McKenzie: A Teraflops Linux Beowulf Cluster for Computational Astrophysics

John Dubinski [a,1], Robin Humble[a], Chris Loken[a], Ue-Li Pen[a] and Peter Martin[a]

[a] Department of Astronomy and Astrophysics and Canadian Institute for Theoretical Astrophysics,
Mclennan Labs, 60 St. George St., University of Toronto, Toronto, ON M5S 3H8

[1] *dubinski@astro.utoronto.ca*

We describe a new 512-CPU Linux Beowulf cluster built at Toronto dedicated to problems in computational astrophysics. The cluster incorporates a new network topology based on inexpensive, 24-port gigabit switches and exploitation of a second gigabit port on each server. This topology provides network performance competitive with more expensive networking configurations. Our speed rating is 1.11 Teraflops on 512 CPUs according to the HPL Linpack benchmark on the Top 500 Supercomputers list. We also describe some preliminary results from our first simulations on this new machine.

Nous décrivons ici une nouvelle grappe 512 processeurs, de type Beowulf, conçue à Toronto et dédiée à la résolution de problèmes numériques en astrophysique. La grappe prend avantage d'une nouvelle topologie réseau basée sur l'utilisation de commutateurs gigabit de 24 ports peu dispendieux, ainsi que sur l'usage d'un deuxième port gigabit pour chacun des serveurs.B Cette topologie réseau permet d'atteindre une performance comparable à celle de configurations réseau plus dispendieuses. La puissance mesurée est de 1.11 Teraflops sur 512 processeurs selon le test HPL Linpack cité sur la liste Top 500 des plus puissants superordinateurs. Nous montrons également quelques résultats préliminaires provenant de nospremières simulations sur cette nouvelle grappe.

Overview

The Olympic motto "Citius, altius, fortius" ("Swifter, higher, stronger") succinctly describes the direction of 21st century supercomputing - swifter processors, higher resolution and stronger algorithms. In computational astrophysics, the need for all of these qualities is even greater than most supercomputing applications. While the bulk of the physics of the formation of the planets, stars, galaxies and the large-scale structure in the universe are now well understood and in many cases the initial conditions are well posed, the challenge of computing the formation of objects and structures in the universe is difficult. The standard methods of computational fluid dynamics (including magnetohydrodynamics) and gravitational N-body simulation are stressed by the huge dynamic range in physical quantities - density, pressure, temperature - that exist in nature. Ever-finer computational meshes and greater numbers of N-body particles are needed to capture the physics of the formation of things in the universe correctly.

The steady growth of computational speed and capacity through Moore's law has played a large role in improving the quality and reality of simulation results. However, the development of parallel algorithms has allowed researchers to leap ahead another factor of up to 1000 or more by harnessing the computational power of symmetric multi-processor (SMP) supercomputers or clusters of networked computers running Linux commonly called Beowulf clusters.

In recent years, the relative low cost of commodity servers running the Linux operating system and more importantly switches and network interface cards has led to the growth in performance and competitiveness of Beowulf clusters versus SMPs. The recent list of the Top 500 fastest supercomputers (www.top500.org) reveals that 55 out of the top 100 are clusters rather than SMPs or vectorized supercomputers. The price of commodity gigabit networking using familiar copper ethernet has dropped dramatically this past year and for many applications is competing directly with low-latency but higher-priced networking options such as Quadrics and Myrinet.

The spirit of building a Beowulf cluster is to get the most bang for the buck in terms of gigaflops per dollar. Most high performance Beowulfs now use low-latency networking with costs several times the cost of current commodity gigabit networking. Large-scale 200+ port gigabit switches are a slightly cheaper option but still eat significantly into the cost of a cluster. Networking costs alone can be more than half the cost. We describe here a new cheaper option that uses off-the-shelf 24-port commodity gigabit switches in a 256- node/512 CPU cluster. We see below that this configuration performs well with a HPL LINPACK benchmark of 1.11 Tflops placing it in the top 50 computers (rank 41) according to the Top 500 list of November 2002 and currently number one in Canada by this same measure. The key to achieving this high performance is new innovation in the networking topology involving both a stack of commodity 24-port gigabit SMC switches and the configuration of each Linux server as a network router through a second gigabit network port. Our economical networking strategy competes directly with both large-scale 200+ port switches and low-latency networking at a fraction of the cost. The ability to optimize the Linux kernel's various functions is also critical to getting the best performance.

Cluster Hardware Specifications

In this section, we describe the design and hardware specifications of the cluster. The hardware was assembled and installed by Mynix Technologies, Montreal, PQ at the University of Toronto. The CITA Beowulf Cluster dubbed McKenzie is comprised of 268 dual Xeon 2.4 Ghz rack-mounted format Linux servers (536 processors in total) distributed over 7 racks (Figure 1). All the nodes are 1U format (1.75” high) with the exception of two 2U format head nodes. The hardware specifications for each server are given in Table 1. From the total of 268 nodes, 256 nodes are dedicated to parallel, message passing supercomputing, 8 nodes for code development and smaller scale applications, 2 spare nodes also running to act as hot replacements in case of hardware failures on the compute nodes and 2 head nodes. The main head node “bob” contains the home disk space as well as the root space images and Linux kernels for the compute slave nodes. All slave nodes boot through the network from bob using PXElinux allowing easy kernel upgrades and maintenance for the cluster nodes. A secondary master node “doug” mirrors bob’s root and home directories on a daily basis and acts a backup in case of a bob crash. This allows quick recovery and continued operation if bob fails.

Motherboard, CPUs and Memory

Each server is based on the Intel Westville motherboard that fits into a 1U chassis and runs dual Xeon 2.4 Ghz processors and 4 memory slots. We currently are running with 1 GB RAM per node though we could expand to 2 GB. There are also 2 integrated gigabit network ports both of which we exploit in our networking topology. There is also one PCI slot that is unused.

Disks

Each server contains two 80 GB IDE hard drives each of which is divided into 3 partitions - 2 small 2 GB partitions to hold the root filesystem and the swap space and one large partition for scratch disk space. The root filesystem is installed on the first disk and swap on the second. Two empty partitions are reserved for possible new future Linux installations. The large partitions are joined with the Linux software RAID0 providing about 150GB of scratch disk space per node. This space is intended for checkpointing data but also future applications designed for out-of-core memory usage. The total available scratch space is about 46 TB.

The Westville board includes a Promise hardware raid controller. We discovered that detection of this controller in the BIOS made network booting impossible so we disabled the hardware raid configuration in the BIOS. We did examine the performance of the onboard raid controller using a patched Redhat installation provided by Intel. The RAID performance was similar if not slightly worse than Linux software RAID. The NFS performance of the Intel patched distribution was poor. This patch only worked for one specific kernel so we decided to go with Linux software raid to allow us the flexibility of kernel tuning for networking, disk and other performance issues. Network booting is also a necessity for installation and operation of a cluster our size so the Promise raid controller was seen as a major pain rather than a benefit. In the end, we are forced to boot the compute nodes with PXE or a boot CDROM and the head nodes themselves can only be booted with a boot CDROM. We hope these BIOS issues are resolved soon.

We also have attached a 3 TB RAID box on the head node to act as a centralized repository for simulation data.

The head nodes bob and doug were configured with 20 GB root partitions plus a spare 20 GB partition used for mirroring. Bob is used to install and monitor nodes on the cluster as well as acting as the cluster entry point and repository of centralized home disk space for users. Bob’s root partition is mirrored daily on doug’s spare partition and vice versa. Essential system files and the home filesystem are also mirrored from bob to doug daily. If bob crashes, doug can quickly take over cluster operation through a quick reboot of the mirrored partition and some minor system reconfiguration.

Cooling

Cooling is important for a densely packed 1U rack-mounted configuration. Each chassis contains 3 high-powered fans that draw air in through openings in the front and the sides of the chassis. Hot air is blow out the back. Our racks are located in a central machine room with large air-conditioning capacity. Floor tiles in the front and back of the machine are grilled to maximize cool airflow. We carefully monitor the temperature of each node using onboard sensors through the Linux kernel using the Ganglia toolkit. Generally, we find the cooling is adequate though we have identified a few hot nodes running near 70 deg. C and have taken action to cool them down. The mean node temperature when the machine is fully loaded is 50 deg. C that is within specs for the motherboard.

Power

We have allocated about 350 W of power per node according to the motherboard specifications. At full operation, the cluster requires about 90 kW of power.

Switches and Cables

The cluster is networked using 19 SMC Tiger 24-port configurable gigabit switches. Seventeen switches are used for the main 256-node cluster network while the development cluster is connected to its own switch and the 19th switch is reserved as a spare and connected to the running spare nodes. We defer the discussion of the network topology to the section below.

Rack Configuration

The nodes and switches are mounted on 7 racks as shown in Figure 1. The racks are 44U high with 128 compute nodes mounted on the first 3 and last 3 racks. The central rack is reserved for the head nodes, development nodes and spares as well as the stack of switches. The switches are mounted on the back of the racks to simplify the cabling. The interface is a special 1U monitor and keyboard which slide in and out of the rack.

TABLE 1. Mynix PowerRACK-HX Parts List

Nodes:	256 compute nodes + 8 development nodes + 2 head nodes + 2 spares = 268 total
Chassis:	Chenbro RM11802 1U format (1.75” high)
Motherboard:	Intel Westville SE7500WV2
CPUS:	Dual Xeon 2.4GHz processors E7500 chipset/512KB L2 cache, 400 MHz system bus
Memory:	1 Gbyte DDR-200 RAM
Disk Drives:	2X80 Gbyte Seagate IDE drives
Networking:	Dual Intel Pro/1000XT gigabit ethernet Ports
Switches:	19 SMC Tiger SMC8624T 24-port managed gigabit switches 1U format

The head nodes are the same except that they have 2GB RAM and an extra 200GB hard drive housed in a 2U chassis (Chenbro RM21400).

Additional storage (beyond the disks on the individual nodes) is provided by a 3TB Arena RAID array. This unit has 16x200GB drives and is SCSI-attached to a master node.

Summary: In total, we have 46TB of distributed disk storage, 270GB RAM, 2km of cat6 network cables, 60m of Velcro to bundle cables together, and 40m of labeling tape to label the ends of all cables.

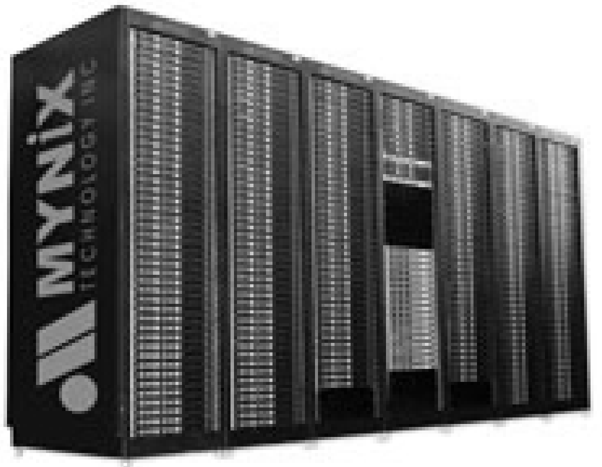


Figure 1. Mynix Technology Inc. PowerRACK-HX

Installation

Work on the cluster began a couple of weeks prior to delivery of the main hardware. We received some test nodes from Mynix, tested Linux kernel configurations and developed the installation procedure. Mynix delivered and installed the bulk of the hardware within 2 days and the machine was cabled and the Linux operating system was installed on each node by a small team of local experts in CITA and astronomy over the next few days.

The cluster was installed using OSCAR 2.0, an open source package designed for efficient Linux cluster installation[2]. We first installed a complete Redhat 7.3 distribution on the head node bob (a requirement for OSCAR) and ran the installation script. Building an oscar image was straightforward and node configuration and installation using PXElinux worked smoothly (after debugging the raid controller/netbooting BIOS conflict) in tests on single nodes but installing a cluster of our large size proved to be problematic.

It is important to know the identity of each node for maintenance as well as our networking topology. The only broadcast identifier is the MAC address unique to each network interface. Oscar does provide a MAC collection facility by listening on the network for DHCP requests. Unfortunately, the servers take a few minutes to reach the network booting stage in the BIOS and so simple interactive collection of MAC addresses and node installation one by one using the OSCAR MAC tool would have required more than 24 continuous hours of tedious work with human error playing a major factor.

We came up with a simple fast solution for installing and collecting MACs. We configured the dhcp server on the head node to assign IP numbers randomly to incoming requests to run the oscar PXElinux installation initrd ramdisk. In this way, large blocks of 10-20 nodes could be powered up simultaneously, netbooted and then oscar-installed through the head node. We tried larger number of power-ons but found that in practice that overloaded the head node so 10-20 was the practical limit. We signalled the completion of an installation by executing a script to initiate some hard drive activity and light up the red indicator light on the front of the node (Node shutdown + power down would have been preferable but did not work with our motherboard). In this way, the whole cluster was installed in a few hours with random IP number assignment.

At this stage, each node was fully installed and functional but the relation between the MAC/IP address and it's position on the racks was unknown. To determine each node's MAC address, we set up a simple network running through the second gigabit port and used it as a probe with a laptop point-to-point connection to determine the MAC addresses of the first port. MACs collected in this way took only a couple of hours. Once the table of MAC addresses and node positions on the rack were assembled a master /etc/dhcpd.conf file could be reset in some oscar database files using available scripts in the oscar packager. Nodes were then rebooted and assigned a sequential rather than a random IP number making easy identification on the racks.

We also double-checked this procedure by using the configurable SMC switch tools that report the probed MAC addresses on each switch port. This allowed us quickly to find a few misprobed nodes and establish the node identities on the racks unambiguously.

Networking

When considering options for networking the cluster, we discovered to our dismay that it can be very expensive. We decided against the proprietary low-latency networking options but learned that large-scale 200+ port switches were also very expensive and while cheaper than low-latency options would still consume a significant fraction of our budget. On the other hand, 24-port gigabit switches were very affordable and performed reasonably well according to specs so we considered networking schemes that could take advantage of this inexpensive hardware.

Fat-tree networks that link switches together through a hierarchy are a straightforward configuration that provide connectivity but relatively low cross-sectional bandwith. Each node comes with a second gigabit port, however, which could be exploited in some way through point to point connections with other machines. With this in mind we came up with the following scheme. Switches (or pairs of trunked switches) can be thought of as separate tightly coupled networks of a few dozen nodes. These nodes can be assigned to vertices in some more complex network topology. How can you connect these vertices to form a high-bandwidth global network linking all the compute nodes? After loading a switch with compute nodes there are few ports left to link to other vertices. However, each compute node provides a single connection through its second port that can in principle be connected to a compute node on an adjacent network vertex. If we enable routing capability in each compute node, network traffic can be relayed through the second port to other nodes. This was the strategy we employed in setting up a high-bandwidth global network.

The Fat-Tree Maintenance Network

Before setting up our high performance global network, we first set up a simple fat-tree that we treat as a robust but lower-performance maintenance network. This network was used to install the cluster originally. In our configuration, we trunk pairs of switches together making network vertices containing 48 ports with a total of 8 vertices using 16 switches. We first connect 32 compute nodes to each vertex. For 6 vertices, we run a 4-port trunk to a 17th master switch filling it to capacity. The fat-tree networking is completed by connecting the last two vertices with 4-port trunks to 2 vertices trunked to the master switch. The head nodes bob and doug, plus the development and spare nodes are all plugged into this network as well allowing direct communication between all available nodes.

This fat-tree network configuration is robust and provides connectivity to all the 256 nodes and is adequate for maintenance, installation and embarassingly parallel applications. However, the cross-sectional bandwidth is not at all ideal for heavy duty MPI applications. We now describe how we use this fat-tree network to bootstrap to our high-bandwidth network which runs predominantly through point to point connections between the second network ports of all machines.

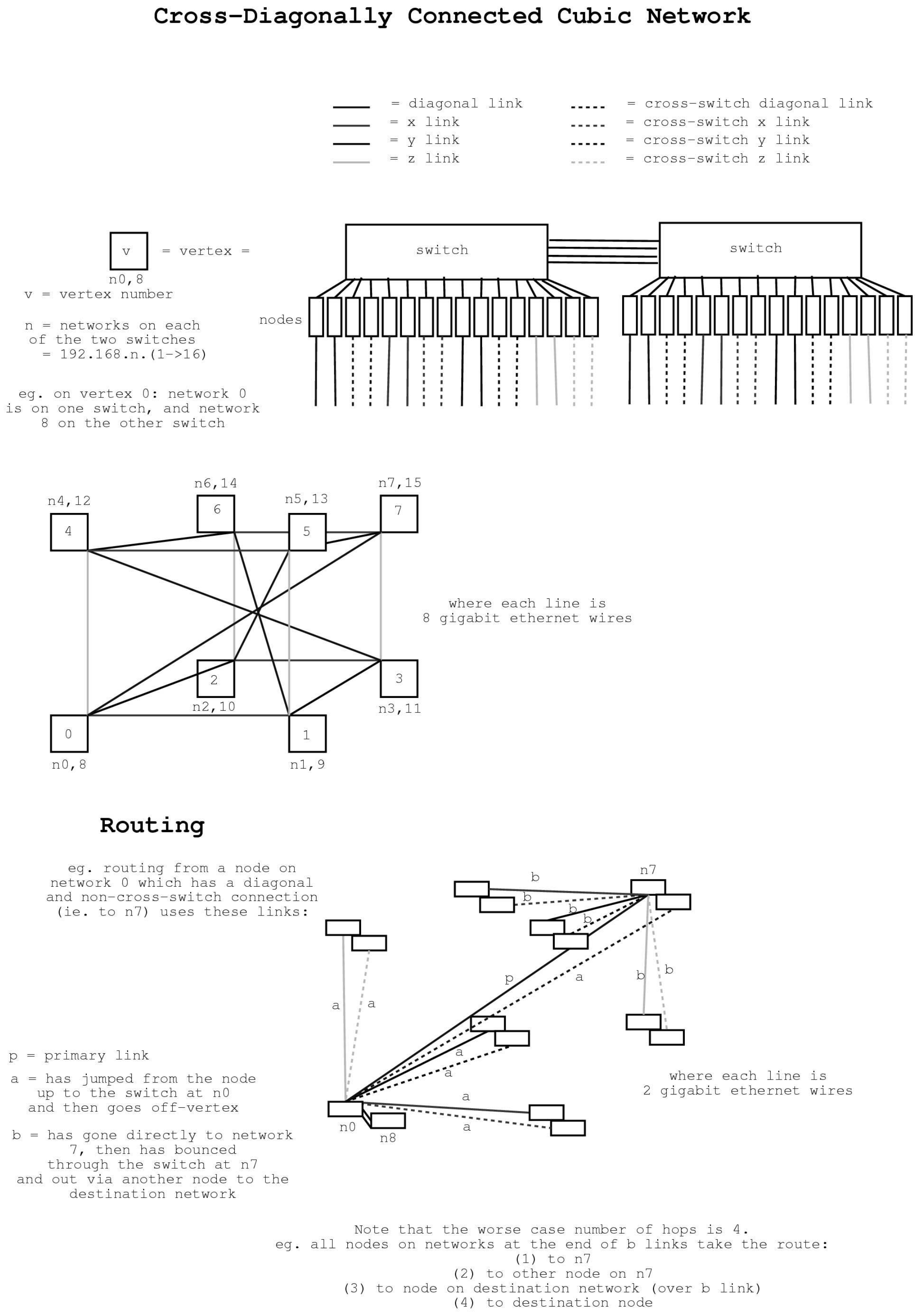
The Cross-Diagonally Connected Cube Network

The network vertices described above can be thought of as independent networks that we can connect to each other according to some topology. The use of a master switch to build a fat tree is one such topology but it has fairly low cross-sectional bandwidth. How can we use the second port on each compute node to connect these vertices to increase the network bandwidth and minimize network hops to reduce latency? There are many possibilities but we finally settled on a cubic network topology with a twist (see Figure 2.)

In our chosen topology, the 8 network vertices are arranged on the corners of a cube. Communication between vertices can occur along the edges of the cube but we also connect opposite corners through diagonals that cross through the cube centroid. We call this topology a Cross-Diagonal Connected Cube (CDCC) (although we're sure it has its own name in the computer science literature). Each vertex then has 4 outgoing lines of communication to its adjacent corners and opposite along the diagonal.

The CDCC topology requires hard-wired routing tables between nodes as described below. We have 32 gigabit lines on each vertex provided by the second network port on each compute node and 4 vertex connections. We therefore assign 8-gigabit lines to each vertex connection. These lines connect 8 compute nodes on one vertex to another 8 on a neighbouring vertex through direct port to port connections with a straight-thru cable. In this way, there is an 8-gigabit pipeline connecting each vertex which must be shared between 32 compute nodes. Also, the 4-port trunks connecting pairs of switches are 4-gigabit pipelines relaying traffic for 16 compute nodes per switch.

In this way, the CDCC topology with 256 nodes approximately represents a fully-switched network running at 250 Mbit, although it may be slightly better since nodes on the same switch are fully-switched at 1 Gbit. When constructing the routing tables, the maximum number of network hops to get from one node to another is 4 so latency can be greater than a fully-switched system as well. The performance of this configuration is about half the 500 Mbit performance that is expected for large gigabit switches that typically connect 16 ports to an 8-gigabit backplane internally. Latency is also about 4 times greater. However, the total cost for the 17 required 24-port switches plus cabling is currently about 20% the cost of a large switch. We show below that our cluster provides competitive benchmark speeds for similar clusters with better networking and performs well for our applications.


Cross-Diagonally Connected Cubic Network

v = vertex number

n = networks on each
of the two switches
= 192.168.n.(1->16)

eg. on vertex 0: network 0
is on one switch, and network
8 on the other switch

where each line is
8 gigabit ethernet wires

Routing

eg. routing from a node on
network 0 which has a diagonal
and non-cross-switch connection
(ie. to n7) uses these links:

p = primary link

a = has jumped from the node
up to the switch at n0
and then goes off-vertex

b = has gone directly to network
7, then has bounced
through the switch at n7
and out via another node to the
destination network

where each line is
2 gigabit ethernet wires

Note that the worse case number of hops is 4.
eg. all nodes on networks at the end of b links take the route:
(1) to n7
(2) to other node on n7
(3) to node on destination network (over b link)
(4) to destination node

Figure 2. Cross-Diagonally Connected Cubic Networking Topology, Cabling and Routing Scheme

Cabling

The task of cabling the cluster also proved to be formidable but we came up with some ways to minimize confusion and tangling. Connections from the first ethernet port to the switches is straightforward if the switches are centralized to one location at the centre of the rack array in a stack. Network cables are run over and under the cluster in small bundles and colour coded by rack number and then plugged into their appropriate ports in the central stack of switches. Although tedious, this stage of the cabling took a small team of people including Peter Martin's sons and their girlfriends about two days to complete. We point out that the same cabling effort is required even for more expensive multi-port switches so there is no significant overhead in manual labour.

In the second stage, it was necessary to wire the CDCC network using the point to point connections through the second network port on each node. Wiring these connections could potentially lead to a tangled mess of cables if the nodes are ordered sequentially according to their vertex numbers. A simple solution to this wiring problem is to re-order the nodes on the racks in pairs that are to be connected to one another. In this way, only a short 1 foot length of cable is required and wiring can be done quickly. The logic of the connectivity of the CDCC network was used to create the correct ordered list of nodes on the rack for this arrangement as well as a wiring diagram for the connecting cables to the right switches in the fat-tree diagram. The only extra level of complexity for doing things this way in comparison to 200+ port monster switches is the node ordering and the need to make sure each cable is connected to the correct port in the switch stack. In practice, we have re-wired the network a couple of times to test the performance of different topologies and it generally only takes a few hours to do.

Routing

Another essential (and complex) part of getting the CDCC network to perform was the configuration of the routing tables for each Linux server. Network packets relayed through each server on this server have to know where to go according to our defined network topology.

A nice part of our design is that we have a back-up maintenance network (the fat-tree) that we can always use to gain access to each node in case of routing problems or broken links in the CDCC network. This allowed us to experiment extensively with the scripts for generating the routing tables of the CDCC and led to an optimized system. Our strategy is to generate purely static routing tables on each server with a simple script at startup. Each of these scripts contains routing commands to establish about 50 static routes to individual nodes and sub-networks (vertices) in the CDCC network. We find this system works very well in practice and can be taken up and down quickly if necessary.

Our arrangement is also very fault-tolerant in case of node failures or broken links. If one or more nodes fails in the system, it will create holes in the CDCC network but as a contingency we can route around the broken nodes simply by using the fat-tree network. We have written a simple network repair script that pings all neighbours on the CDCC - if a node fails to answer we attempt to establish a direct route to it through the fat-tree - if that fails the node is assumed to be dead. In this way, we can continue to run jobs on the cluster even when main compute nodes go down. Either the hot spares, or development nodes can fill in while we replace the failed nodes.

Figure 2. shows some final details of our routing scheme that are worth pointing out. It turned out that the network performance of the trunks was not as good as we expected. We therefore modified our routing scheme to avoid using the trunks as much as possible. Network traffic from one node to another on different vertices used cross links that avoid the trunks. The only network traffic going through the trunks is between the nodes on bonded pairs of switches.

Benchmarks

We ran the High-performance Linpack Benchmark (HPL), a portable implementation for Distributed-Memory Computers to measure the speed of our cluster and compare with others on the Top 500 list [1]. This code uses the Message Passing Interface (MPI) libraries. The actual benchmark involves the inversion of the largest matrix that can be stored on your cluster. In our case, the matrix size was 160000 X 160000 elements. We compiled the code using the Intel C compiler icc version 7.0 and linked to the Intel MKL math libraries and Kazushige Goto's optimized BLAS libraries [3]. We used the LAM version 6.6b1 for the as the MPI library. The HPL output from our fastest benchmark is shown in Table 2. We achieved a speed of 1.11 Teraflops about 45% the peak speed of 2.46 Teraflops. This ratio of real to peak speed is comparable to many of the machines quoted on the November 2002 Top 500 list including similar sized clusters running with proprietary networking such as Myrinet. Our ranking on the list would currently be number 41 making us the fastest quoted supercomputer in Canada. Of course, it remains to be seen how we will rank on the next list released in a few months since these trends tend to evolve very quickly!

Table 2. HPL logfile output for our 512 processor benchmark. The measured speed is 1113 Gflops or 1.11 Teraflops.

```
==================================================
T/V            N          NB    P    Q    Time          Gflops
---------------------------------------------------------------------------
W01R2L4      160000   200    8   64    2452.90        1.113e+03
---------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =       0.0043290 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =       0.0023799 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =       0.0004091 ...... PASSED
```

SCIENCE

At the time of writing this paper, we have just begun some scientific production runs using the cluster. These include a simulation of the collision of two galaxies each containing central supermassive blackholes and magnetohydrodynamic (MHD) simulations of the accretion of gas onto blackhole.

The galaxy collision is a pure N-body simulation using more than 300 million particles integrated for about 10000 timesteps representing a physical time of several billion years. The goal is learn about the process of the formation of central blackhole binaries and their effect on the internal structure of galaxies. Figure 3. is a snapshot from the simulation showing the galaxies shortly after collision. While the current results are still preliminary, the ability to run simulations at this spatial and time resolution will be revolutionary in helping our understanding of the dynamics and evolution of galaxies.

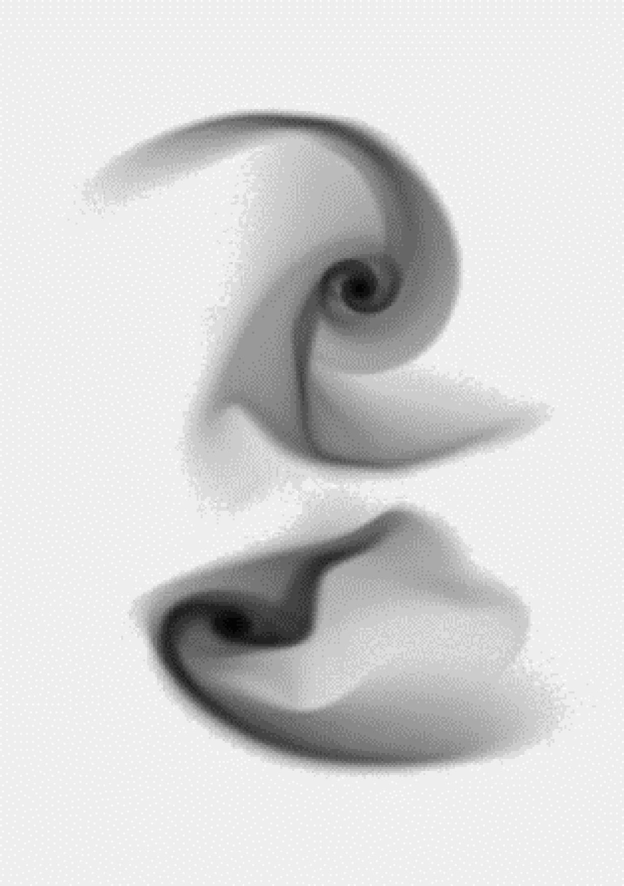


Figure 3. A snapshot of the collision of two spiral galaxies using 307.2 million particles computed on the CITA Beowulf Cluster McKenzie

References

[1] Top 500 Supercomputers website, http://www.top500.org

[2] The OSCAR cluster installation package, http://oscar.sourceforge.net

[3] Kazushige Goto's Optimized BLAS Library, http://www.cs.utexas.edu/users/flame/goto/