douglas B. Kothe
National Center for Computational Sciences
Oak Ridge National Laboratory

Questions Considered...

- Characterization of code scalability: measures could include benchmark runs on specific systems, graphs showing scalability as number of processors increases, discussion of whether the code displays weak or strong scalability up to a certain number of processors, sustained rate in TFLOPs, or other.
- What technical advances are required in order to achieve good scalability? These could include better physical models, better algorithms, better computer science, etc.
- What is the business model in developing scalable code? Business models include factors such as funding model for code development (private funding, government funding, volunteer effort), organizational model (private company, university, government center, volunteer community, individual), leadership model (charismatic leader, appointed leader, steering committee, etc.), intellectual property model (proprietary, open source, public domain, informal), sustainment model (license fees, contributions, sustained government funding, sustained private funding, embedding in an ongoing research group).
- Measures of acceptance and adoption of scalable code within the engineering community: these could include discussion of reference accounts, estimates of number of users, engineering areas where the code is most used, or other.
Random Observations (with some liberties taken)

- Usage model for engineering codes: parallel or not?
- Workflow is a big deal
- Users are hungry for bigger, better, faster (downloads)
- Software infrastructure is important (CIT with FEM)
- Grids are obstacles (parallel meshing, lat-long)
- Take some requirements with a grain of salt (turn-around time – make sure your metrics are what you want)
- Open source is alive and well and thriving
  - Proc: widespread adoption, standardization, feature evolution, defect reduction, collaboration/training/funding mechanism
  - Code: IP ownership, copyright issues murky at best; needs evolution/refinement – currently like Linux in the 1990s
- Community libraries (e.g., linear solvers) are not the norm
  - And why not? Emerging hardware is enough of a motivator, but there are many more

Random Observations (with some liberties taken)

- Feature-specific algorithms (e.g., surface models in a volumetric domain) can be performance roadblocks
- Parallelism along the time dimension (e.g. parallel) is not needed (or is it?)
- This does not appear to be a recognized domain-specific middleware layer
  - Not “just” linear and nonlinear solvers: let’s go further: Particle pushers, MD kernels, ...
- Hybrid computing and accelerators: “embrace the horror”: is it that hard? Depends, but let’s quit complaining and waiting and get busy.
  - Accelerators (e.g. GPUs) are game changers: laptops in a few years bigger than “ASCI White”
- Export control issues associated with S/W source and/or usage remain
- Lots of code and time spent on testing. How much is “enough”?
- Regular release cycles common and good (evolutionary delivery life cycle)
- Funding model for development: still “heroes in the attic” or an “add-on” to something more applied?
- Analytics who know how to run big problems and analyze the data are needed and valued
Random Observations
(with some liberties taken)

- Many problems and challenges are not technical (instead more managerial and programmatic)
- Large research entities need to communicate, collaborate, and coordinate more (Ex: DOE, DoD, NASA, NOAA, NIH, NSF, ...)
- Create and maintain a “fierce sense of urgency”
- Scaling and performance analysis/optimization for the sake of scaling and performance analysis/optimization (no user or stakeholder asking for it) always pays off
  - Ex: the Joule metric, climate, ...
- The “give and get” collaboration mantra works and makes sense
- Traditional incentives and recognition (from stakeholders and peers) for code development successes lacking (“science citation” or “h-index” for code downloads, usage, results, data generation?)
- Software development best practices are fairly well socialized and accepted, but
  - Lots of things are still “hard”: build and test system, requirements management, support, high-risk, high-reward endeavors, etc.

Random Observations
(with some liberties taken)

- US Industry is very hungry for large, scalable engineering applications and results
  - Ex: GM, GE, P&W, P&G, BMI, Boeing, Goodyear, Whirlpool, UTC, Ramgen, ...
Engineering Code Projects
Project Management Lessons Learned

- The Project Leader can actually be a hindrance rather than a help; he/she should not be on the critical path
- Milestones can be good
- Documentation is not easy or fun, but it's necessary. You may not find a great product that allows multiple authors simultaneously working
- If you have never done this before, consider the "pi factor" on estimates of time/resources
- Embrace process but not without scrutiny and thought. It must be tailored for your project
- High risk, exploratory research can co-exist within an applied software project. But nourish it
- Record software statistics so informed estimations can be made for time/resources required for future activities

Engineering Code Projects
Project Management Lessons Learned

- Establish requirements first
- Have a lasting lead
- Adopt "alpha users" as part of your project
- Invite and encourage formal peer reviews
- Do not grow too big and too soon (big projects reach a point of diminishing returns)
- Do not over-promise. Set expectations early and often to clients and customers.
- Software culture change is possible – it must be top down
- Seek help from any and every body. But be careful who you bring on board formally
- Have a good project charter that allows you to say NO – scope creep is just too irresistible in science
Engineering Code Projects
Software Engineering Lessons Learned

- Design your software for and implement unit testing
  - Can you “make test” in every dir? Ex: buggy pageant
- Be aware of the impact of your choice of data structures – you have to live with it
- Use levelized design (defined interfaces, data hiding) as a mantra
- All your tests should be tied to requirements
- Conduct code and design reviews!
- Have formal releases early and often
- Do not under estimate the large difference between research prototype and production. Resist the urge to view your prototype as production. Embrace throwing away prototypes.
- Pair programming is good thing and really works.
- Assessments are a big pain, but they can help to force culture change (for better or worse)

Engineering Code Projects
Software Engineering Lessons Learned

- Get on as many platforms as possible – it makes your software better
- It doesn’t work to worry about performance later
- Unified Build Theory can consume all people cycles; similarly for a testing harness
- Training new team members takes time! Must plan for and institutionalize this
- Frameworks (backplanes, environments, etc.) should evolve
- If you don’t test it, it will break (“bit rot”)
- Even for large projects, really only see 3-5 active “committers” to the repository
- Don’t just design tests based on physics/algorithms; think about all use cases!
- Establish and document your software process
- Commit hot spots are usually symptomatic of a bigger problem
- Spending most of your time on testing is not a bad thing
- Writing code is very personal: allow personalities to flourish; don’t be too rigid
Appendix C. Presentations

We have increased system performance by 1,000 times since 2004

- NNSA and DoD have funded much of the basic system architecture research
- Cray XT based on Sandia Red Storm
- IBM BG designed with Livermore
- Cray X1 designed in collaboration with DoD

- DOE SciDAC and NSF PetaApps programs are funding scalable application work, advancing many apps
- DOE-SC and NSF have funded much of the library and applied math as well as tools
- Computational Liaisons key to using deployed systems

<table>
<thead>
<tr>
<th>Year</th>
<th>System</th>
<th>TF</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Cray X1 3 TF</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>Cray XT3 Single-core 26 TF</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Cray XT3 Dual-Core 54 TF</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Cray XT4 118 TF</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>Cray XT4 Quad-Core 283 TF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cray XT5 Systems 12-core, dual-socket SMP 2000+ TF and 1000 TF</td>
<td></td>
</tr>
</tbody>
</table>

Our Science requires that we advance computational capability 1000x over the next decade

- Deliver transforming discoveries in climate, materials, biology, energy technologies, etc.
- Ability to investigate otherwise inaccessible systems, from regional climate impacts to energy grid dynamics

- Providing world-class computational resources and specialized services for the most computationally intensive problems
- Providing stable hardware/software path of increasing scale to maximize productive application development

<table>
<thead>
<tr>
<th>Year</th>
<th>System</th>
<th>TF</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Cray XT5 2+ PF Leadership system for science</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>OLCF-3: 10-20 PF Leadership system with some HPCS technology</td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td>OLCF-4: 100-250 PF based on DARPA HPCS technology</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>OLCF-5: 1 EF</td>
<td></td>
</tr>
</tbody>
</table>
Future large-scale systems present challenges for applications

- Dramatic increases in node parallelism
  - 10 to 100X by 2015
  - 100 to 1000X by 2018
- Increase in system size contributes to lower mean time to interrupt (MTTI)
- Dealing with multiple additional levels of memory hierarchy
  - Algorithms and implementations that prioritize data movement over compute cycles
- Expressing this parallelism and data movement in applications
  - Programming models and tools are currently immature and in a state of flux

Exascale Initiative Steering Committee

Future large-scale systems present challenges for applications

- Intel 48-core experimental chip shipping this summer
- NVIDIA 512-core Fermi GPU

Within 5 years, these challenges will become increasingly significant at the desktop level
### Jaguar PF: World’s most powerful computer—Designed for science from the ground up

Based on the Sandia & Cray designed Red Storm System

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak performance</td>
<td>2,332 PF</td>
</tr>
<tr>
<td>System memory</td>
<td>300 TB</td>
</tr>
<tr>
<td>Disk space</td>
<td>10 PB</td>
</tr>
<tr>
<td>Disk bandwidth</td>
<td>246+ GB/s</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>18,088</td>
</tr>
<tr>
<td>AMD &quot;Istanbul&quot; Sockets</td>
<td>37,376</td>
</tr>
<tr>
<td>Size</td>
<td>5,000 feet²</td>
</tr>
<tr>
<td>Cabinets</td>
<td>200 (8 rows of 25 cabinets)</td>
</tr>
</tbody>
</table>

### Impressive Sustained Application Performance

<table>
<thead>
<tr>
<th>Science Area</th>
<th>Code</th>
<th>Contact</th>
<th>Cores</th>
<th>Total Performance</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Materials</td>
<td>DCA++</td>
<td>Schulthess</td>
<td>213,120</td>
<td>1.9 PF*</td>
<td>2008 Gordon Bell Winner</td>
</tr>
<tr>
<td>Materials</td>
<td>WL-LSMS</td>
<td>Eisenbach</td>
<td>223,232</td>
<td>1.8 PF</td>
<td>2009 Gordon Bell Winner</td>
</tr>
<tr>
<td>Chemistry</td>
<td>NWChem</td>
<td>Apra</td>
<td>224,196</td>
<td>1.4 PF</td>
<td>2006 Gordon Bell Finalist</td>
</tr>
<tr>
<td>Nano Materials</td>
<td>OMEN</td>
<td>Klimmek</td>
<td>222,720</td>
<td>860 TF</td>
<td></td>
</tr>
<tr>
<td>Seismology</td>
<td>SPECFEM3D</td>
<td>Carrington</td>
<td>149,704</td>
<td>165 TF</td>
<td>2008 Gordon Bell Finalist</td>
</tr>
<tr>
<td>Weather</td>
<td>WRF</td>
<td>Michalakes</td>
<td>150,000</td>
<td>50 TF</td>
<td></td>
</tr>
<tr>
<td>Combustion</td>
<td>S3D</td>
<td>Chen</td>
<td>144,000</td>
<td>83 TF</td>
<td></td>
</tr>
<tr>
<td>Fusion</td>
<td>GTC</td>
<td>PPPL</td>
<td>102,000</td>
<td>20 billion Particles / sec</td>
<td>2008 Gordon Bell Winner</td>
</tr>
<tr>
<td>Materials</td>
<td>LSF3D</td>
<td>Lin-Wang Wang</td>
<td>147,456</td>
<td>442 TF</td>
<td>2008 Gordon Bell Winner</td>
</tr>
<tr>
<td>Chemistry</td>
<td>MADNESS</td>
<td>Harrison</td>
<td>140,000</td>
<td>550+ TF</td>
<td>2008 Gordon Bell Winner</td>
</tr>
</tbody>
</table>
Appendix C: Presentations

**Jaguar Cray XT5 Nodes**

- Powerful node improves scalability
- Large shared memory
- OpenMP Support
- Low latency, high bandwidth interconnect
- Upgradable processor, memory, and interconnect

<table>
<thead>
<tr>
<th>GFLOPS</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory (GB)</td>
<td>16</td>
</tr>
<tr>
<td>Cores</td>
<td>12</td>
</tr>
<tr>
<td>SeaStar2+</td>
<td>1</td>
</tr>
</tbody>
</table>

**The pace for ultra-scale computing is fast and furious**

Measured Performance (teraflops):

- MADNESS: 850
- OMEN: 1042
- LANL HPL: 1759
- ORNL HPL: 1840
- LSMS-WL: 1960
- DCA++: 1832

Peak Performance (teraflops):

- MINERvA: 1003
- Cornell: 1029
- China TENSOR: 1026
- LANL RoadRunner: 1379
- ORNL Jaguar: 2332

Memory Size (terabytes):

- MINERvA: 54
- Cornell: 98
- China TENSOR: 99
- LANL RoadRunner: 133
- ORNL Jaguar: 364

*Top 3 US and Top 3 Foreign Systems*
# Appendix C. Presentations

## Jaguar XT5 Early Science Projects – Jul 2009

All projects successfully executed leadership-size simulations

<table>
<thead>
<tr>
<th>Science Domain</th>
<th>Project Description</th>
<th>Ages Right (cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Astrophysics</strong></td>
<td>A Three-Dimensional Model of SN1987A / Type Ia Supernovae</td>
<td>139,173</td>
</tr>
<tr>
<td></td>
<td>/                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /</td>
<td>36,000</td>
</tr>
<tr>
<td><strong>Biology</strong></td>
<td>Calculating the Size of A Colloidal Model: Understanding the Dynamics of Cell Membranes</td>
<td>398,479</td>
</tr>
<tr>
<td></td>
<td>/                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /</td>
<td>0</td>
</tr>
<tr>
<td><strong>Chemistry</strong></td>
<td>Quantum Monte Carlo Simulation of the Electronic Structure of a Molecule</td>
<td>391,173</td>
</tr>
<tr>
<td></td>
<td>/                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /</td>
<td>139,173</td>
</tr>
<tr>
<td></td>
<td>/                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /</td>
<td>139,173</td>
</tr>
<tr>
<td><strong>Materials Science</strong></td>
<td>Investigating the Hidden Model of Osmosis by Soft Matter Model</td>
<td>379,173</td>
</tr>
<tr>
<td><strong>Fusion</strong></td>
<td>Advancing Turbulent Transport Calculations</td>
<td>20,493</td>
</tr>
<tr>
<td></td>
<td>/                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /</td>
<td>379,173</td>
</tr>
<tr>
<td><strong>Geosciences</strong></td>
<td>Working on Reactive Plumes in Porous Media</td>
<td>373,000</td>
</tr>
<tr>
<td><strong>Clima</strong></td>
<td>Evaluations of High Impact Weather Events in an Oceanic Perspective</td>
<td>16,099</td>
</tr>
<tr>
<td></td>
<td>/                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /</td>
<td>16,099</td>
</tr>
<tr>
<td><strong>Combustion</strong></td>
<td>Testing a Critical Protective Fluid Using the Community Combustion System Model</td>
<td>39,000</td>
</tr>
<tr>
<td></td>
<td>/                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /                           /</td>
<td>39,000</td>
</tr>
<tr>
<td><strong>Nuclear Energy</strong></td>
<td>Deriving a 3D Traceable Core for Multi-Scale Nuclear Energy Applications</td>
<td>57,949</td>
</tr>
<tr>
<td><strong>Nuclear Physics</strong></td>
<td>Developing a 3D Traceable Core for Multi-Scale Nuclear Energy Applications</td>
<td>57,949</td>
</tr>
</tbody>
</table>

**Jaguar/XT5 job size distribution**

Early Science applications in Jan – Jul 2009 time period

- **CPI**: Hours utilized by jobs requesting greater than 50% of the available resources
- **CPU** Hours utilized by jobs requesting between 30% and 60% of the available resources
- **CPI**: Hours utilized by jobs requesting less than 30% of the available resources

![Jaguar XT5 job size distribution graph](image-url)
Moving to the Exascale

- The U.S. Department of Energy requires exaflops computing by 2018 to meet the needs of the science communities that depend on leadership computing.
- Our vision: Provide a series of increasingly powerful computer systems and work with user community to scale applications to each of the new computer systems.
  - Today: Upgrade of Jaguar to 8-core processors in progress.
  - OLCF-3 Project: New 10-20 petaflops computer based on early DARPA HPCS technology.

Preventing for the Exascale

Long-Term Science Drivers and Requirements

- We have recently surveyed, analyzed, and documented the science drivers and application requirements envisioned for exascale leadership systems in the 2020 timeframe.
- These studies help to:
  - Provide a roadmap for the ORNL Leadership Computing Facility.
  - Uncover application needs and requirements.
  - Focus our efforts on those disruptive technologies and research areas in need of our and the HPC community's attention.
**OLCF-3 system description**

- Same number of cabinets, cabinet design, and cooling as Jaguar
- Operating system upgrade of today’s Cray Linux Environment
- New Gemini interconnect
  - 3-D Torus
  - Globally addressable memory
  - Advanced synchronization features
- New accelerated node design
- 10-20 PF peak performance
- Much larger memory
- 3x larger and 4x faster file system
- ≈ 10 MW of power

---

**OLCF-3 Applications Analysis**

informed by two requirements surveys

- Project application requirements
  - Elicited, analyzed, and validated using a new comprehensive requirements questionnaire
  - Project overview, science motivation & impact, application models, algorithms, parallelization strategy, SW development process, SQA, V&V, usage workflow, performance
  - Results, analysis, and conclusions documented in 2009 OLCF application requirements document

- OLCF-3 baseline plan developed in consultation with 50+ leading scientists in many domains
  - What are the science goals and does OLCF-3 enable them?
  - What might the impact be if the improved science result occurs?
  - What does it matter if this result is delivered in the 2012 timeframe?
**What do the Science Codes Need?**

What system features do the applications need to deliver the science?

- 20 PF in 2011–2012 time frame with 1 EF by end of the decade
- Applications want powerful nodes, not lots of weak nodes
  - Lots of FLOPS and OPS
  - Fast, low-latency memory
  - Memory capacity ≥ 2GB/core
- Strong interconnect

---

**Coverage Table for 10 Sample Applications**

<table>
<thead>
<tr>
<th>App.</th>
<th>Science Area</th>
<th>Algorithm(s)</th>
<th>Grid type</th>
<th>Programming Languages</th>
<th>Geophysical Variables</th>
<th>Computational Libraries</th>
<th>Math Libraries</th>
<th>Acceleration Status</th>
<th>Difficulty (1-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD</td>
<td>Climate</td>
<td>spectral finite elements, linear &amp; non-linear equations, particles</td>
<td>structured</td>
<td>F90, Python, R, REX</td>
<td>TRAX</td>
<td>MPI, TRAX</td>
<td>3</td>
<td>Times identified for acceleration, programmed in C, Fortran</td>
<td>4</td>
</tr>
<tr>
<td>DAKK</td>
<td>materials</td>
<td>molecular dynamics, FIC, classical equations</td>
<td>non-structured</td>
<td>C++, C</td>
<td>MPI, TRAX</td>
<td>COMM</td>
<td>3</td>
<td>Identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
<tr>
<td>CMXV</td>
<td>biology</td>
<td>molecular dynamics, FIC, particles</td>
<td>non-structured</td>
<td>C++, C</td>
<td>MPI, TRAX</td>
<td>COMM</td>
<td>3</td>
<td>Identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
<tr>
<td>MADNESS</td>
<td>chemistry</td>
<td>quantum chemical, molecular dynamics, particles</td>
<td>structured</td>
<td>F90, Python, R, REX</td>
<td>TRAX</td>
<td>MPI, TRAX</td>
<td>3</td>
<td>Times identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
<tr>
<td>KIO</td>
<td>combustion</td>
<td>chemical reaction, finite difference equations, particles</td>
<td>structured</td>
<td>F90, Python, R, REX</td>
<td>TRAX</td>
<td>MPI, TRAX</td>
<td>3</td>
<td>Times identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
<tr>
<td>DBN</td>
<td>self-assembly</td>
<td>molecular dynamic, fluid dynamics, chemical reactions, particles</td>
<td>structured</td>
<td>F90, Python, R, REX</td>
<td>TRAX</td>
<td>MPI, TRAX</td>
<td>3</td>
<td>Times identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
<tr>
<td>DIP</td>
<td>reactor codes</td>
<td>sodium-coated reactor, particle physics</td>
<td>structured</td>
<td>C++, Python, Fortran</td>
<td>TRAX</td>
<td>MPI, TRAX</td>
<td>3</td>
<td>Times identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
<tr>
<td>AOKA</td>
<td>Solid</td>
<td>thermal inertia, particle physics</td>
<td>structured</td>
<td>F90, Python, R, REX</td>
<td>TRAX</td>
<td>MPI, TRAX</td>
<td>3</td>
<td>Times identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
<tr>
<td>ML</td>
<td>numerical</td>
<td>fluid dynamics, finite difference equations</td>
<td>structured</td>
<td>F90, Python, R, REX</td>
<td>TRAX</td>
<td>MPI, TRAX</td>
<td>3</td>
<td>Times identified for acceleration, programmed in C, Fortran</td>
<td>3</td>
</tr>
</tbody>
</table>
### Science Application Computing Requirements

- We have elicited, analyzed, & validated science application requirements using a new comprehensive requirements questionnaire.
  - Analyzed project overview, science motivation & impact, application models, algorithms, parallelization strategy, SVN development process, SQA, V&V, usage workflow, performance.
  - Results, analysis, and conclusions documented in 2009 OLCF application requirements document.

![Diagram showing requirements](http://www.ncsa.illinois.edu/media/ncsa_reports/olcf-requirements.pdf)

### Community Library Requirements

<table>
<thead>
<tr>
<th>Science domain</th>
<th>Code</th>
<th>LO Libraries</th>
<th>Math Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator design</td>
<td>TIP</td>
<td>Nek5000</td>
<td>Omega, ParMETIS, Ztan</td>
</tr>
<tr>
<td>Aerodynamics</td>
<td>CORINA</td>
<td>HDF5, gPROMS</td>
<td>LAMMPS, Fortran, Ztan</td>
</tr>
<tr>
<td>Biology</td>
<td>DSSP</td>
<td>HDF5</td>
<td>LAMMPS, FFTW</td>
</tr>
<tr>
<td>Chemistry</td>
<td>MDL/CHARMM</td>
<td>BLAS</td>
<td>HEA, SuiteSparse, FFTPACK</td>
</tr>
<tr>
<td>Climate</td>
<td>CAM</td>
<td>Nek5000</td>
<td>SuiteSparse, FFTPACK</td>
</tr>
<tr>
<td>Fluids</td>
<td>GERRIS</td>
<td>Nek5000</td>
<td>SuiteSparse, FFTPACK</td>
</tr>
<tr>
<td>Materials science</td>
<td>LSDD</td>
<td>Nek5000</td>
<td>Nek5000, PARPACK, EXAS</td>
</tr>
<tr>
<td>Nanoscience</td>
<td>CASINO</td>
<td>BLAS</td>
<td>HEA, SuiteSparse, FFTPACK</td>
</tr>
<tr>
<td>Nuclear physics</td>
<td>Coupled</td>
<td>MPICH</td>
<td>HEA, SuiteSparse, FFTPACK</td>
</tr>
<tr>
<td>QCD</td>
<td>MUEC, Chroma</td>
<td>BLAS</td>
<td>HEA, SuiteSparse, FFTPACK</td>
</tr>
</tbody>
</table>

![Table showing community library requirements](http://www.ncsa.illinois.edu/media/ncsa_reports/olcf-requirements.pdf)
Appendix C. Presentations

It is Important to Analyze Performance

Science Algorithms: Current and Trends
Appendix C. Presentations

Algorithms Moving Forward
What are application teams asking for?

- Automated diagnostics
  - Dynamic performance analysis, application verification, bug
  fixing, H/W fault detection and correction, failure prediction and
  avoidance, system tuning, and requirements analysis
- Hardware latency
  - Will see improvement nearly as much as flop rate, parallelism, SW in coming years?
  - Can SW strategies mitigate high HW latencies?
- Hierarchical algorithms
  - Applications will require algorithms aware of the system
    hierarchy (compute/memory)
  - In addition to hybrid data-parallelism, and I/O-based
    checkpointing, algorithms may need to include dynamic
    decisions between recomputing and storing fine-scale
    task-data hybrid parallelism, and in-memory checkpointing
- Parallel programming models
  - Improved programming models needed to allow developer
    to identify an arbitrary number of levels of parallelism and
    map them onto hardware hierarchies at runtime
  - Models continue to be coupled into larger modules, driving
    the need for arbitrary hierarchies of task and data
    parallelism
- Solver technology and innovative solution techniques
  - Global communication operations across 10^6 processors
    will be prohibitively expensive. Future will likely to
    diode-based communication where flexible and
    robust, inelastic effects where it cannot be avoided. Research
    on more effective local preconditioners will become a very
    high priority.
  - If increases in memory latency continue to lag the number of
    cores added to each node, future research needs to focus
    on ways to effectively hide I/O for memory latencies
- Accelerated time integration
  - Are we ignoring the time dimension along which to exploit
    parallelism? (e.g. elements)
- Model coupling
  - Coupled models require effective models to implement,
    verify, and validate the coupling, which can occur across
    wide scales and temporal scales. The coupling
    requirements drive the need for robust methods for
    distribution, upscaling, and expert workbench scaling
  - Evaluation of the accuracy and importance of coupling
    drives the need for methods for validation, uncertainty
    analysis, and sensitivity analysis of these complex models
- Maintaining current libraries

PF Survey Findings

- A rigorous & evolving apps regime process pays dividends
  - Needs to be quantitative: apps cannot "feel" with
    performance analysis
- Algorithm development is evolutionary
  - Can we break this model?
  - Ex. Explore new parallel dimensions (time, energy)
- Hybrid/multi-level programming models virtually nonexistent
- No algorithm "sweet spots" (one size fits all)
  - But algorithm footprints share characteristics
- V&V and SQA not in good standing
  - Frivolities on compute systems as well as expt results
  - No one is really clamoring for new languages
- MPI until the water gets too hot (fog analogy)
- Apps lifetimes are >3-5x machine lifetimes
  - Refactoring a way of life
- Fault tolerance via defensive checkpointing due to
  fads standard
  - Won't this eventually bite us? Artificially drives I/O
    demands
- Weak or strong scale or both (no winner)
- Data analytics paradigm must change
  - The middleware layer is surprisingly stable and
    agnostic across apps (and should expand!)
Summary & Recommendations: EF Survey

- We are in danger of failing because of a software crisis unless concerted investments are undertaken to close the H/W-S/W gap
  - H/W has gotten way ahead of the S/W (same ole – same ole?)
- Structured grids and dense linear algebra continue to dominate, but …
  - Increase projected for Monte Carlo algorithms, unstructured grids, sparse linear algebra, and particle methods (relative decrease in F/FTs)
  - Increasing importance for AMR, implicit nonlinear systems, data assimilation, agent-based method, parameter continuation, optimization
- Priority of computing system attributes
  - Increase: interconnect bandwidth, memory bandwidth, mean time to interrupt, memory latency, and interconnect latency
    - Reflect desire to increase computational efficiency to use peak flops
  - Decrease: disk latency, archival storage capacity, disk bandwidth, wide area network bandwidth, and local storage capacity
    - Reflect expectation that computational efficiency will not increase
  - Per-core requirements relatively static, while aggregate requirements will grow with the system

Summary & Recommendations: EF Survey

- System software must possess more stability, reliability, and fault tolerance during application execution
  - New fault tolerance paradigms must be developed and integrated into applications
    - Job management and efficient scheduling of these resources will be a major obstacle faced by computing centers
- Systems must be much better “science producers”
  - Strong software engineering practices must be applied to systems to ensure good end-to-end productivity
  - Data analytics must empower scientists to ask “what-if” questions, providing S/W & H/W infrastructure capable of answering these questions in a timely fashion (Google desktop)
  - Strong data management will become an absolute at the exscale
- Just like H/W requires disruptive technologies for acceleration of natural evolutionary paths, so too will algorithm, software, and physical model development efforts need disruptive technologies (Invest now!)
Appendix C. Presentations

The Joule Metric

(SC GG 3.1/2.5.2) Improve computational science capabilities, defined as the average annual percentage increase in the computational effectiveness (either by simulating the same problem in less time or simulating a larger problem in the same time) of a subset of application codes. Efficiency measure: X% (FY09, x=100)

The Annual Joule Process

Benchmark information gathered per application

- Description of Problem Domain, Target Problems
- Description of Application Software, Algorithm Implementation
- Benchmark Parameters Q2
  - problem instance
  - build environment, build
  - runtime environment, run script
- Benchmark Results Q2
  - performance data
  - wall time
  - machine events
  - simulation results
- Benchmark Parameters Q4
  - problem instance
  - build environment, build
  - runtime environment, run script
- Benchmark Results Q4
  - performance data
  - wall time
  - machine events
  - simulation results
- Comparative Analysis of Q2 and Q4 results
  - description of problem related findings
  - description of software enhancements

FY 2009 Annual Report of Joule Software Metric SC GG 3.1/2.5.2, Improve Computational Science Capabilities

December 2009

Appendix C. Presentations

**Joule Metric Exercise**
Benchmarked applications FY04-FY10

<table>
<thead>
<tr>
<th>Category</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>climate research</td>
<td>4</td>
</tr>
<tr>
<td>condensed matter</td>
<td>2</td>
</tr>
<tr>
<td>fusion</td>
<td>5</td>
</tr>
<tr>
<td>high energy physics</td>
<td>2</td>
</tr>
<tr>
<td>nuclear</td>
<td>2</td>
</tr>
<tr>
<td>subsurface modeling</td>
<td>1</td>
</tr>
<tr>
<td>astrophysics</td>
<td>2</td>
</tr>
<tr>
<td>combustion chemistry</td>
<td>4</td>
</tr>
<tr>
<td>bioinformatics</td>
<td>1</td>
</tr>
<tr>
<td>math, data analytics</td>
<td>2</td>
</tr>
<tr>
<td>reservoir dynamics, electronica structures</td>
<td>2</td>
</tr>
<tr>
<td>nuclear energy</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>28</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray</td>
<td>X1</td>
</tr>
<tr>
<td></td>
<td>X1E</td>
</tr>
<tr>
<td></td>
<td>XT3</td>
</tr>
<tr>
<td></td>
<td>XT4</td>
</tr>
<tr>
<td></td>
<td>XT5</td>
</tr>
<tr>
<td>IBM</td>
<td>SP Power3</td>
</tr>
<tr>
<td></td>
<td>P690</td>
</tr>
<tr>
<td></td>
<td>Power5</td>
</tr>
<tr>
<td></td>
<td>BGI/L</td>
</tr>
<tr>
<td>SGI</td>
<td>Altix</td>
</tr>
<tr>
<td>HP Itanium-2</td>
<td>QCDOC</td>
</tr>
</tbody>
</table>

**FY09 Joule Metric Results Summary**
Applications: RAPTOR, CAM, XGC1, Visit

<table>
<thead>
<tr>
<th>Application</th>
<th>Visit</th>
<th>CAM</th>
<th>XGC1</th>
<th>RAPTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Hardware (core)
  - Q4 13,728
  - Q8 26,472

- Time (seconds)
  - Q4 0:00:46 per scenario, 8.378

- Metric target
  - Q2-Q4 core time = 1.0
  - Q2-Q4 core time = 2.0

- Metric result
  - 1.00

Managed by UT-Battelle for the U.S. Department of Energy
Appendix C. Presentations

Software effectiveness on Jaguar/XT5
Improved computational science capabilities: Joule metric

- Four FY09 "Joule codes" have been exercised extensively on Jaguar/XT5 during ES period
  - They possess science drivers for both strong and weak scaling
  - A very successful set of performance improvements have been realized for all 4 codes
    (by virtue of this metric)

- Raptor
  - Chemically reacting, turbulent multiphase flows in complex geometries
  - Combustion modeling of turbulent flame dynamics and mixing processes
  - Performance improvement: MPI halo communication redesign refactor

- VisIt
  - Open source visualization and analysis tool for processing massive data sets
  - 2005 R&D 100 award winner with > 100K downloads
  - Performance improvement: MPI all-to-all communication redesign refactor

- CAM
  - Fifth generation Community Atmosphere Model of the NCAR AGCM used in climatic studies
  - Performance improvement: moving to a hybrid MPICH/OpenMP node model

- XGC1
  - 5-D gyrokinetic particle-in-cell code designed to model the whole plasma dynamics in experimentally realistic device geometry
  - Performance improvement: moving to a hybrid MPICH/OpenMP node model

- Much more forthcoming in FY09 Q4 Joule report
  - Take away: pushing scalability as driven by the science led to unanticipated performance improvements and fixes
  - Scaling is hard! Even the best people working on mature are confronted with surprises

---

Supplemental Material
**Appendix C. Presentations**

---

**OLCF-3 and Roadrunner**

- **Similarities**
  - both leverage commodity processors
  - both forward-looking, hybrid architectures
    - CPU + accelerator hardware, MPI + threading programming models
  - both represent best forward-looking systems deliverable in their respective time-frames and budgets

- **What’s different about OLCF-3?**
  - widespread adoption of GPU hardware as accelerator
  - widespread adoption of GPU programming model
  - large and rapidly-growing user/developer community
    - wide array of software that we can leverage
    - many Universities now teaching GPU programming
  - solid future for technology beyond HPC
  - broader acceptance now that "MPI-everywhere" is nearing end-of-life

---

**Lessons-learned from Roadrunner**

- Strong management support is critical.
- A close and open partnership with vendor is key.
- A small team of domain scientists and talented computational scientists can accelerate real applications.
- Development of tools for new architectures is challenging.
  - talented computational scientists can make whatever is available work, but effective tools are essential for greater developer productivity
- The specific accelerator hardware is almost irrelevant.
  - The primary intellectual challenge was not programming the Cell or dealing with multiple compilers, but rather restructuring applications with an emphasis on data movement over arithmetic.
  - Modifications of this type will deliver improved performance on a wide spectrum of future architectures.
  - Accelerating kernels & libraries is a necessary first step in preparing applications for future architectures.
This process has resulted in some similarities across applications.

- most DOE-SC applications are MPI everywhere
  - some use subgroups to handle on-node concurrency
  - often the subgroup algorithm is deployed on a few nodes
- a few codes are hybrid OpenMP / MPI
  - OpenMP intra-node and MPI inter-node
    - GTC, XGC1, POP, FV-CAM
  - a few Pthreads-based codes exist
    - MADNESS
- many codes leverage optimized libraries
  - BLAS, LAPACK, ScaLAPACK, FFT (FFTW)
  - PETSc, SuperLU, Trilinos

What happens when you add accelerators?

- The nature of code segments that can be off-loaded to an accelerator are similar to those that can be paralleled with OpenMP or Pthreads.
  - both threading & acceleration require identification of self-contained work blocks of appropriate granularity
  - both require enough work to offset the overhead of the constructs involved
    - thread creation, synchronization, data movement/sharing
    - overheads are different for the two paradigms
  - both map naturally to nested parallel kinds of algorithms
  - both often require re-structuring / re-factorizing of code
  - both can be done incrementally

MPI + threads is becoming a dominant programming model for emerging architectures, whether homogeneous or heterogeneous.
Appendix C. Presentations

What kind of software infrastructure do we want?

- inter-node layer is straightforward
  - MPI, SHMEM, Global Arrays, Co-Array Fortran, maybe UPC
- intra-node layer that allows us to easily move identified kernels to the accelerator
  - it should be as facile as OpenMP
  - directive-based where accelerator regions are bounded
    - work with C/C++/Fortran
  - single compiler handles all aspects of the system intra-node architecture
  - integrated libraries for BLAS/FFT/LAPACK

Tools can enable more effective application development.

- pre-processing technology to manage complexity
  - ROSE (http://rosecompiler.org/)
  - pre-processor leading to compiler-integrated technologies
  - performance hints, including opportunities for buffering
  - frameworks that generate code
    - MADNESS
    - Tensor Contraction Engine (TCE), http://www.csc.lsu.edu/~gb/TCE/
    - MAGMA (Atlas+ for GPUs)
- build application-centric functionality into compiler/tools chain
- encapsulate appropriate prescribed tasks for accelerator work
  - similar to evolution of vectorizing or OpenMP compilers & technologies