Chapter in ’Contemporary HPC Architectures’
Contents

1 LLGrid: Supercomputer for Sensor Processing 1
Jeremy Kepner, William Arcand, William Bergeron, Nadya Bliss, David Bestor, Chansup Byun, Matthew Hubbell, Peter Michaleas, Julie Mullen, and Andrew Prout, Albert Reuther, Antonio Rosa, Charles Yee
  1.1 Overview ........................................... 2
  1.1.1 Program background ........................... 2
  1.2 Applications and workloads ....................... 3
    1.2.1 Application characteristics ................ 4
    1.2.2 Benchmarking ............................... 7
  1.3 System overview ................................... 8
  1.4 Hardware Architecture ........................... 9
  1.5 System Software .................................. 9
  1.6 Programming system ............................. 10
  1.7 Data center/facility ............................ 11

Bibliography 13
Chapter 1

LLGrid: Supercomputer for Sensor Processing

Jeremy Kepner, William Arcand, Willam Bergeron, Nadya Bliss, David Bestor
Chansup Byun, Matthew Hubbell, Peter Michaleas, Julie Mullen
Andrew Prout, Albert Reuther, Antonio Rosa, Charles Yee

MIT Lincoln Laboratory, Lexington, MA

1.1 Overview

1.1.1 Program background

1.2 Applications and workloads

1.2.1 Application characteristics

1.2.2 Benchmarking

1.3 System overview

1.4 Hardware Architecture

1.5 System Software

1.6 Programming system

1.7 Data center/facility

This work is sponsored by the Department of the Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.
1.1 Overview

MIT Lincoln Laboratory is a federally funded research and development center that applies advanced technology to problems of national interest. Research and development activities focus on long-term technology development as well as rapid system prototyping and demonstration. A key part of this mission is to develop and deploy advanced sensor systems. Developing the algorithms for these systems requires interactive access to large scale computing and data storage. Deploying these systems requires that the computing and storage capabilities are transportable and energy efficient. The LLGrid system of supercomputers allows hundreds of researchers simultaneous interactive access to large amounts of processing and storage for development and testing of their sensor processing algorithms. The requirements of the LLGrid user base are as diverse as the sensors they are developing: sonar, radar, infrared, optical, hyperspectral, video, bio and cyber. However, there are two common elements: delivering large amounts of data interactively to many processors and high level user interfaces that require minimal user training. The LLGrid software stack provides these capabilities on dozens of LLGrid computing clusters across Lincoln Laboratory. LLGrid systems range from very small (a few nodes) to very large (40+ racks).

1.1.1 Program background

Sensor prototyping at Lincoln spans a wide range of sensor modalities: sonar, radar, infrared, optical, hyperspectral, video and cyber. The goal is to demonstrate sensor systems that push the limits of size, resolution and bandwidth. These requirements have been a part of Lincoln Laboratory since before its inception and can be traced back to the first radar processing systems developed at the MIT Radiation Laboratory during World War II [1]. At the start of the Cold War MIT Lincoln Laboratory was established to develop anti-aircraft defenses. Interactive, high performance digital computing systems were at the core of this effort and led to such groundbreaking systems as Whirlwind, TX-0 (Transistor eXperiment zero), and TX-2 [2]. Whirlwind was the first magnetic core memory system. The TX-0 design was spun out of Lincoln Laboratory and formed the basis of Digital Equipment Corporation (now a part of Hewlett-Packard). TX-0 was one of the first computer systems equipped with a graphical user interface (GUI) and among its many accomplishments were the interactive first video games.

Today, development of sensor systems at Lincoln requires access to parallel computing and parallel data storage by hundreds of users. In addition, the timeframe for sensor algorithm development is often very short (weeks to months). Algorithm developers need to work interactively in high level languages (e.g., Matlab, Java, and Python). These requirements have led to the
LLGrid family of supercomputers (TX-α, TX-β, TX-2500, TX-3D, TX-DoD, TX-X, TX-Green, . . .). The LLGrid parallel high level language work and interactive scheduler have been described extensively in the literature (see [3, 5, 4] and references therein). In addition, the LLGrid architecture has had a significant impact outside of Lincoln Laboratory and has influenced the design and deployment of a range of systems such as the Mathworks Parallel Computing Toolbox and the Microsoft .NET framework.

As the amounts of data produced by sensors increases, sensor algorithms are increasingly being deployed in an energy constrained environment. Demonstrating the algorithms on energy efficient hardware reduces deployment risks. TX-Green is the first LLGrid system that will be an interactive supercomputer in a transportable and energy efficient infrastructure. In addition, TX-Green is the pathfinder for the 20 Megawatt Massachusetts Green High Performance Computing Center (MGHPCC.org) located next to the hydroelectric power station in Holyoke, MA.

The DoD High Performance Computing Modernization Program (HPCMP) has been the primary sponsor of many of the largest LLGrid systems via its Distributed High Performance Investments (DHPI) portfolio. The goal of DHPI is to address needs that lie outside of those normally met by the HPCMP supercomputing centers. These needs typically include novel hardware (in this case energy efficient supercomputing) and workloads (in this case interactive supercomputing via high level programming environments).

1.2 Applications and workloads

The dominant use of LLGrid systems is for interactive processing of sensor data. This work typically occurs within the context of prototyping a new sensor system that consists of three components: sensors, algorithms, and computing. Likewise, the development of these systems typically occurs in three steps:

1. Initial sensor data collect
2. Develop algorithms
3. Port algorithms to an embedded real-time system

LLGrid primarily services users in steps 2 and 3 of this process. Sensor algorithm development is a highly interactive and iterative process that takes the algorithm analyst from a rudimentary understanding of the data to a complete characterization of all its signals, noise, and clutter. The iterative process consists of changing the algorithm code, running the code, and observing the results of the code. The goal of LLGrid is to allow the user to quickly complete
these iterations on large data sets. A unique feature of LLGrid is that it is also a deployable system. Thus, if a user's code runs on LLGrid then it will also run on a similarly configured deployed system.

The above process requires LLGrid to be highly interactive. As such, all processors are devoted to interactive usage and users jobs run immediately. LLGrid systems are sized so that during normal loads, there are always some processors available to the users. Larger fractions of the system are available to the user upon requests that can usually be satisfied inside a few minutes. The median job on LLGrid is <30 seconds and is a result of users debugging their code. The mean job on LLGrid is >30 minutes and are jobs that run for hours after the code is working properly.

1.2.1 Application characteristics

The details of sensor processing algorithm development differ for each sensor. However, there are some common computational elements, such as:

- Data upload
- File format conversion
- Metadata harvesting
- Registration
- Clutter detection
- Clutter removal
- Background characterization
- Background normalization
- Target detection
- Target parameter estimate
- Target track estimation
- Track fusion

LLGrid users are first and foremost scientists and engineers. The LLGrid system has been optimized to help the users get through the above steps with as little effort as possible. For nearly all the steps, the focus is the LLGrid central filesystem. The inputs and output of nearly every stage of sensor processing development flow through the LLGrid central file system[7].
Data upload

A typical data set will consist of a finite duration of data collected from a prototype sensor system. The volume of data is typically constrained to what can be held on several commodity storage devices. The data are typically in files with formats that are unique to the sensor. The size and number of files can range from many small files to a few large files. The first step is to transfer the data from the storage devices into the LLGrid central filesystem either via network or by directly connecting the storage devices into LLGrid. The data transfer time can range from hours to days. To make this process simple for the users, they can mount the LLGrid central file system to their desktop and simply copy files via “drag and drop.”

Conversion and Metadata Harvesting

The next step in the process is for the analyst to write a program that converts all the data from a custom file format to a more generally usable format. In addition, the file metadata (e.g., collection time, collection location, sensor parameters, etc.) will often be harvested and stored in a database. At this point the analyst is now in a position to start developing their signal processing algorithms.

Registration

Sensors that are pointed toward the ground must be registered with respect to a global coordinate position. This can be as simple as reading the coordinate info from the metadata or as hard as comparing the image with a large collection of other images to determine the precise location of the image. In the former case, the computational power is minimal, while in the latter case a large amount of parallel computing power is required to compare many images. Often, a large part of the algorithm development effort can be expended during the highly iterative process of changing an algorithm and determining if the registration has improved. Typically high level languages are used during this algorithm development process. As the registration improves, the user will increase the number of images they test their algorithm on until eventually they are registering all the of images.

Clutter

Many sensor systems have defects that cause them to produce spuriously high or low signals. Likewise, certain environmental conditions can cause similar sensor responses. Regardless of their source, these signals are clutter and need to be removed in order to see targets of interest. The process for detecting, characterizing and removing clutter is very similar to that for registration, although it is usually less computationally intensive. A highly iterative process of changing an algorithm and determining its effects on clutter is used. Typically these algorithms are implemented in high level languages.
Background

The process of detecting targets typically consists of modeling the data as a combination of random background noise and non-random targets. Characterizing the background requires sampling some of the data to compute means, variances and other statistical measures. These statistical measures are then applied to all the data to normalize or subtract off the background signal from the measurements. The algorithms for analyzing the background signal can range from fairly simple to very complex. The process of developing and testing background removal algorithms is similar to registration and clutter removal.

Targets

After removing clutter and background noise, in theory, all that is left are the targets of interest. Target detection algorithms typically try to establish a noise varying threshold for determining that a target is present. In addition, after a detection has been made, the target parameters (e.g., location, size, and shape) are often estimated via additional algorithms. The ultimate goal of the entire registration, clutter removal, background subtraction and detection process is to maximize the probability of detection ($P_D$) and minimize the probability of false alarm ($P_{FA}$). Developing these algorithms is similar to the processes mentioned in the previous steps. However, while the data read in is large (i.e., all the data that has come through the previous steps), ideally, the data written out is small. In addition, while the previous steps were data independent (i.e., the amount of processing does not depend upon the actual measured values), the amount of processing for targets is highly variable and depends upon the number and distribution of targets. In all the previous steps, all the data is read in and written out to a filesystem. In detection, it is common to write out target information to a database.

Tracks

The increasing use of persistent sensors that cover large areas for long times can result in many detections. Detection measurements are aggregated into tracks that combine a whole series of related measurements. These tracks represent data over space and time and can contain a variety of metadata associated with these tracks. Track construction and analysis is an active area of research, but the approach to developing these algorithms is very similar to those for target detection, with the exception that input detections will usually come from a database. In addition, the mathematics of track analysis is often “sparse” or based on graphs[6] instead of dense and a based on traditional signal processing techniques. As a result, the computational efficiency of these “post-detection” or “back-end” algorithms are often significantly lower than the “pre-detection” or “front-end” algorithms.
Repeat

Development of large collections of track data has now spawned a whole new area often referred to as Signal Processing on Graphs (SPG)[8]. In SPG, the techniques of registration, clutter removal, background subtraction, detection and tracking are then reapplied to these large collections of relatively unstructured data to mine tracks of interest buried in collections of track data. This work is driving the development of an entirely new set of tools and mathematics to deal with the sparse signal processing of databases[6, 16, 15].

1.2.2 Benchmarking

Benchmarking plays two important roles in LLGrid. The first is acceptance testing of the hardware to verify that the vendor has provided what was specified. The second is to optimize the software tools in the LLGrid software stack.

Acceptance testing

LLGrid is a fleet of dozens of clusters, made of mostly commodity components. Even the largest of these clusters is too small to justify the effort to develop a custom procurement and acceptance test. Furthermore, competitive pressures mean that the vendors can only devote a fixed amount of time to satisfying an acceptance test.

The LLGrid goal for acceptance testing is to verify that the hardware components delivered by the vendor are as specified. The HPC Challenge[9] benchmark suite (see www.hpcchallenge.org) is one solution to this problem. HPC Challenge tests the four key components of an HPC system: processor, memory, network bandwidth and network latency. HPC Challenge is an open benchmark freely available with publicly posted results. In addition, HPC Challenge is required by many large supercomputing procurements so vendors are familiar with it and often have in-house expertise in optimizing and running HPC Challenge.

The process LLGrid follows for using HPC Challenge in acceptance testing is as follows. After a vendor has been selected the peak HPC Challenge performance for the system is estimated by picking the closest equivalent system in the public HPC Challenge results and extrapolating to the precise size of the system being acquired. The HPC Challenge benchmarks are typically “contest” level benchmarks that are achieved with a significant amount of system tuning that may be specific to the benchmarks (e.g., specific network buffer sizes and memory swap sizes). For acquisition purposes, the performance on the system as it will be configured for use is more important, and the target performance is set at half the peak estimated performance. These targets provide for a good balance between customer verification and vendor time spent acceptance testing.

While HPC Challenge is good for measuring the peak performance of the
system, there is also a need to do endurance testing to verify the reliability of the system. For endurance testing LLGrid again uses HPC Challenge and follows the DoD HPCMP acceptance testing procedures that are also well known to many supercomputing vendors. The HPCMP procedures indicate that endurance testing is to be conducted for 14 days while keeping the system at least 50% loaded. This is easily done with a simple set of scripts that continuously run HPC Challenge over a wide range of sizes. In addition, at the end of these test there is a wealth of data that can be submitted to the HPC Challenge website for the benefit of the community.

Optimization

LLGrid optimization is primarily focused on the LLGrid specific software stack after the hardware has been accepted. The most important of these is the optimization of the LLGrid central filesystem. The filesystem performance is measured using the IO Zone benchmark (see www.iozone.org).

Most LLGrid users use high level programming environments such as Java, Python and Matlab. Parallel Matlab performance is optimized using the pMatlab implementation of HPC Challenge[3], which is a part of the pMatlab package.

Increasingly, graph and database operations have become more important to LLGrid users. The Graph500[10] benchmark (see www.graph500.org) is used for measuring the performance of both sparse matrix (i.e., graph) operations and database performance [11].

1.3 System overview

LLGrid is designed for rapid prototyping of sensor processing algorithms. The LLGrid architecture meets this design goal by using:

- Commodity processing nodes that run standard software without re-compilation
- Open source system software that allows the system to be tuned for its mission
- Application specific interfaces for rapidly prototyping sensor algorithms
1.4 Hardware Architecture

LLGrid compute hardware can be thought of as a commodity cluster with a few key distinguishing features. LLGrid compute nodes have high memory per core because high level programming environments are memory intensive. LLGrid compute nodes have a large amount of local RAID storage that can support various distributed filesystem and distributed database technologies.

The vast majority of LLGrid network traffic flows through its central filesystem. The LLGrid central storage array is not a commodity system and is usually the fastest system that can be purchased that runs an open source parallel filesystem. In addition, the network topology is designed around the filesystem. A single core switch is used so all nodes have equal access to the filesystem. Historically, the support for file IO has been broadest for the Ethernet fabrics and this is what LLGrid systems use.

1.5 System Software

LLGrid compute nodes need to run the same software that LLGrid users run on their desktops. The system software on LLGrid compute nodes is similar to a desktop Linux configuration with kernel extensions to support the parallel filesystem and additional security. The system software is updated monthly to keep up with the rapid pace of software packages in the Linux community. Many of the packages that LLGrid users run are large. To reduce pressure on the central filesystem these packages are installed on the compute nodes. Multiple versions of these packages are installed so that the compute node can exactly match the version of the software the user is running on their desktop. To support these large system images the LLGrid team has written a high performance system imager that it optimized to these particular requirements.

The central filesystem is mounted on every compute node, every login node, and every user’s desktop system. All LLGrid software tools are made available to the users via the LLGrid central filesystem and most users run jobs on LLGrid directly from their desktop environment without logging into LLGrid. The central filesystem is an open source parallel file system (e.g., Lustre) that allows the LLGrid team to tune the performance via modifying the system parameters and by modifying the source code.

The interactive nature of LLGrid requires that jobs be launched immediately. All jobs run on LLGrid are executed and managed by open source schedulers (e.g., GridEngine). Using an open source scheduler allows the LL-
Chapter in ‘Contemporary HPC Architectures’

Grid team to tune the performance via modifying the scheduler parameters
and by modifying the source code.

LLGrid system status information is collected in real-time on every node
and every job run on LLgrid. This data is then fed into a set of high perfor-
mance data analytics tools and databases. The data is analyzed and displayed
in a 3D model of the LLGrid hardware that is accessible via a Massively Multi-
player On-line Role Playing Game (MMORPG) interface[13]. The interface
provides instantaneous visual feedback on the health and status of all LLGrid
resources. For example, nodes can change color, size, and position based on
their state. The location of a node in the Game interface is identical to that in
the real data center. The data analysis system continually compares all mea-
sured values (e.g., CPU load and free memory) with target values and sends
out alerts whenever these are exceeded. These alerts have significantly im-
proved the reliability of LLGrid as they identify transient behaviors in users
programs that can become unstable if they collide with other users. These
program behaviors are almost always unintentional and the user is pleased to
be made aware of these issues as it makes their programs run faster and more
reliably.

More recently, the LLGrid stack has begun to incorporate the ability to
launch Virtual Machines (VMs) and “Big Data” technologies (e.g., Hadoop).
Interestingly, LLGrid users are most interested in the capability offered by
these technologies, but not the specific technologies themselves (see next
section).

1.6 Programming system

Distributed arrays (or Partitioned Global Address Spaces - PGAS) is the
dominant programming model on LLGrid. Distributed arrays work naturally
with matrices and linear algebra. Most LLGrid users have no prior experience
in parallel computing. Distributed arrays allow these users to get up and
running quickly. A typical LLGrid user can go from account setup to getting
real speedup on their application in less than two hours.

Most LLGrid user programs follow the pattern of distributing an array
across a large number of processors, performing calculations on the local part
of the array, and then performing a collective operation on the entire array.
This pattern is then looped over many times within the program. The dis-
tributed arrays programming model supports these programs very well. The
distributed arrays programming model is independent of whether the user is
running on distributed memory nodes, shared memory nodes, or a hybrid.
Distributed arrays are also good for expert programmers because they sup-
port very complex data movements such as multi-dimensional transposes and
nearest neighbor boundary conditions. Combining high level programming en-
environments with distributed arrays consistently results in a high productivity parallel programming environment[14].

Message passing programming models are also supported by LLGrid. These are typically used via 3rd party software. LLGrid support several flavors of open source MPI libraries so as to be compatible with these 3rd party packages.

Users do not need to write scheduler submission scripts to run a job on LLGrid. The appropriate scheduler scripts are automatically constructed when the user launches a parallel job in a high level environment. Some users do write scheduler submission scripts to execute programs that consist of running the same program on a list of inputs. These programs fall within the map/reduce parallel programming model. As the map/reduce parallel model grew in popularity, the LLGrid team implemented a simple LLGridMapReduce command line program that automatically generates and submits the appropriate scheduler scripts for such a job. Users enjoy the simplicity of map/reduce, but most outgrow it quickly[17]. Map/reduce is a good entry point into parallel programming, but it is entirely encompassed by the the distributed arrays model, which is more scalable and can handle more complex programs.

The increasing need to apply signal processing techniques to unstructured data (e.g., text, cyber record, DNA sequences) has led to the need to support databases for these applications. The LLGrid team has developed the Dynamic Distributed Dimensional Data Model (D4M) library to provide such an interface. D4M allows users to interact with unstructured data via strings and graphs. Data in a parallel program or in a database are both represented using an “associative” array which is a large sparse matrix that uses strings for the rows, columns and values. The primary advantage of D4M is that it allows complicated algorithms to be written more quickly than using standard approaches[16].

1.7 Data center/facility

LLGrid is spread across a range of facilities that range from small rooms to the 300 rack MGHPCC. Increasingly, the energy cost and environmental impact of supercomputing has become important. To mitigate this, LLGrid has led in the development of data center capabilities in Holyoke, MA next to the 50 megawatt hydroelectric station. At this site the electrical costs are approximately half what is paid at LLGrid’s other sites and 90% of the electricity used emits no CO₂.

Costs are further reduced by using container based computing infrastructure that “breathes” in response to the computing load and the external environment. Using this approach it should be possible to avoid cooling for 90%
of the year. Using direct expansion cooling eliminates the need for any water infrastructure. Finally, using a high speed digital power transfer switch eliminates the cost and infrastructure incurred by a backup generator.
Bibliography


A. Rosa, and C. Yee, “Driving Big Data With Big Compute,” IEEE HPEC, Sep 10-12, 2012, Waltham, MA


