

Building a Bigger Beowulf

CITA
Canadian Institute for
Theoretical Astrophysics



McKenzie: A Teraflops Linux Beowulf Cluster For Computational Astrophysics

Robin Humble, John Dubinski, Ue-Li Pen, Chris Loken, Peter Martin

1uv 2003

SWIN
BUR
NE

CENTRE FOR
ASTROPHYSICS AND
SUPERCOMPUTING

Building a Bigger Beowulf

CITA
Canadian Institute for
Theoretical Astrophysics



- Outline
 - Why a large Beowulf?
 - Network Interconnects Overview
 - The Machine
 - Installation
 - Routing Design
 - Pretty Pics
 - 2nd Generation Routing
 - Future Machines
 - Conclusions



- Let's spend Can\$900k (~ Aus\$1M)
- Already have a large expensive SMP machine
 - 32-cpu Alpha 731MHz with 64G RAM (~\$3M)
 - for OpenMP (threads) parallel codes, or serial jobs that need large memory
- Beowulf cluster is a good complementary technology
 - many “normal” machines connected by a fast network
 - for MPI parallel codes (separate processes talking via messages), or many small-ish memory serial jobs



- Myrinet, Quadrics, Infiniband, SGI's interconnect, ...
 - low latency: few to 10 μ s
 - high bandwidth: 200+MB/s
 - expensive! ~\$1000/port
 - doesn't scale well past ~few hundred to 1000 ports
- Large Gigabit Ethernet Switch (Foundry, Cisco, ...)
 - high latency: 50 to 100 μ s
 - good bandwidth: 100MB/s
 - but almost as expensive!
 - only up to ~128 ports

The Problem : Overview



- These “exotic” interconnects mean up to 50% of the cost of the machine is in the networking
 - Noooo! I want a more nodes...
- None of the expensive networking options scale well to 1000+ node machines
 - typically use a fat tree which limits bandwidth
- Serial (non-parallel) jobs don't use the fast network at all

The Solution : Overview



- Server machines typically come with 2 gigabit ethernet ports
- Use both of them!
- Connect machine with a stack of cheap 24-port commodity gigabit ethernet switches

The Nodes

CITA
Canadian Institute for
Theoretical Astrophysics



The Nodes (ctd.)



- Built by Mynix Technologies, Montréal, Quebec (HPC division is now part of Ciara, Montréal)
- 268 dual CPU servers running RedHat Linux 7.3
 - two 2.4GHz Xeons (with HyperThreading disabled)
 - Westville (SE7500WV2ATA) motherboard
 - dual onboard gigabit (Intel e1000) ethernet ports
 - includes remote-management over LAN (unused)
 - 1GB RAM
 - two 80GB Maxtor IDE drives

The Nodes (ctd.)



- Compute nodes (eh1 to eh256) total
 - 512 cpus, 256G RAM, 35TB (usable) scratch disk
- 2 front-end nodes (bob and doug) are the same except additional
 - 1G RAM, 200G Maxtor IDE disk, SCSI card
 - 2.2TB (usable) SCSI-attached IDE RAID-5 (Zentra)
- 8 extra development nodes thrown in by Intel (thanks!)
- 2 spare nodes

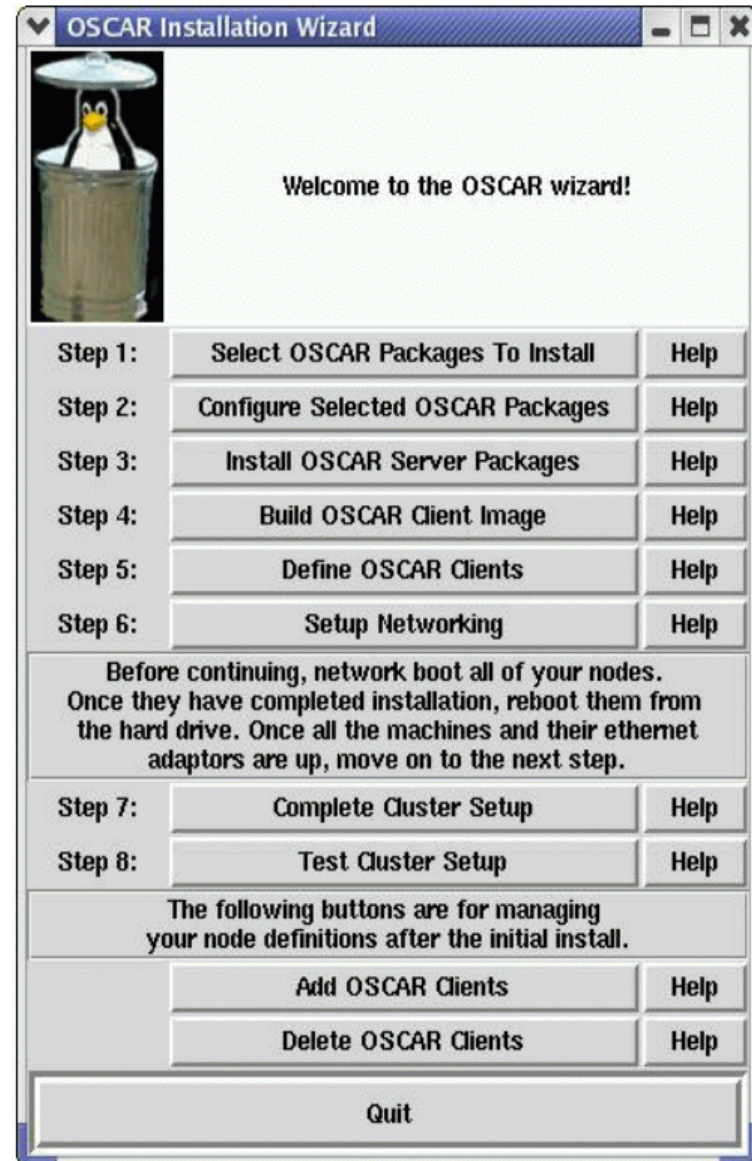


- Aug '02 – Begin in-house discussions
- Sep-Oct – Vendor discussions
- 1 Nov – Selected Mynix technologies
- Nov-Dec – assembly and burn-in in Montréal
- 19 Dec – boxes arrive in Toronto
- 21-22 Dec – Cabled network connections
- 23 Dec – Test simulations start (Xmas? Nah...)
- Jan '03 – debugging, tweaking and science
(Sep '03 – Redesigned network!)

Installation – OSCAR



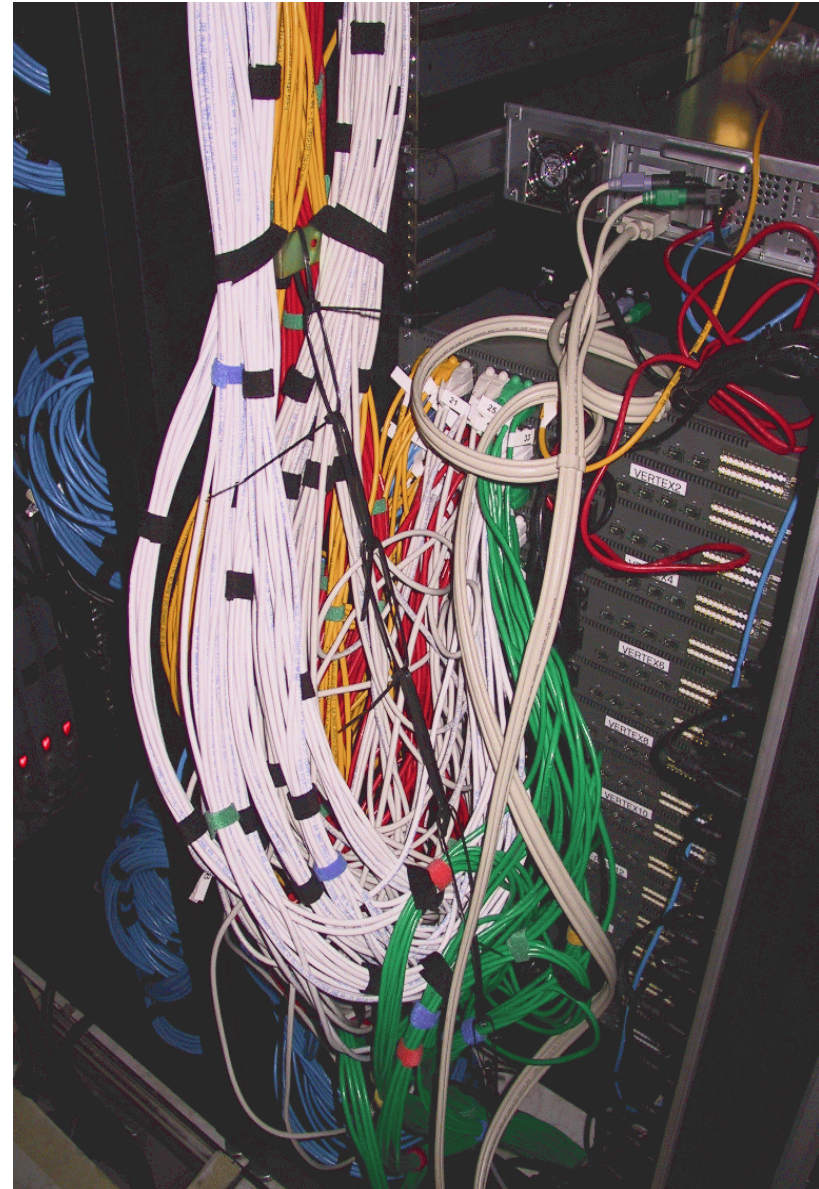
- PXE boot (dhcp + tftp + pxelinux)
 - downloads and boots kernels (+ optional ramdisk) over the network
 - #@!\$@!!@\$ Promise IDE RAID
 - nodes either network or CDROM boot (head nodes boot off CDROM!)
- ~15 mins to boot, partition, build soft RAID, format, and install 32 nodes
- based on “rsync” and a directory containing a RedHat 7.3 install
- easy commands eg.
cexec :33-64 reboot
cpushimage :1-16 oscarimage1



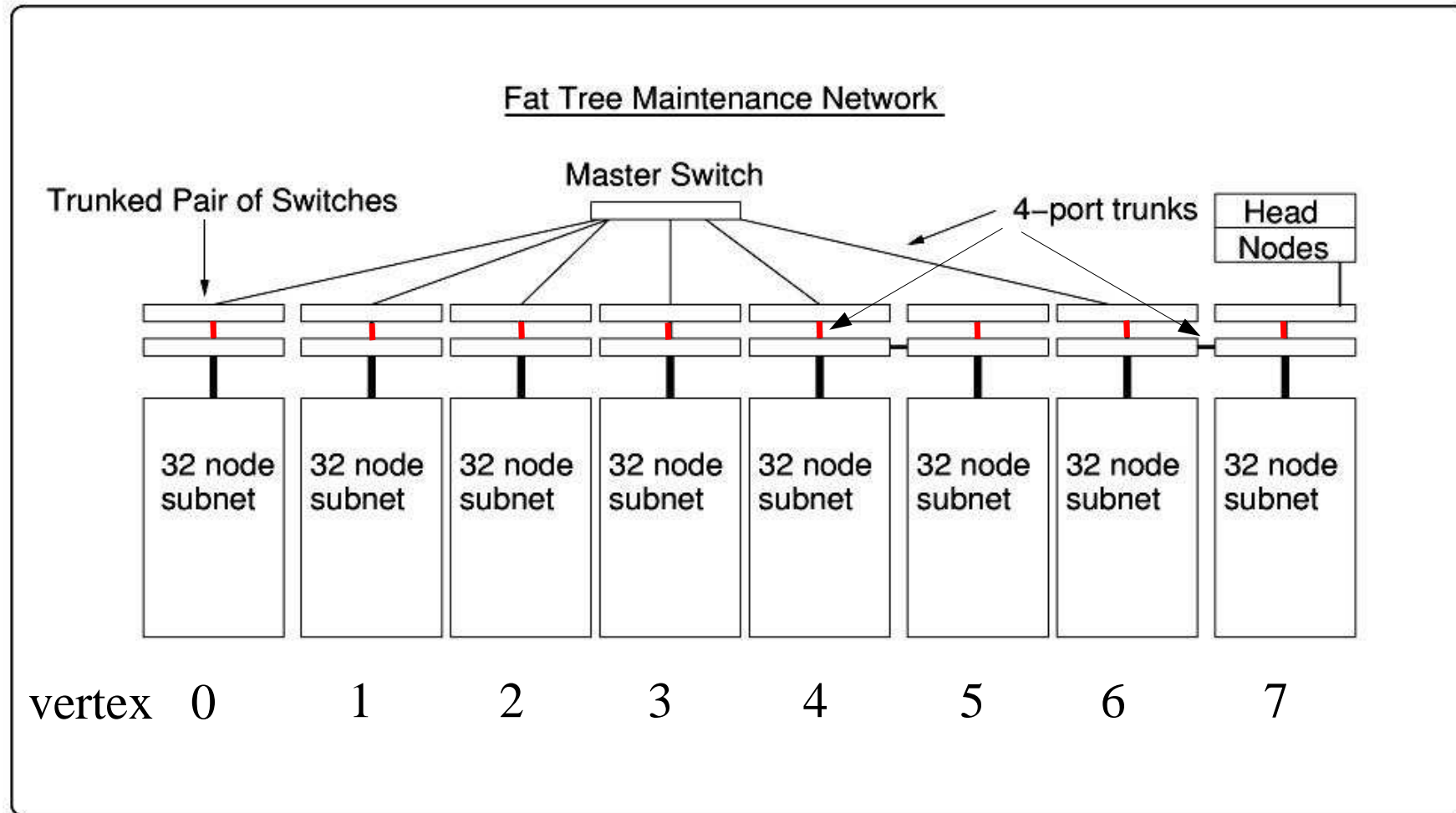
Routing Overview



- Use both of the gigabit ethernet NICs on each node
- ALL nodes are Routers
- Use seventeen 24 port SMC gigabit switches (SMC8624T TigerSwitch)
- 2 logically different networks
 - slow fat tree “maintenance” network
 - high bandwidth communications network



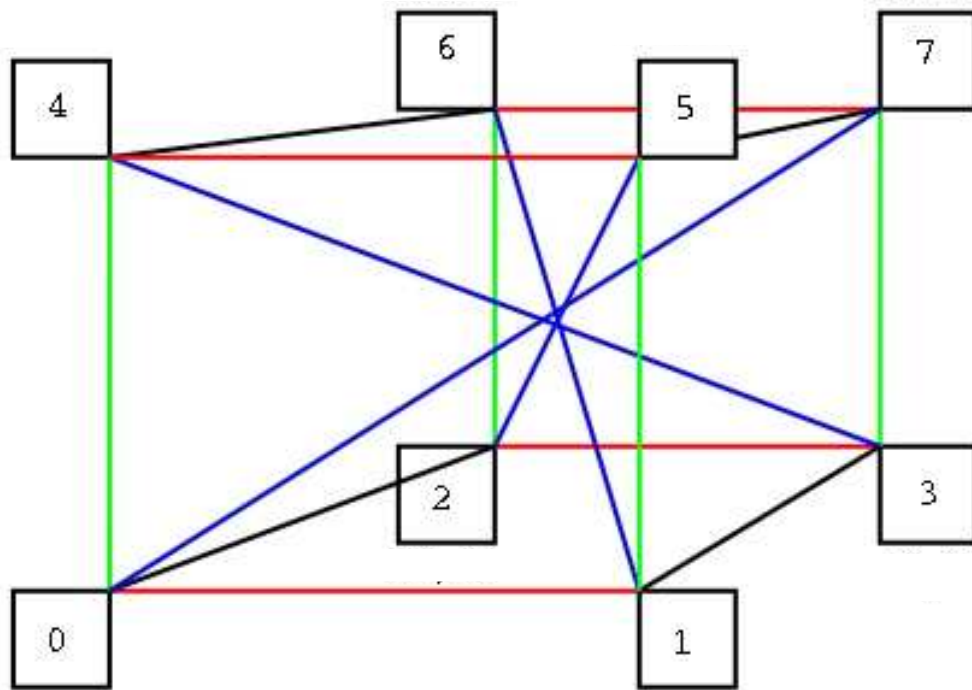
Routing – Slow Network



— are 'trunks' between 2 switches on a vertex – making a (low bandwidth) ~48 port switch



Cross-Diagonally Connected Cube (CDCC)



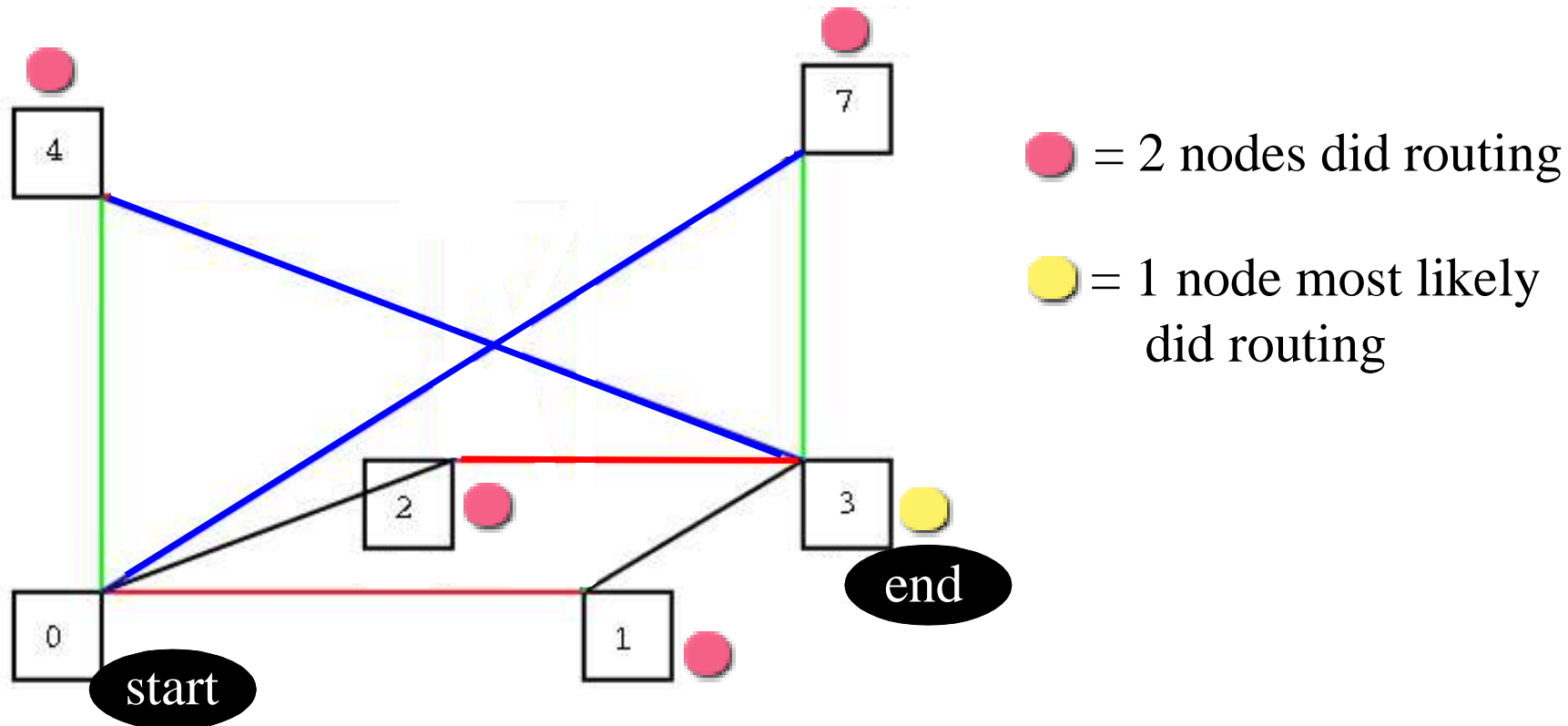
x wires are red
y wires are black
z wires are green
diagonal wires are blue

- Vertex consists of 32 nodes on 2 switches
- Each coloured line is 8 node-to-node wires, 16Gbit total

Routing – CDCC Example



eg. Route from a node on vertex 0 to a node on vertex 3

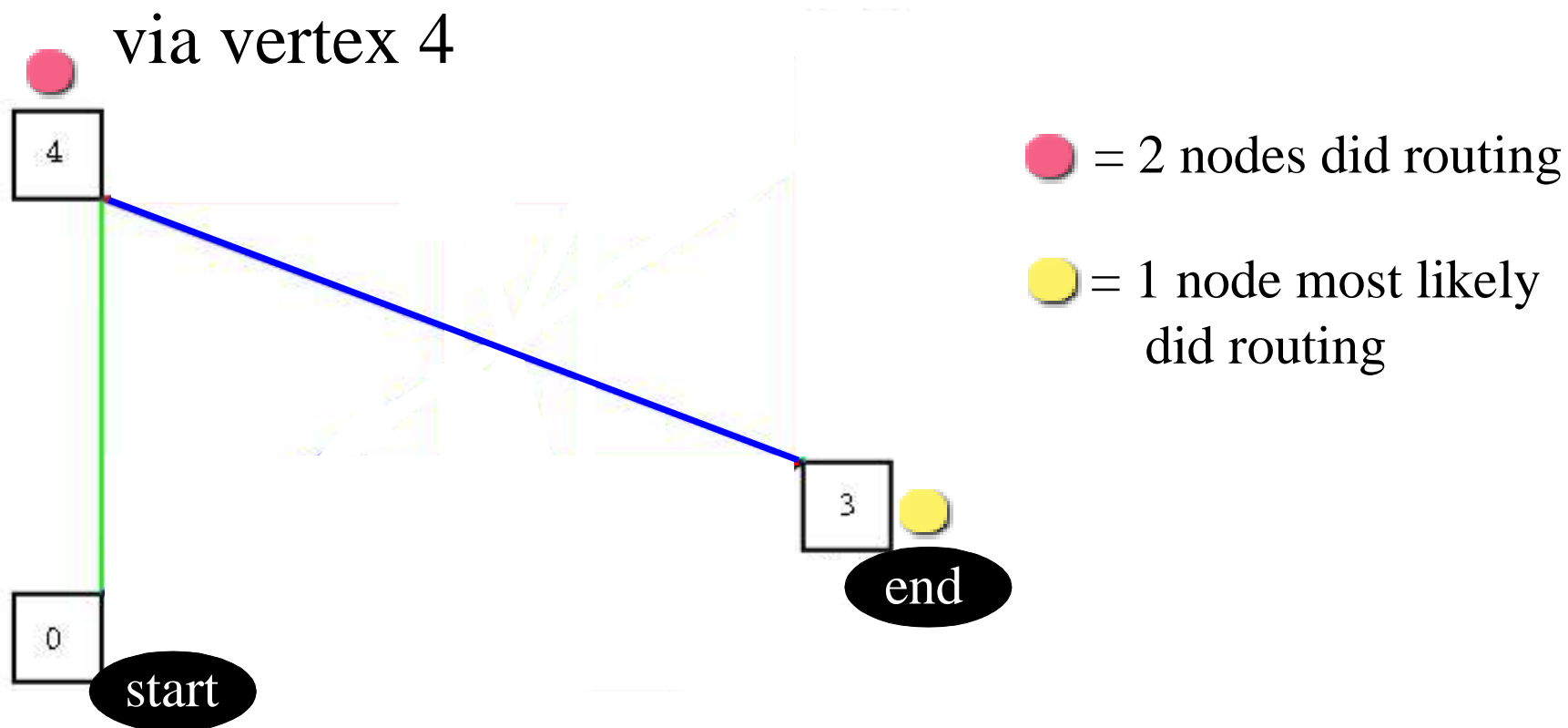


- 4 different routes are possible depending upon whether the starting node is **x**, **y**, **z**, or **diag** connected

Routing – CDCC Example



eg. Route from a node on vertex 0 to a node on vertex 3
via vertex 4



- Traffic goes via 3 nodes that do routing



- Summary
 - Every node routes traffic for other nodes
 - Unique routing table for each node
 - Generated from python scripts that are aware of the geometry and connectivity of the machine
 - Worst case is going through 3 routing nodes on the way from one node to another
 - Most traffic does go through multiple nodes
- What does the routing table (for eh1) look like?

Routing Table for eh1



```
#!/bin/sh
# node eh1, with eth1 ip 192.168.0.1, has a 'diag' and straight-across link
route add -net 192.168.7.0 netmask 255.255.255.0 gw 192.168.7.1 # these go straight out eth1 to eh2
route add -net 192.168.6.0 netmask 255.255.255.0 gw 192.168.7.1
route add -net 192.168.5.0 netmask 255.255.255.0 gw 192.168.7.1
route add -net 192.168.3.0 netmask 255.255.255.0 gw 192.168.7.1
route add -net 192.168.14.0 netmask 255.255.255.0 gw 192.168.7.1 # these go in the same direction, but will swap sides later
route add -net 192.168.13.0 netmask 255.255.255.0 gw 192.168.7.1
route add -net 192.168.11.0 netmask 255.255.255.0 gw 192.168.7.1
route add -net 192.168.15.0 netmask 255.255.255.0 gw 192.168.7.1 # littleMore mode, so let the other vertex change sides for us
# none of these go in our direction, so send them (round-robin) via an appropriate node on our switch:
route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.100.65 # eh65 (= 192.168.0.5)
route add -net 192.168.9.0 netmask 255.255.255.0 gw 192.168.100.97 # eh97 (= 192.168.0.7)
route add -net 192.168.2.0 netmask 255.255.255.0 gw 192.168.100.129 # eh129 (= 192.168.0.9)
route add -net 192.168.10.0 netmask 255.255.255.0 gw 192.168.100.161 # eh161 (= 192.168.0.11)
route add -net 192.168.4.0 netmask 255.255.255.0 gw 192.168.100.193 # eh193 (= 192.168.0.13)
route add -net 192.168.12.0 netmask 255.255.255.0 gw 192.168.100.225 # eh225 (= 192.168.0.15)
# exceptions/optimisations to the above networks for machines connected to eth1 on this switch
route add -host 192.168.1.6 gw 192.168.100.81 # eh81 (= 192.168.0.6)
route add -host 192.168.9.8 gw 192.168.100.113 # eh113 (= 192.168.0.8)
route add -host 192.168.2.10 gw 192.168.100.145 # eh145 (= 192.168.0.10)
route add -host 192.168.10.12 gw 192.168.100.177 # eh177 (= 192.168.0.12)
route add -host 192.168.4.14 gw 192.168.100.209 # eh209 (= 192.168.0.14)
route add -host 192.168.12.16 gw 192.168.100.241 # eh241 (= 192.168.0.16)
# nodes on our vertex:
# route add -host 192.168.0.1 gw 192.168.100.1 # eh1
route add -host 192.168.0.2 gw 192.168.100.17 # eh17
route add -host 192.168.0.3 gw 192.168.100.33 # eh33
route add -host 192.168.0.4 gw 192.168.100.49 # eh49
route add -host 192.168.0.5 gw 192.168.100.65 # eh65
route add -host 192.168.0.6 gw 192.168.100.81 # eh81
route add -host 192.168.0.7 gw 192.168.100.97 # eh97
route add -host 192.168.0.8 gw 192.168.100.113 # eh113
route add -host 192.168.0.9 gw 192.168.100.129 # eh129
route add -host 192.168.0.10 gw 192.168.100.145 # eh145
route add -host 192.168.0.11 gw 192.168.100.161 # eh161
route add -host 192.168.0.12 gw 192.168.100.177 # eh177
route add -host 192.168.0.13 gw 192.168.100.193 # eh193
route add -host 192.168.0.14 gw 192.168.100.209 # eh209
route add -host 192.168.0.15 gw 192.168.100.225 # eh225
route add -host 192.168.0.16 gw 192.168.100.241 # eh241
route add -host 192.168.8.1 gw 192.168.100.9 # eh9
route add -host 192.168.8.2 gw 192.168.100.25 # eh25
route add -host 192.168.8.3 gw 192.168.100.41 # eh41
route add -host 192.168.8.4 gw 192.168.100.57 # eh57
route add -host 192.168.8.5 gw 192.168.100.73 # eh73
route add -host 192.168.8.6 gw 192.168.100.89 # eh89
route add -host 192.168.8.7 gw 192.168.100.105 # eh105
route add -host 192.168.8.8 gw 192.168.100.121 # eh121
route add -host 192.168.8.9 gw 192.168.100.137 # eh137
route add -host 192.168.8.10 gw 192.168.100.153 # eh153
route add -host 192.168.8.11 gw 192.168.100.169 # eh169
route add -host 192.168.8.12 gw 192.168.100.185 # eh185
route add -host 192.168.8.13 gw 192.168.100.201 # eh201
route add -host 192.168.8.14 gw 192.168.100.217 # eh217
route add -host 192.168.8.15 gw 192.168.100.233 # eh233
route add -host 192.168.8.16 gw 192.168.100.249 # eh249
```

} routes out our wire to other vertices

} routes to other vertices via nodes on our vertex (they involve less hops than us sending directly)

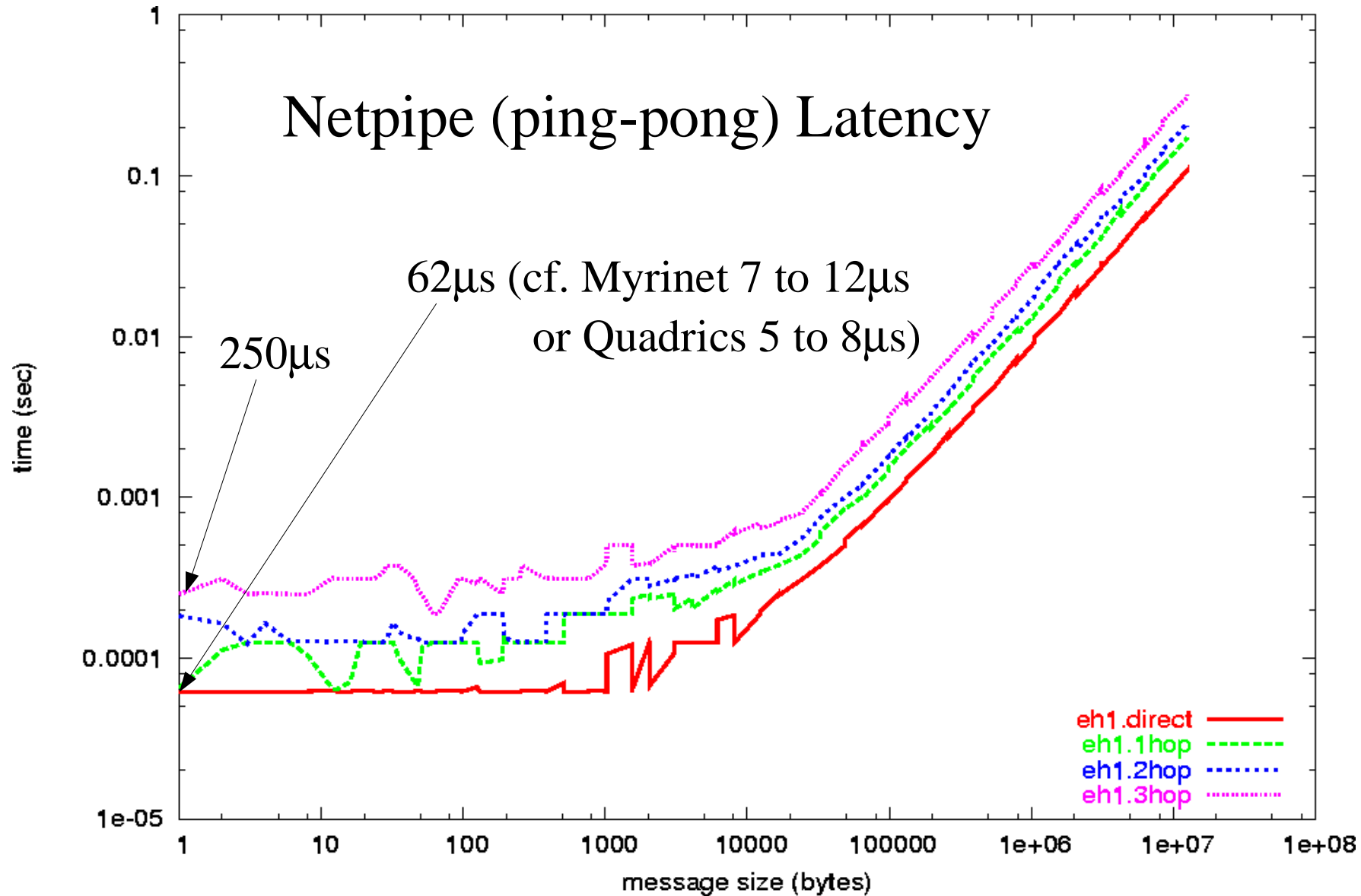
} tweaks

} routes to nodes on own vertex

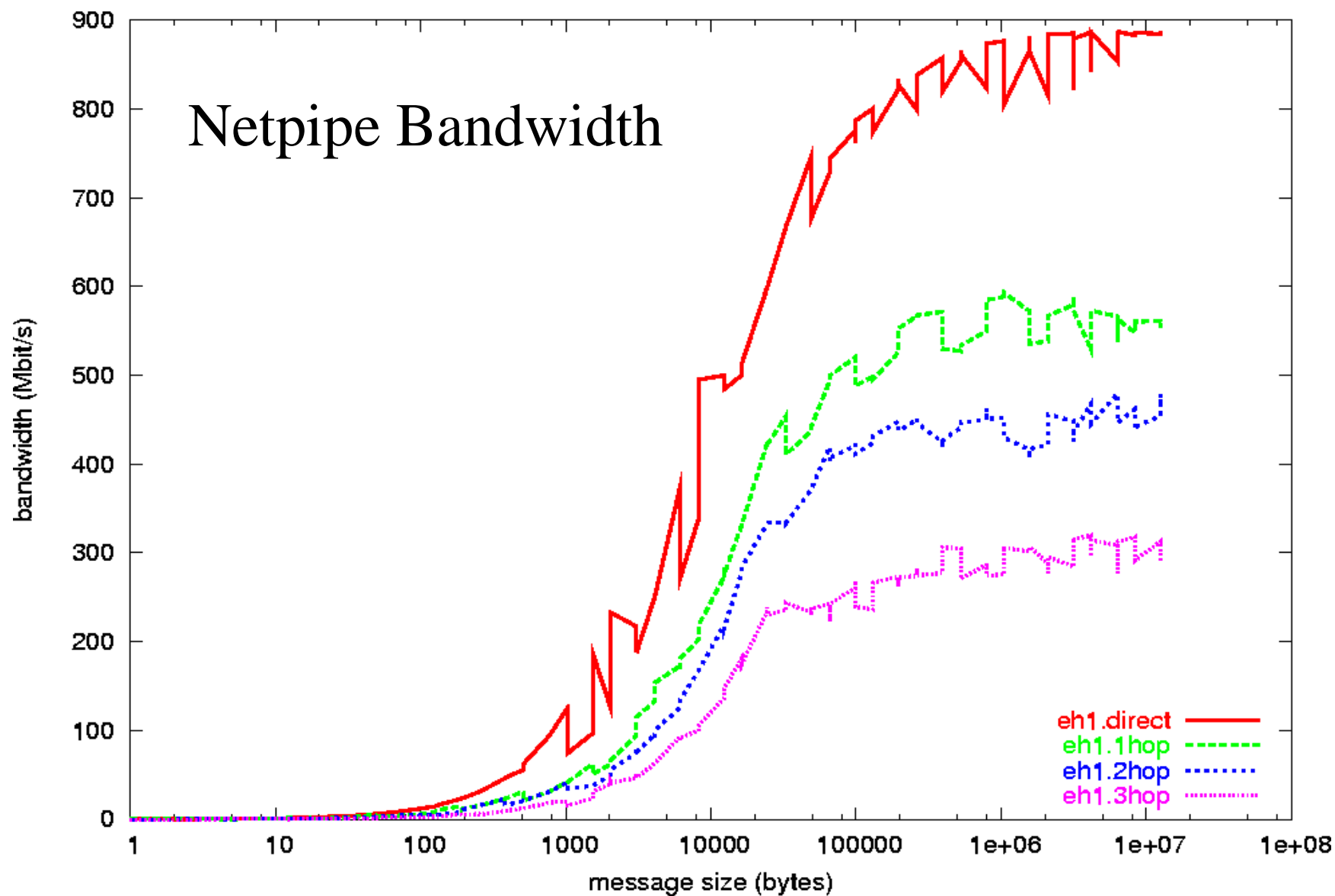


- Actually we don't route on a “vertex” basis
- “Trunks” between the 2 switches on a vertex are a bandwidth bottleneck
 - $8*4*2 = 64\text{Gbit/s}$ for Trunks
 - $8*8*2 = 128\text{Gbit/s}$ for **x,y,z,diag** wires
- So routing is actually done on a per switch basis to (mostly) avoid the “trunk” and keep 128Gbit/s
- Wires travel from each switch not just **x,y,z,diag** but:
 - **x,y,z,diag** to “left hand” switches on vertices
 - **x,y,z,diag** to “right hand” switches on vertices

Routing – Performance



Routing – Performance

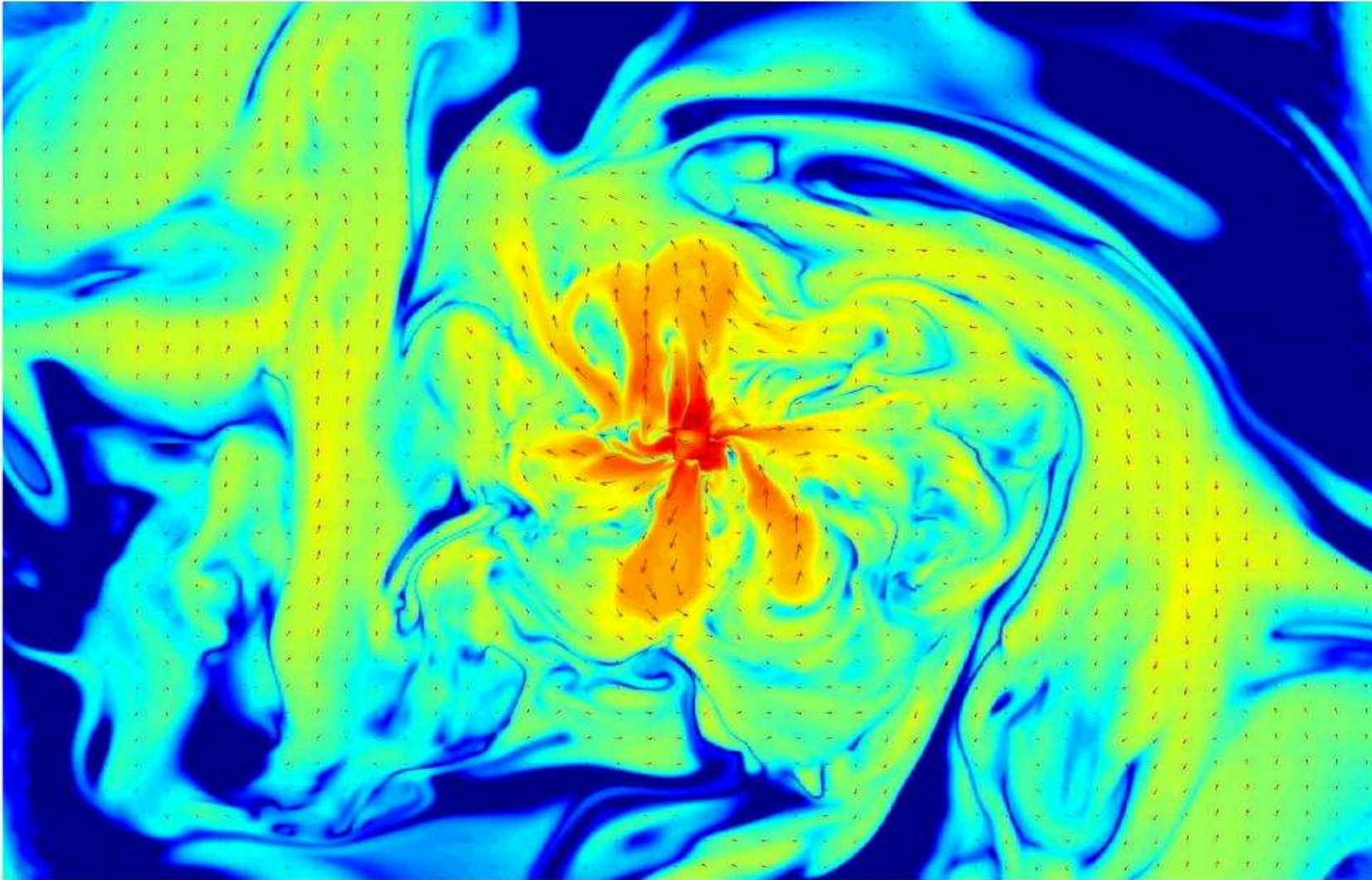




- Routing bandwidth drops steeply through each node
 - I really don't understand this...
 - Looks to be independent of size and number of packets
 - Not a driver performance problem?
 - Kernel needs work?
 - Dual Xeon to blame?
- Latency goes up linearly with the number of nodes
 - As expected
- It all works!

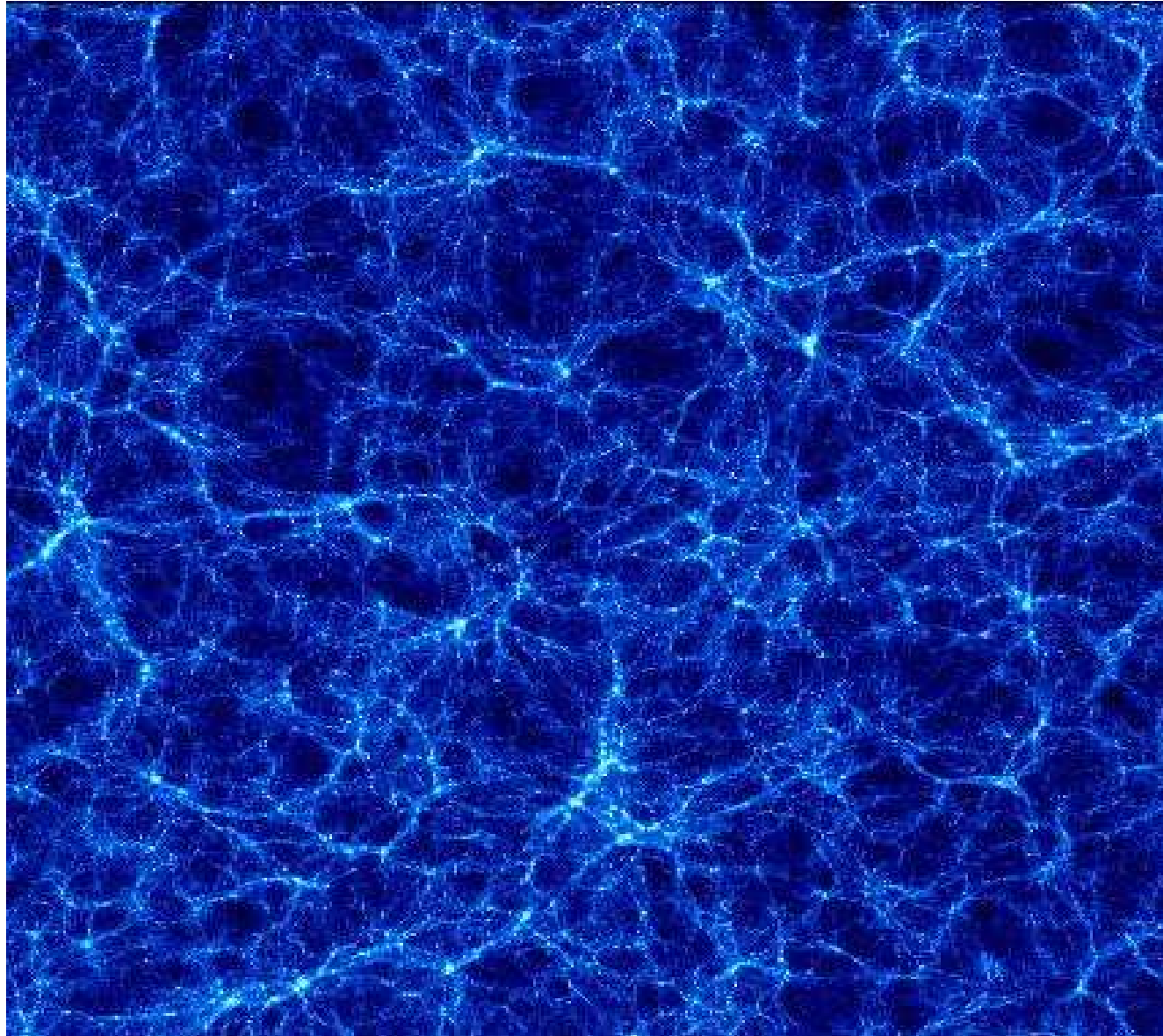
Pretty Pics

CITA
Canadian Institute for
Theoretical Astrophysics



MHD Flow onto a Blackhole Pen, Matzner, Wong 2003

Pretty Pics (ctd.)



Cosmology
Simulation

GOTPM

1536^3 mesh=3.6B

$N=768^3=452M$

Dubinski, Kim,
Park, Humble 2003

Junk CDCC – go Mesh!

CITA
Canadian Institute for
Theoretical Astrophysics

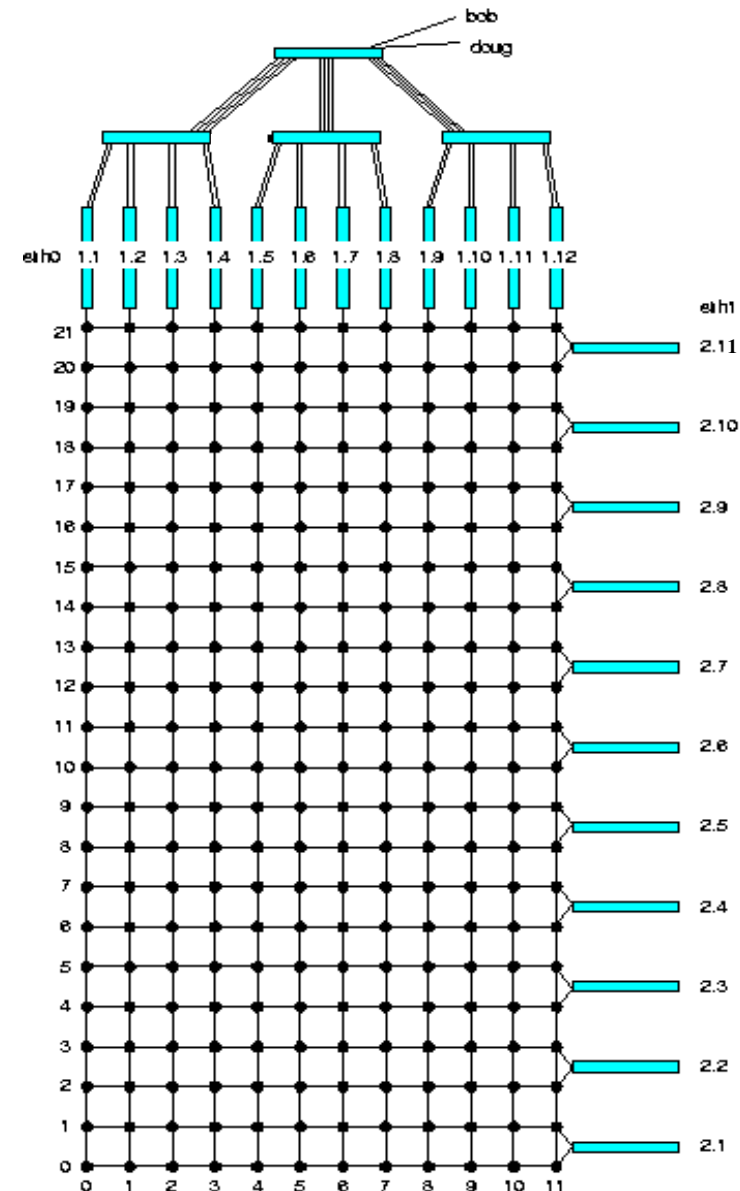


- Sep '03 – Start again!
 - Almost, but not really...

Routing – Mesh



- Same basic idea – all nodes are routers
 - Every node connected to an x-switch and a y-switch
- 7 extra 24 port switches from SMC (thanks!)
 - 27 switches total (incl a few spares we had from before)
 - Include devel nodes in mesh
- Maximum of one hop through a router node



Mesh Performance



- Node to node bandwidth and latency are the same as direct and 1-hop from CDCC scheme
- Not sure what the Cross-Sectional Bandwidth number is!
- top500.org (Linpack) scores:

	Processors	Rmax	% of Peak	Rank
CDCC	512	1.202 TFLOPS	49	#38
Mesh	528	1.455 TFLOPS	57	#34

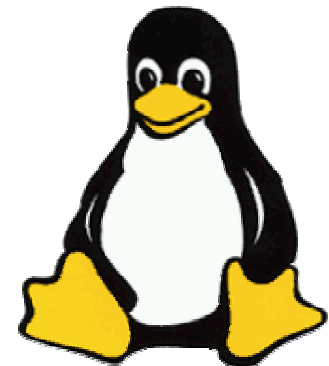


- The Universe Simulator. eg.
 - 91 48-port switches, 45×46 mesh = 2070 nodes
 - 4-way Opteron nodes
 - ⇒ 8280 processors
- Want bigger? Can add a 3rd NIC and use the z-dimension
 - then it's 2-hops between nodes :-)
- Mesh (and CDCC) work for other network fabrics
 - Could do the same with Myrinet, or 10Gbit ethernet

Conclusions



- Teraflop is achievable with commodity networking and modest budget
- CDCC and Mesh are not for everyone but work well for latency tolerant parallel algorithms
- Networking (switches and cables) ~6% of total cost
- Scalable to much larger designs
- Linux rocks! (but routing and HT scheduler could be improved a tad)
- “Fastest” Computer in Canada, and faster than any in Australia too :-)



References



McKenzie – www.cita.utoronto.ca/webpages/mckenzie/

OSCAR – oscar.sourceforge.net

Netpipe – www.scl.ameslab.gov/netpipe/

Goto's BLAS - www.cs.utexas.edu/users/flame/goto/

LAM MPI - lam-mpi.org

Top500 – top500.org

Thanks to:



Canadian Foundation for Innovation, Ontario Innovation Trust, SMC, Intel, Centre for Astrophysics and Computing - Swinburne University, Craig West

Fin

CITA
Canadian Institute for
Theoretical Astrophysics

