Summary: Panel Discussion on Large-scale Astrophysical Calculations

Robert Rosner

Depts. of Astronomy & Astrophysics and Physics, The University of Chicago, Chicago, Illinois 60637

Abstract. I summarize the key points of the discussion following the astrophysics presentations in the panel discussion sessions covering large-scale simulations in high energy, accelerator and astrophysics.

INTRODUCTION

Large-scale computations come in two flavors in astronomy and astrophysics: First, modern astronomical data sets are now largely digital in nature, and have reached levels of size and complexity that call for large-scale storage and archiving facilities, and require significant computational resources in order to analyze the data. There is a serious shift in how observational data is gathered and organized: as the sky is being digitized over a broad energy range (from the radio to gamma rays), we are seeing a shift to the "Virtual Observatory", in which the notion of "observing" is coming to mean the remote mining of a multi-wavelength digital data repository. Second, large-scale parallel computational facilities have now reached the point that numerical simulations can now be used both to model astrophysical systems with considerable fidelity, and to explore the physics of astrophysical systems by using numerical simulations as a "laboratory" for conducting astrophysically-relevant "experiments"; such calculations produce "fully instrumented" data sets, allowing heretofore unrivaled detailed study of physically interesting regimes. (What has changed to make this possible is that the dynamic range of 3-dimensional numerical simulations can now be sufficiently large that the Reynolds number of flows can be pushed into the turbulent regime, so that it is now possible to start the exploration of the fully nonlinear behavior of such systems). The Panel provided a hint of the broad range of astronomical problems that can now be tackled via very large-scale computations.

Numerical simulations

Two of the computational talks focussed on the extremes of modern astrophysical simulations: Bruce Fryxell (1) discussed calculations related to thermonuclear flashes associated with degenerate stars, and Paul Ricker (2) related experiences with simulations which coupled fluid dynamics to gravitational N-body calculations (in the context of structure and cluster formation). Characteristic of these types of calculations is that parallel communications are both local and global; and that the latter type of communications are unavoidable in most astrophysics calculations involving gravity, magnetic fields, or radiation transport.

Data Bases, Data Mining, and Exploratory Data Analysis

The data-oriented talks focussed on two of the essential aspects of modern astronomical data bases: first, Alex Szalay (3) showed how one goes about to create a data base whose size can be in the terabyte range, yet allows multiple users effective and practical access, based on his experiences with Object-Oriented data base programming of the SDSS[1] science data base. As an illustration of what can be done with such a data base, Jim Annis (4) discussed the problem of finding structure in the SDSS data. The computations encountered in this domain involve primarily local parallel communications, and therefore scale extremely well on massively-parallel machines.

THE ISSUES

Our discussions centered on the various issues which we must deal with in developing application tools for large-scale computations. These issues largely reflect the fact that software for our computational hardware (e.g., large parallel computers) remains immature, especially in the commercial market; that the half-life of much of this

[1] SDSS = Sloan Digital Sky Survey

hardware is very short, measured in 1-2 years; and that standards (in computer architecture, in code architecture, and – in general – in the computing and coding infrastructure) are far from settled.

Language paradigms

The first issue discussed revolved around the role of object-oriented (OO)-based programming. It quickly became clear that OO-based programming is becoming more pervasive, primarily because the programming community in astrophysics is realizing the advantages to be gained: code re-use via modular design, ease of code maintenance, greater emphasis on clean code design (and whence increased code reliability). As for the specifics of what language to use, there remains considerable diversity: many hydrodynamicists continue to rely on Fortran (though in its more modern flavors, e.g., F90/F95), while the data-intensive astrophysicists are focusing on the use of C/C++. Some sentiments were expressed that in the ideal programming environment, code agnosticism should prevail; and that proper "glue" components (e.g., wrappers, written in languages such as Python) would be able to take care of interfacing between modules written in different languages.

Code architecture paradigms

The second issue raised focused on the programming paradigm for structuring large codes. Two contending philosophies vie for the favor of the coder: "frameworks" and "toolkits". Frameworks tend to define all data structures and interfaces, and building a new code basically entails constructing the modules for computing the desired physics, and "hooking" them to the framework backbone. For example, frameworks such as Lawrence Livermore's *Samrai* which are designed for computational physics/fluid dynamics provide services such as mesh generation, adaptive mesh refinement, various types of time stepping algorithms, solvers for various types of partial differential equations, input/output and visualization tools, all packaged to work seamlessly on a parallel machine. In contrast, toolkits focus their energies on defining a rich set of interfaces, and thus rely on a very lightweight code framework. Modules such as mesh generators, adaptive mesh codes, visualization tools and so forth are then all regarded as peers; this is the code architecture paradigm represented by the Common Component Architecture (CCA) working group. Obviously, the choice between these two paradigms depends entirely on the particulars of the application: In my view, application codes with well-defined, restricted application domains are likely to benefit from the rigor and consistency of frameworks; while application codes that are likely to have varied (and difficult to define in advance) application domains are most likely to benefit from the flexibility of toolkits. Toolkits also have the (current) advantage of broader support – frameworks tend to be supported by small individual development groups, with little commonality between competing frameworks.

Unique issues to simulations

Certain issues that were discussed were clearly unique to the world of numerical simulation. For example, until very recently, the dominant parallel programming paradigm has been the use of message-passing (primarily based on MPI), which works well for single processor/node architectures such as that of Beowulf clusters. However, several computer vendors (IBM perhaps the most prominent) have pushed towards a clustered SMP parallel architecture, based on multiple processors per node (with the ASCI White IBM machine currently at 16 processors/node); in this case, the use of threads (and the OpenMP paradigm) has become much more important in order to achieve decent parallel efficiency. Since MPI does not currently support threads, we are facing the (hopefully short-term) possibility of once again dealing with non-portable codes.

The verification and validation of codes has become a much more visible stumbling block for code development. Modern code development efforts deal with verification (meaning determining whether a code contains bugs, and properly solves the equations it is supposed to solve) by enforcing strict code development procedures (including version control and automated regression tests). Validation (meaning determining whether a code correctly solves physical problems purported to be approximated by the equations defining the code) in contrast remains a research issue: typically, it is not trivial to arrange laboratory (or other) experiments so that the characteristic control parameters overlap with those governing the numerical simulations; and for those cases in which such an overlap can be constructed, it remains unclear exactly what is meant by "agreement" between code and experiment: local measures are very likely inappropriate (because of the high sensitivity of nonlinear systems to initial conditions), and global measures (e.g., energy spectra) may be insufficient.

Unique issues to Data Mining and Exploratory Data Analysis

Similarly, the world of large-scale data set analysis must also face a number of unique problems. Perhaps the most pressing (and most often mentioned) is the quest for a stable, bug-free, feature-rich OO-based data base. The core problem seems to be that the commercial market for such data bases remains very limited, so that the commercial vendors tend to be insufficiently capitalized to be able to afford the level of problem-resolution (viz., bug-fixing) and feature extension that modern data base efforts such as the SDSS require. A related problem is the fact that the expertise for constructing such databases does not reside in the astronomical community; the expertise instead resides in computer science departments and in the commercial world. The resulting sociological challenge of getting computer scientists, physicists, and astronomers to work together effectively remain a problem common to all large-scale code development efforts in astronomy.

Boundaries and frontiers

Some discussion focused on the current limitations to computing. The traditional measure of code performance – most sensibly measured by "wall clock time" – continues to play a key role (for example, few of the simulators expressed willingness to give up more that 10-20% in performance as a trade-off for clean modular design!). However, it is also clear that other metrics are becoming important: As codes become more complex, the metrics of "time-to-code-completion" and "time-to-solution" (meaning, the time from starting code construction to the time a solution is finally obtained) become critical. These additional code metrics have played an important role in promoting code modularity and code (module) re-use, since both serve to speed the code development process. Modularity also promotes code portability (since the machine-dependent pieces can be readily isolated), as well as promoting code longevity (since new techniques, or new machine architectures, are far more readily accounted for within the context of a modular code architecture). An interesting observation was that for many forefront computational hydrodynamics problems, run time was no longer the dominant limiting factor: available memory size and archival storage capacity, and I/O and (internet) communication speeds seem to have become critical limiting factors for state-of-the-art hydrodynamic simulations.

Where are the current unsolved bottlenecks? Some examples identified by the various speakers and the audience include:

- Parallel I/O: currently, parallel I/O works relatively poorly, and a number of machines do not yet implement a standard parallel I/O (such as the HDF5 standard).
- Visualization: Visualization tools need to scale, as data sets reach the terabyte scale. Commercial tools simply do not perform well in such applications. Research is progressing in the direction of multi-resolution visualization, but this has not yet reached the desktop.
- Debuggers and performance analyzers: These programming tools also need to scale. Tools such as Argonne National Laboratory's *Jumpshot* go a long way to addressing this problem (for performance analysis), but much more needs to be done.

DISCUSSIONS AND CONCLUSIONS

The panel discussions made clear that the recent huge increase in computing capabilities (especially on the memory size end of things) have led to entirely new ways of using computations in astrophysics: The "virtual observatory" is no longer a dream, but is coming into being; and the use of numerical simulations as an "experimental" tool for astrophysics theorists is becoming commonplace.

Certain "rules of thumb" came up during our discussions that seem to be well-worth highlighting:

- Nothing is for the ages/design for obsolescence: Any given application code is not likely to survive for long. For this reason, allowing for code evolution (via modular design and code component re-use) is a very worthwhile goal.
- Don't "over-optimize": The breakeven point for optimization is usually well in the past, precisely because code life is usually short.
- Getting correct results tops all: No amount of coding elegance or algorithmic cleverness will replace the most important point of all – the application code must produce correct results. For this reason, the common error of short-changing code verification and code validation is a terrible blunder.

ACKNOWLEDGMENTS

I would like to thank the organizers for providing the venue for interesting discussions, Rajendran Raja for collaborating on co-organizing this panel session; and the Department of Energy ASCI/Alliances grant at the University of Chicago for partial support.

REFERENCES

1. Bruce Fryxell, these Proceedings.
2. Paul Ricker, these Proceedings.
3. Alex Szalay, these Proceedings.
4. James Annis, these Proceedings.