Evaluation of Biomedical Applications on Heterogeneous Grid Systems

James E. Dobson
Department of Psychological and Brain Sciences
Dartmouth College
James.E.Dobson@Dartmouth.EDU

January 2004

Abstract
Scientific applications are often a mixture of commercial developed code, open-source community code and in house developed code. Biomedical applications are often developed for a single platform where extensive testing and validation is done for clinical use. Common life science and biomedical data formats are not aware of endian-ness. The use of these applications on the Grid requires a heterogeneous computer architecture.

1 Introduction
The fMRI Data Center and Department of Psychological and Brain Sciences at Dartmouth College have developed a computational grid for the analysis of functional magnetic resonance imaging (fMRI) data. This computational grid contains both x86 GNU/Linux equipment as well as Solaris on x86 and SPARC based servers. We will evaluate four codes used commonly by the neuroimaging community for use on heterogeneous grid architectures including the Sun Fire B1600 Blade Platform. The evaluation criteria for these codes will include memory usage, network I/O, 64-bit issues, ability to handle endian-ness, and scalability.

AIR Automated Image Registration[1]. Developed in C. Developed by The University of California at Los Angeles.


SPM Statistical Parametric Mapping[7]. Code written in Matlab. Developed by the University College London, UK.

calcStats A Java-based application for statistical comparisons of functional runs. Developed by the fMRI Data Center.
These applications will be evaluated on the Sun Fire B1600 Blade Chassis from Sun Microsystems with AMD Mobile Athlon and UltraSPARC IIi-based SPARC B100s blades. Other systems include Sun Fire V240, Sun Fire V65x, Sun Fire V880, and other x86 systems.

2 Functional MRI

Functional MRI (fMRI) is a rapid, non-invasive brain imaging technology. Researchers at the Dartmouth Brain Imaging Center collect data from human subjects performing a variety of cognitive tasks in the MR scanner. These data are then pre-processed and analyzed using a suite of data analysis packages. Since the data are from human subjects special care must be taken to follow institutional polices set by the Institutional Review Board (IRB), the NIH, and the HIPPA. The analysis methods of fMRI are similar to other sciences which enables re-use of many technologies discussed in this paper. The processing pipelines and middleware packages used in this paper will be applicable to a number of fields.

Recent advances in technology have enabled neuroimagers to collect data faster and use additional imaging techniques such as diffusion tensor imaging (DTI). It is not uncommon for as much as 850MByte of raw data to be collected in a single session of a single subject. On a typical data over 2GBytes are collected by the DBIC MR scanner. Recent studies at Dartmouth have involve large numbers of subjects, in one case over 100. The larger studies require on the order of 100GByte for data analysis. The fMRI Data Center has over 2.6TByte of peer-reviewed data. These increased data demands are putting stress on the current computational infrastructure and quickly overwhelm desktop systems.

3 Workflows and Pipelines

Many scientific processes are done in a pipeline. Data are carried through several transformations by individual processing commands. Biomedical and life sciences applications, in particular, often have processing steps to remove identifiers to protect the privacy of human subjects.

A Grid Computing environment that is able to understand the relationships between applications and data dependencies can simplify the work need to be performed by computational scientists. Applications not designed for a scriptable input/output model may not be able to be used effectively in such a processing pipeline. Figure 1 illustrates the flow of data from \( N_1 \), the source images to \( N_2 \) the atlas in common space. \( A \) and \( R \) are data transformations with \( R \) having a dependency on \( A_{1-6} \)'s output and \( N_2 \) dependent on the output of \( R_{1-6} \).
Dartmouth has been experimenting with Data Grid and workflow management software to describe the transformation of data from raw form to a usable state for analysis. Initially scripts were created to do the entire pipeline in one script. These serial scripts ensured that data dependencies were addressed and job ordering was correct. Testing and debugging of large scripts is difficult and restarting a pipeline a stage or two in required a new script. Using directed acyclic graphs (DAG) it is possible to describe a pipeline’s input and output and any data dependencies for an application. Condor [12] has a component called DAGman[3], which is able to schedule jobs based on their DAG. Other packages such as Chimera[5] and LONI Pipeline[13] utilities have been evaluated for integration into various processing scripts. These reproducible scripts enable others to confirm results, test new hypotheses, and experiment with processing steps without re-running the entire analysis pipeline.

4 Dartmouth Green Grid

The Green Grid is college funded campus-wide initiative at Dartmouth to build a shared computational environment. The Green Grid has been architectured to meet the research needs of the newly funded Neukom Institute for Computational Science. The groundwork for this project has been layed by two departments; Psychological and Brain Sciences, and Research Computing. The Green Grid’s goal is to give the entire Dartmouth community access to powerful computational systems and lower the cost of doing computational science. Applications run on the Green Grid are distributed to heterogeneous systems located in multiple departments. The architecture of the Green Grid project is still evolving and exists now as informal relationships between departments. Data access, security, and prioritization of local resources are among the largest concerns of the Green Grid’s users.
The applications tested were run through a stack of integrated Grid software. These tools enabled secure remote execution and data transfer of the applications used for this paper. Operating Systems included Solaris 9, Solaris 10 (build 51 & 52), IRIX 6.5, and Red Hat Linux 7.2 & 9.0. Sun Fire V880’s with 750MHz processors were used for many of the runs. The Sun Fire 1600 chassis was configured with both the B100s with UltraSPARC IIe processors and the B100x with AMD Athlon 1.5GHz processors. The x86 systems included the Sun Fire V65x with Intel Xeon 2.8GHz processors and a Beowulf cluster with 2.4GHz processors. The Sun Performance and Utilization Suite was used for I/O intensive applications. This was shared via NFS to many systems including the B100x and B100s servers. The GridFTP server served data from a multi-terabyte SAM-QFS filesystem.

**CondorG** from the Condor 6.6.3 distribution

**DAGman** from the Condor 6.6.3 distribution

**Globus** Toolkit version 2.4.3

**SGE** Sun Grid Engine 5.3 (OpenPBS used as well)

### 6 Data Transfer

The applications tested for this paper all move a considerable amount of data during a run. There are two major classes of data used in Grid computing; the application executable environment, and the input/output data. Applications, for a heterogeneous Grid, need to be packaged for each platform with the correct libraries and environment available. The input data may be replicated and available from a number of sources. The output of the processing will need to be moved to a location were it can be analyzed if the execution system does not have access to a shared filesystem.
6.1 Application Binaries

There are several options available for transfer and remote execution of application binaries on the Grid systems at Dartmouth. The building of applications for each platform is a more complex problem to solve than the distribution. Platforms with rapidly changing system libraries, such as Linux, can be difficult to utilize without packaging many system libraries with the application binaries.

**AFS** The AFS filesystem is widely deployed at Dartmouth. This network filesystem provides a global namespace for multiple administrative domains. There are three major AFS “cells” at Dartmouth used by Grid users. AFS requires several TCP and UDP ports to be opened which can complicate the use of AFS behind a firewall. AFS has strong authentication via Kerberos and an advanced ACL system which is advantageous for Grid applications. The GSSKLOG application can provide AFS tokens with GSI Grid security.

**PACman** The PACman[4] software distribution package is a self contained application environment. The construction of PACman caches can take considerable resources to repackage existing applications. There are, however, several Internet accessible PACman caches which may have applications already package. PACman has the ability to handle UNIX environment variables. The PACMan application needs to be installed on all the grid nodes, however. It is based on Python which may not be available on all systems.

**GridFTP** Applications distributed with a single binary can be transferred easily using GridFTP. Several of the codes used in this paper were distributed as a single binary executable. Systems behind firewalls may not be able use GridFTP without network address translation (NAT) enable on the firewall.

6.2 Input/Output Data

The majority of the applications tested take their input as the root of the filename to read from the local filesystem. Data transfer was performed by a GridFTP from the compute nodes to a GridFTP server. It is often not appropriate to transfer data to a remote node if the data can be re-created quickly. An example of this is the brain extraction using BET. This processing step takes only a few seconds while the overhead of setting up a GASS connection through Globus can take longer than this. The bandwidth between the two departments used for this paper is 11.45MByte/sec as measured with ttcp. Using HTTP to transfer data the rate drops to 2.9MByte/s. GridFTP is able to transfer near 10MByte/sec. Scripts were written for these tests to use tar to archive the results and GridFTP them back to the GridFTP server. The blade servers in the B1600 chassis had a dedicated network interface to a NFS server which served a SAM-QFS filesystem with MR data. These systems wrote directly to the QFS filesystem if it was available.
7 Application: Time-series statistics

An application used by the fMRI Data Center is calcStats. This code is a Java threaded code to generate statistics on time-series of functional runs. The calcStats code takes as input a directory name which contains a single run of pre-processed fMRI data. Endian issues are addressed in calcStats when reading in data. The output of calcStats is a Java serialized object and several volumes for each statistic generated. The dataset used for this paper had 128 3D time-points of volumes which had dimensions of 79x95x68. A total of 264Mbyte and 256 input files were read in by calcStats. The serialized object was a single 45MByte file, with an additional 48MByte of data also generated. The entire time-series is loaded into memory, on the trial runs 510,340 voxels were used for 128 volumes. As a result calcStats has large memory requirements. Typical runs used 2GByte of memory. With longer scanning sessions 64-bit systems will be needed. It is run normally on 8 processor Sun Fire V880 servers with 16GBytes of memory.

The calcStats code uses the FFTW library though a JNI call. Compiled versions of the library were available for both Solaris/SPARC and Linux/x86. A version of FFTW was not built for Solaris x86 for running on the B100x blades. The calcStats application would be a good candidate for running on a blade server architecture. Each functional run could be assigned to a blade (provided enough memory was present) and the results stored on a network file server.

| Binary size     | 233MBytes of Java and C code     |
| Memory usage    | 2GBytes                       |
| Input           | 256 2MByte files , and 256 348Byte file |
| Output          | One 45MByte files, 48MByte additional data |
| Average wallclock time | 27 minutes (8 CPU) |

Table 1: Application Requirements for calcStats
8 Application: Image Segmentation

The FAST code from the FMRIB was run on 107 high resolution anatomical volumes obtained from the Brain Imaging Lab (BIL) at Dartmouth. The images were T1-weighted volumes with dimensions of 256x256x124. The processing scripts segment each volume in three tissue types: Grey Matter, White Matter, CSF. The FMRIB Brain Extraction Tool (BET) is used first to extract the brain from the skull. The brain volume is then opened and the requested tissues are segmented into new volumes for each tissue. Data from public data warehouses such as the fMRI Data Center are anonymized first to remove identifiers such as facial features and the skull.

Figure 4 shows a slice of the source brain volume and slices from the three tissues segmented from the anatomical volume. The FAST code is a single compiled C++ executable. It was compiled for UltraSPARC using the GCC compiler due to a incompatibility with the Sun C++ templates. On Linux and Solaris x86 the binary was generated with GCC as well. The FAST code is regarded as having one of the faster single-threaded MR image segmentation algorithms. The FAST code is able to handle both little and big endian data. This enables a the pipeline to run BET on x86/Linux and FAST on a SPARC/Solaris system. FAST has higher memory requirements than some of the other FSL codes. This can be a problem on shared multiprocessor machines which have other queues defined.

Figure 4: Full Volume, CSF, Gray Matter & White Matter

9 Application: Creation of a Brain Atlas

Template-base atlases are used for the normalization of data into a common space. Commonly used brain atlases are constructed from over 300 averaged
<table>
<thead>
<tr>
<th>Binary size</th>
<th>2.4MBytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory usage</td>
<td>333MBytes</td>
</tr>
<tr>
<td>Input</td>
<td>One 16MByte file, and one 348Byte file</td>
</tr>
<tr>
<td>Output</td>
<td>Four 7.8MByte files, and four 348Byte files</td>
</tr>
<tr>
<td>Average wallclock time</td>
<td>16 minutes</td>
</tr>
</tbody>
</table>

Table 2: Application Requirements for FAST

anatomical volumes. The construction of a population based probabilistic atlas is computational expensive. Algorithms to register anatomical data can take a long time to run. Each high resolution anatomical image is 256x256x124 voxels which is 16MBytes. The process of building an atlas is embarrassingly parallel, a source image is selected and each volume is registered to this source to produce an average. The anatomical data needs to be a stripped brain for the registration algorithms to register in an acceptable timeframe. The BET[6] code from the FMRIB was used for the extraction of the brain. Then a pipeline is constructed of tools from the AIR package. Each stage of the pipeline takes in the ANALYZE format data so data conversions are not needed. AIR’s align_warp was run with a twelfth order, nonlinear 1365 parameter model to register the 3D volumes.

Pipeline for building of an anatomical atlas in common space:

1. **bet** BET does a brain extraction from the anatomical volume.
2. **align_warp** A non-linear 3D image registration algorithm.
3. **reslice_warp** Reslices the original volumes to a new volume in common space.
4. **softmean** Creates an average of the resliced files as atlas in common space.

AIR’s align\_warp code writes the path to the volume to be resliced into the warp file. An alternate volume needs to be specified for the reslice with an option to reslice\_warp. Endian-ness is an issue with AIR. AIR was also modified to output 16-bit data as short ints with scaled voxel intensity values from 0-32767. With some work it is possible to run AIR on heterogeneous Grid architectures.

<table>
<thead>
<tr>
<th>Binary size</th>
<th>551KBytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory usage</td>
<td>32Mbytes</td>
</tr>
<tr>
<td>Input</td>
<td>Two 16MByte files, and two 348Byte files</td>
</tr>
<tr>
<td>Output</td>
<td>2.1KByte (align_warp) 16Mbyte (reslice), 16MByte (softmean)</td>
</tr>
<tr>
<td>Average wallclock time</td>
<td>50 minutes</td>
</tr>
</tbody>
</table>

Table 3: Application Requirements for AIR Pipeline

10 Application: Interactive Data Analysis

The Statistical Parametric Mapping (SPM [7]) package is used for interactive data analysis. SPM is a package of scripts and compiled code which runs under Matlab. SPM, as built, requires a X11 server for display of data as an analysis runs. Users within the Dartmouth Brain Imaging Center (DBIC) run Matlab as an interactive job within the local Sun Grid Engine cluster. Interactive queues have been defined for these jobs. Batch scripting of SPM is done for many common tasks, however, it still requires an X11 server for display of images while it runs. SPM handles endian-ness well, it writes data out in the endian format of the machine used to process data. Since SPM is built on top of Matlab it requires a license to run. Dartmouth has a campus-wide license for matlab but this can still present a problem for systems on private networks and behind firewalls.

The Matlab environment requires the addition of new paths to add support for a new application. Some of the code is compiled using the Mathworks mex compiler. The entire SPM distributed is 30MByte. The code was run out of the AFS filesystem for this paper. Batch scripts resided in each users home directory.
Acknowledgements
The Brain Imaging Lab (BIL) including Joe M. Moran, Heather L. Gordon, and William Kelley at Dartmouth College provided anatomical data and time on their server. Thanks to Scott Frey for assistance with FSL. The fMRI Data Center, including John D. Van Horn, Jeffrey B. Woodward, and Michael C. Schmitt. Susan Schwarz at Dartmouth Research Computing gave me access to their Beowulf clusters. Fred Smith and David Killian at Sun provided the B1600 server chassis and B100x and B100s servers.

References

