

Supercomputing and Science



An Introduction to High Performance Computing

Part VII: Scientific Computing

Henry Neeman, Director
OU Supercomputing Center
for Education & Research



Outline

- Scientific Computing Pipeline
- Scientific Libraries
- I/O Libraries
- Scientific Visualization



Scientific Computing Pipeline

Real World

Physics

Mathematical Representation (continuous)

Numerical Representation (discrete)

Algorithm

Program

Port (to a specific platform)

Result (run)

Analysis

Thanks to Julia Mullen of WPI for this concept.



Scientific Libraries



Preinvented Wheels

Many simulations perform fairly common tasks; for example, solving systems of equations:

$$\mathbf{Ax} = \mathbf{b}$$

where \mathbf{A} is the matrix of coefficients, \mathbf{x} is the vector of unknowns and \mathbf{b} is the vector of knowns.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$



Scientific Libraries

Since there are tasks that are very common across many science applications, groups of researchers have put a lot of effort into writing scientific libraries: collections of routines for performing these commonly-used tasks (e.g., linear algebra solvers).

The people who write these libraries know a lot more about these things than we do.

So, a good strategy is to use their libraries, rather than trying to write our own.



Solver Libraries

Probably the most common scientific computing task is solving a system of equations

$$\mathbf{Ax} = \mathbf{b}$$

where \mathbf{A} is a matrix of coefficients, \mathbf{x} is a vector of unknowns, and \mathbf{b} is a vector of knowns.

The goal is to solve for \mathbf{x} .



Solving Systems of Equations

Don'ts:

- **Don't** invert the matrix ($\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$). That's much more costly than solving directly.
- **Don't** write your own solver code. There are people who devote their whole careers to writing solvers. They know a lot more about writing solvers than we do.



Solving Do's

Do's:

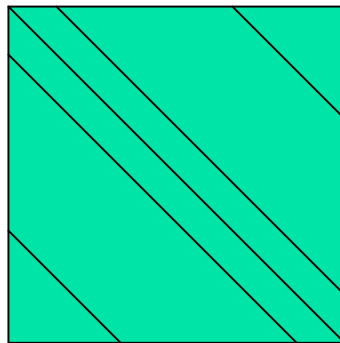
- **Do** use standard, portable solver libraries.
- **Do** use a version that's tuned for the platform you're running on, if available.
- **Do** use the information that you have about your system to pick the most efficient solver.

All About Your Matrix

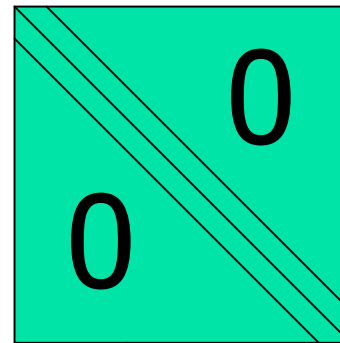
If you know things about your matrix, you maybe can use a more efficient solver.

- Symmetric: $a_{i,j} = a_{j,i}$
- Positive definite: $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ for all $\mathbf{x} \neq 0$
(e.g., if all eigenvalues are positive)

- Banded:
0 except
on the
bands



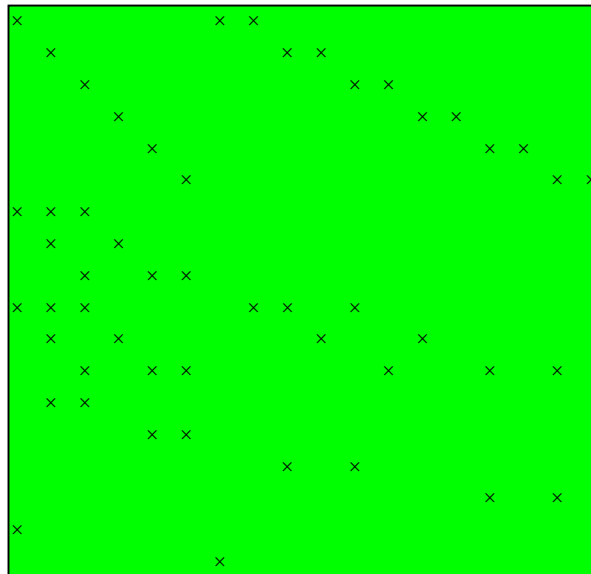
- Tridiagonal:



and ...

Sparse Matrices

A sparse matrix is a matrix that has mostly zeros in it. “Mostly” is vaguely defined, but a good rule of thumb is that a matrix is sparse if more than, say, 90-95% of its entries are zero. (A non-sparse matrix is dense.)





Linear Algebra Libraries

- LAPACK^[1]
- scaLAPACK^[2]
- PETSc^{[3],[4],[5]}



LAPACK

LAPACK (Linear Algebra PACKage) solves dense or special-case sparse systems of equations depending on matrix properties such as:

- Precision: single, double
- Data type: real, complex
- Shape: diagonal, bidiagonal, tridiagonal, banded, triangular, trapezoidal, Hessenberg, general dense
- Properties: orthogonal, positive definite, Hermetian (complex), symmetric, general



LAPACK Example

```
REAL,DIMENSION(numrows,numcols) :: A
REAL,DIMENSION(numrows)           :: B
REAL,DIMENSION(numcols)           :: X
INTEGER,DIMENSION(numrows)         :: pivot
INTEGER :: row, col, info, numrhs = 1

DO row = 1, numrows
  B(row) = ...
END DO !! row = 1, numrows
DO col = 1, numcols
  DO row = 1, numrows
    A(row,col) = ...
  END DO !! row = 1, numrows
END DO !! col = 1, numcols
CALL sgesv(numrows, numrhs, A, numrows, pivot, &
&          B, numrows, info)
DO col = 1, numcols
  X(col) = B(col)
END DO !! col = 1, numcols
```



LAPACK: a Library and an API

LAPACK is a library that you can download for free from the Web:

`www.netlib.org`

But, it's also an Application Programming Interface (API): a definition of a set of routines, their arguments, and their behaviors.

So, anyone can write an implementation of LAPACK.



It's Good to Be Popular

LAPACK is a good choice for serial solving, because its popularity has convinced many supercomputer vendors to write their own, highly tuned versions.

The API for the LAPACK routines is the same as the portable version from NetLib, but the performance can be much better.

Also, some vendors have shared memory parallel versions of LAPACK.



scaLAPACK

scaLAPACK is the distributed parallel (MPI) version of LAPACK. It actually contains only a subset of the LAPACK routines, and has a somewhat awkward Application Programming Interface (API).

Like LAPACK, scaLAPACK is also available from **`www.netlib.org`**.



PETSc

PETSc (Portable, Extensible Toolkit for Scientific Computation) is a solver library for sparse matrices that uses distributed parallelism (MPI).

PETSc is designed for general sparse matrices with no special properties, but it also works well for sparse matrices with simple properties like banding and symmetry.

It has a simpler, more intuitive Application Programming Interface than scaLAPACK.



Pick Your Solver Package

- Dense Matrix
 - Serial: LAPACK
 - Shared Memory Parallel: vendor-tuned LAPACK
 - Distributed Parallel: scaLAPACK
- Sparse Matrix: PETSc



I/O Libraries



I/O Challenges

I/O presents two important challenges to scientific programming:

- Performance
- Portability

The performance issue arises because I/O is much more time-consuming than computation.

The portability issue arises because different kinds of computers can have different ways of representing real (“floating point” numbers).



Storage Formats

When you use a **PRINT** statement in Fortran, you are asking the program to output data in human-readable form:

```
x = 5
```

```
PRINT *, x
```

But what if the value that you want to output is a real number with lots of significant digits?

```
1.3456789E+23
```



Data Output as Text

1.3456789E+23

When you output data as text, each character takes 1 byte.

So if you output a number with lots of digits, then you're outputting lots of bytes.

For example, the above number takes 13 bytes to output as text.

Jargon: text is sometimes called ASCII.



Output Data in Binary

Inside the computer, a single precision real number (Fortran **REAL**, C/C++ **float**) typically requires 4 bytes, and a double precision number (**DOUBLE PRECISION** or **double**) typically requires 8.

That's less than 13.

Since I/O is very expensive, it's better to output 4 or 8 bytes than 13 or more.

Happily, Fortran, C and C++ allow you to output data in binary (internal representation) rather than text.



Binary Output Problems

When you output data in binary rather than text, you output substantially fewer bytes, so you save time (since I/O is very expensive) and you save disk space.

But, you pay two prices:

- Readability: humans can't read binary.
- Portability: different kinds of computers have different ways of internally representing numbers.



Binary Readability: No Problem

Readability of binary data isn't really a problem in scientific computing, because:

- You can always write a little program to read in the binary data and display its text equivalent.
- If you have lots and lots of data (i.e., MBs or GBs), you wouldn't want to look at all of it anyway.



Binary Portability: Big Problem

Binary data portability is a very big problem in scientific computing, because data that's output on one kind of computer may not be readable on another, and so:

- You can't output the data on one kind of computer and then use them (e.g., visualize, analyze) on another kind.
- Some day the kind of computer that output the data will be obsolete, so there may be no computer in the world that can input it, and thus the data are lost.



Portable Binary Data

The HPC community noticed this problem some years ago, and so a number of portable binary data formats were developed. The two most popular are:

- HDF (Hierarchical Data Format) from the National Center for Supercomputing Applications: **`hdf.ncsa.uiuc.edu`**
- netCDF (Network Common Data Form) from Unidata:

`http://www.unidata.ucar.edu/packages/netcdf`



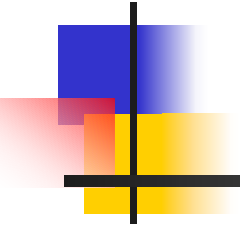
Advantages of Portable I/O

Portable binary I/O packages:

- give you portable binary I/O;
- have simple, clear APIs;
- are available for **free**;
- run on most platforms;
- allow you to annotate your data (e.g., put into the file the variable names, units, experiment name, grid description, etc).

Also, HDF allows distributed parallel I/O.

Scientific Visualization





Too Many Numbers

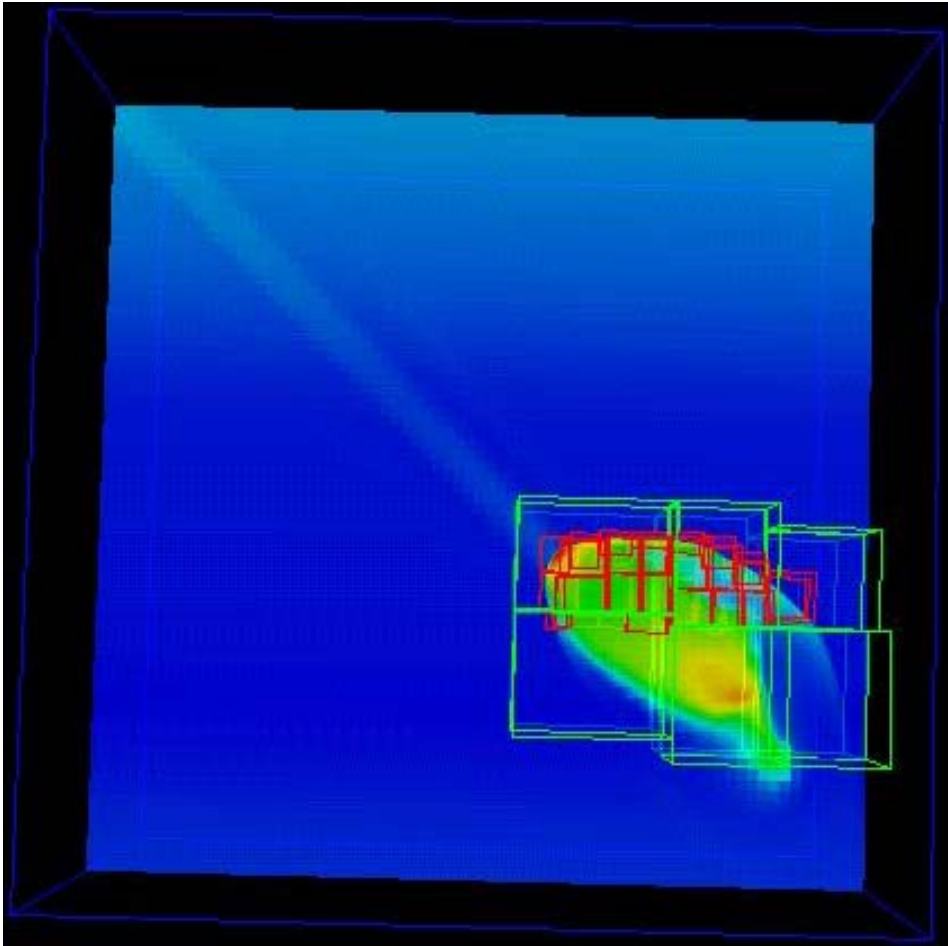
A typical scientific code outputs lots and lots of data.

For example, the ARPS weather forecasting code, running a 5 day forecast over the continental U.S. with a resolution of 1 km horizontal and 0.25 km vertical outputting data for every hour would produce about 10 terabytes (i.e., 10^{13} bytes).

No one can look at that many numbers.

A Picture is Worth ...

... millions of numbers.



This is Comet Shoemaker-Levy 9, which hit Jupiter in 1994; the image is from 35 seconds after hitting Jupiter's inner atmosphere.^[6]



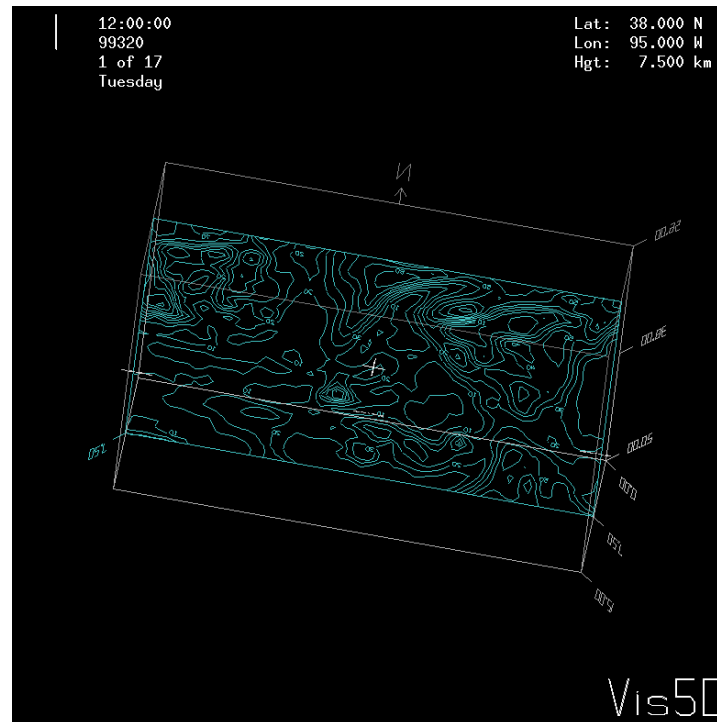
Types of Visualization

- Contour lines
 - Slice planes
 - Isosurfaces
 - Streamlines
 - Volume rendering
- ... and many others.

Note: except for the volume rendering, the following images were created by Vis5D,^[7] which you can download for free.

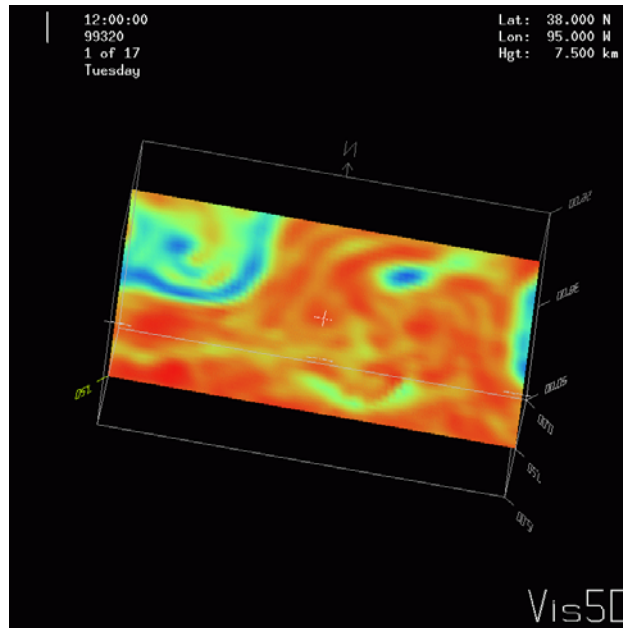
Contour Lines

This image shows contour lines of relative humidity. Each contour line represents a single humidity value.



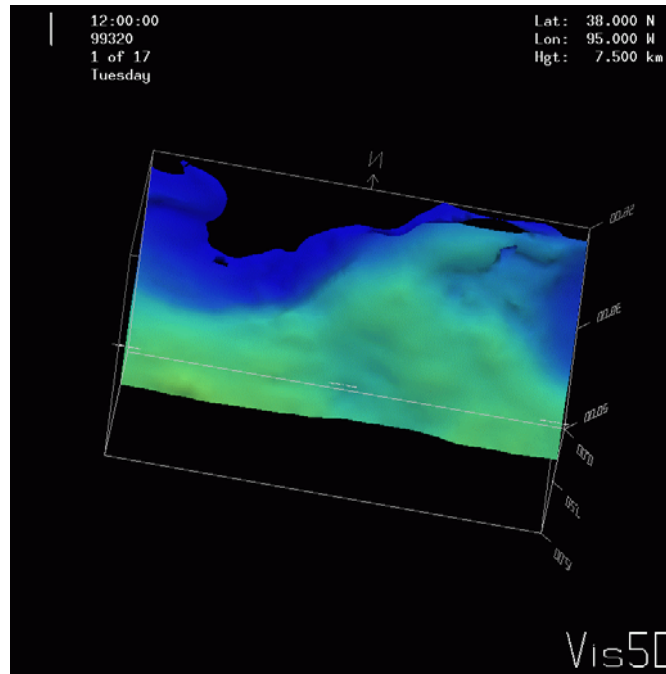
Slice Planes

A slice plane is a single plane passed through a 3D volume. Typically, it is color coded by mapping some scalar variable to color (e.g., low vorticity to blue, high vorticity to red).



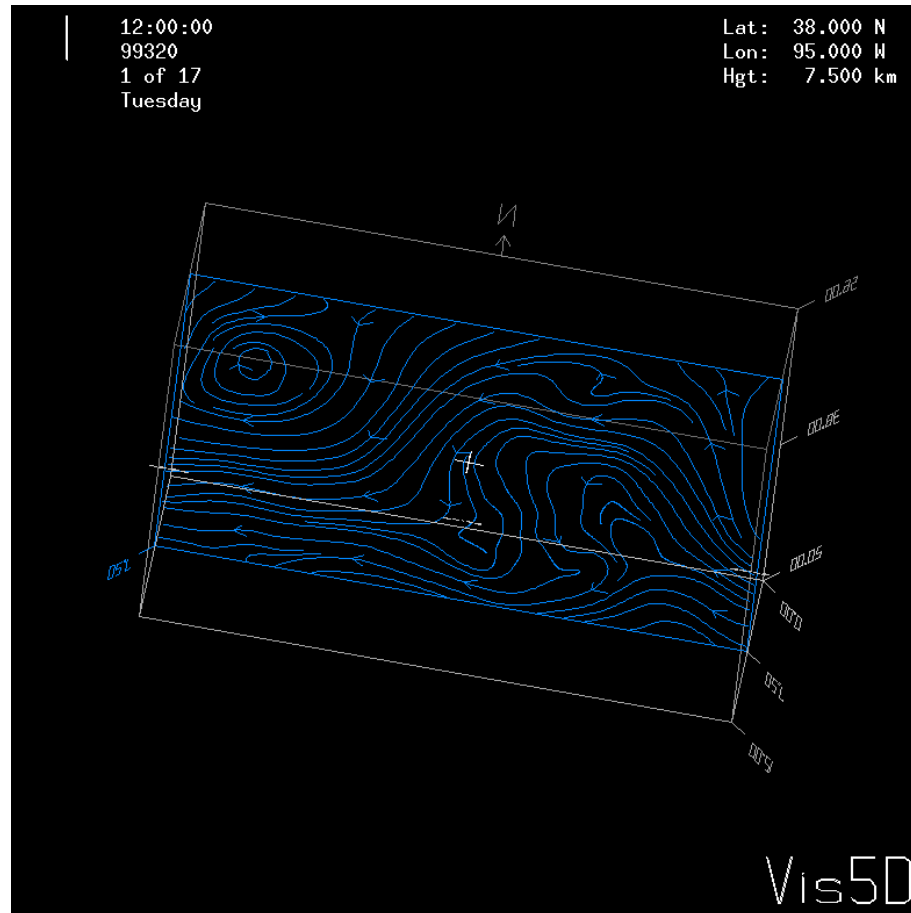
Isosurfaces

An isosurface is a surface that has a constant value for some scalar quantity. This image shows an isosurface of temperature at 0° Celsius, colored with pressure.



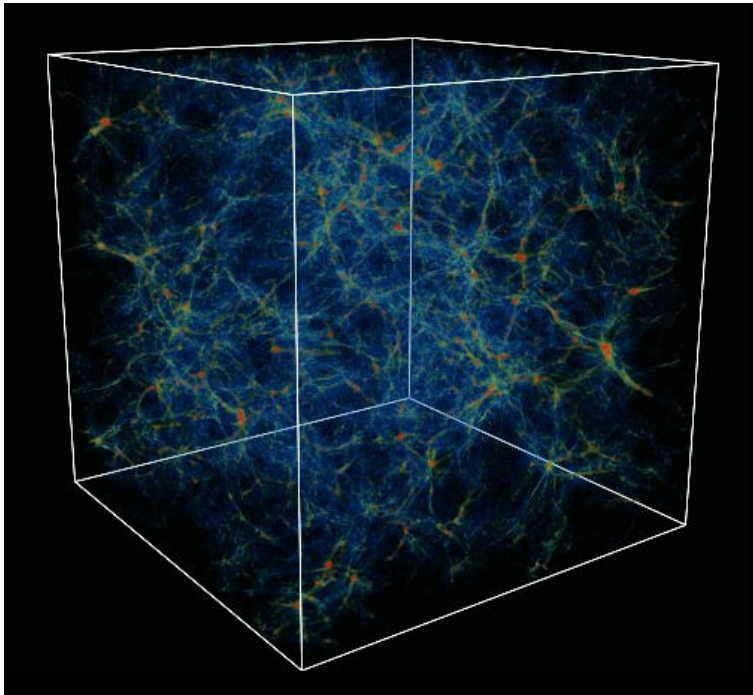
Streamlines

A streamline traces a vector quantity (e.g., velocity).



Volume Rendering

An image is rendered by mapping some variable (e.g., energy) to color and another variable (e.g., density) to opacity:



This image shows the overall structure of the universe.^[8] Notice that the image looks like thick colored smoke.



Have a great break!



References

- [1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK Users' Guide*, 3rd ed, 1999:
<http://www.netlib.org/lapack/>
- [2] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petit, K. Stanley, D. Walker, R. C. Whaley, *ScaLAPACK Users' Guide*, 1997:
<http://www.netlib.org/scalapack/>
- [3] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, L. Curfman McInnes and B. F. Smith, PETSc home page, 2001:
<http://www.mcs.anl.gov/petsc>
- [4] S. Balay, W. D. Gropp, L. Curfman McInnes and B. Smith", *PETSc Users Manual*, ANL-95/11 - Revision 2.1.0, Argonne National Laboratory, 2001.
- [5] S. Balay, W. D. Gropp, L. Curfman McInnes and B. F. Smith, "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries", in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset and H. P. Langtangen, editors, Birkhauser Press, 1997, 163-202.
- [6] See <http://scarecrow.caps.ou.edu/~hneeman/hamr.html> for details.
- [7] <http://www.ssec.wisc.edu/~billh/vis5d.html>
- [8] Image by Greg Bryan, MIT:
http://zeus.ncsa.uiuc.edu:8080/chdm_script.html