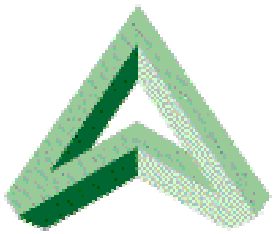
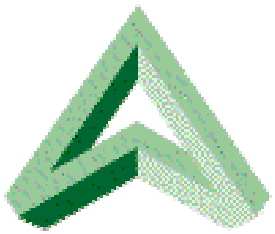


Hybrid Programming Fun:
Making Bzip2 Parallel with MPICH2 & pthreads on the Cray XD1

Charles Wright
HPC Systems Administrator @
Alabama Supercomputer Authority
http://www.asc.edu



bzip2

Lossless data compression program and library

Can achieve very **high compression ratios** (50+:1)

Requires a lot of CPU time

- AMD Opteron 2.2 Ghz CPU processes about 4.8 GB/hour uncompressed

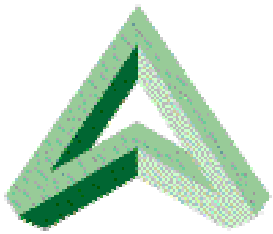
Sample Application

Permanent **offline backups** for disaster recovery

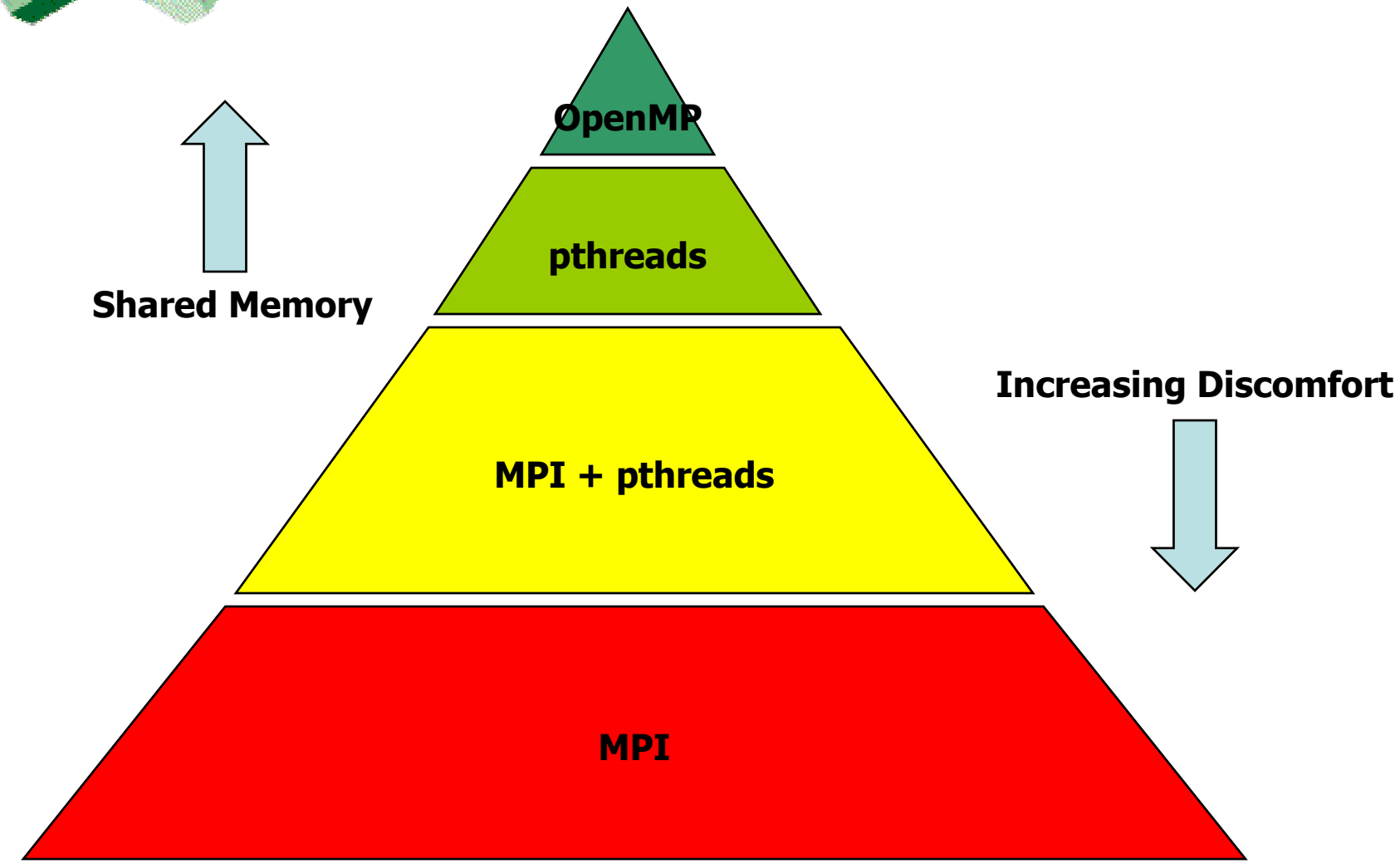
Amount of files that I would like to backup on ASN's XD1

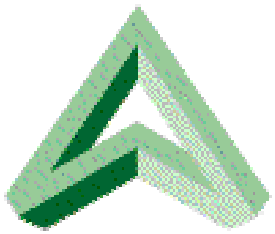
```
   168G /opt/asn  (Applications)
+ 261G /home    (Home Directories)
   ----------------
   429 Gigabytes
```

Serial bzip2 would take about **4 days to compress**



Hierarchy of Pain





Why Hybrid Programming?

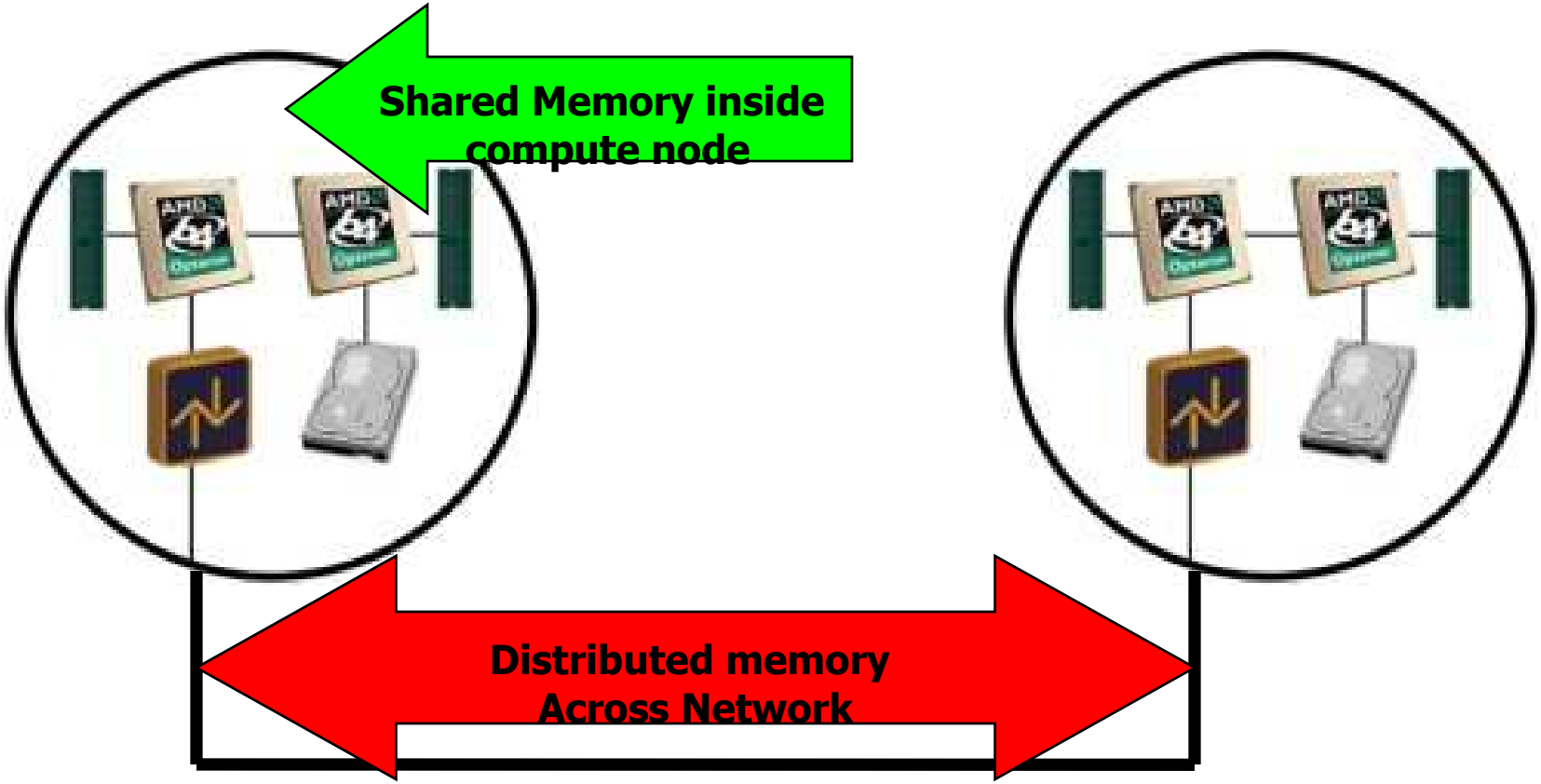
Hybrid Programming: **Combining two or more means of parallel programming** (MPI + pthreads or MPI + OpenMP, etc)

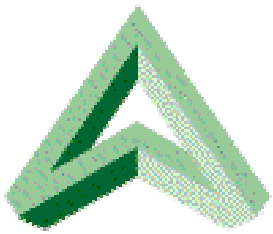
Clusters consisting of SMP Nodes will remain a cost effective means of high performance computing well into the future

One parallel programming method **may not best match** the problem to the hardware

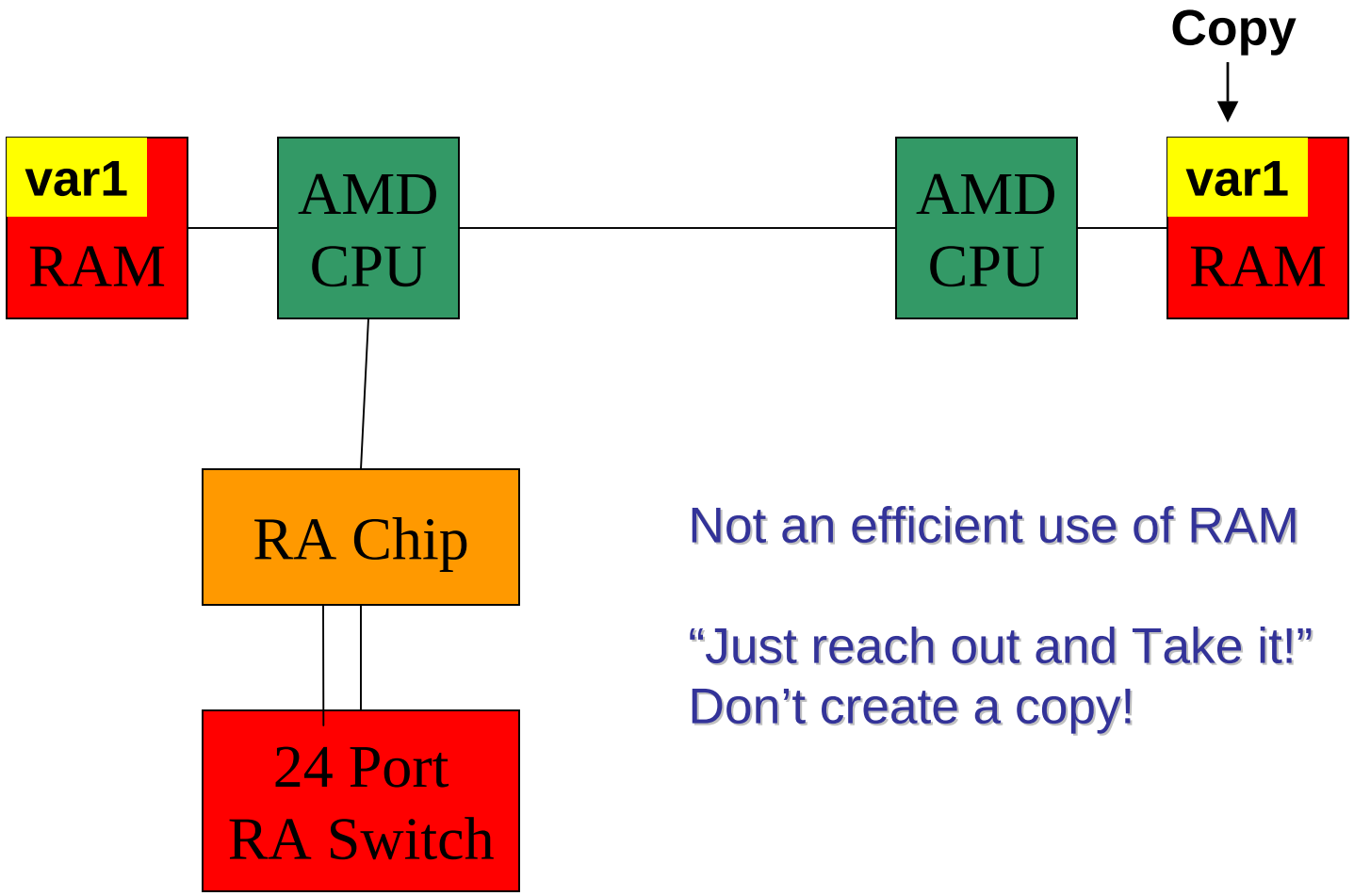
In the parallel bzip2 program **threads made sense for communication and I/O tasks**

Why Hybrid Programming on an XD1?





Problems with MPI on an SMP

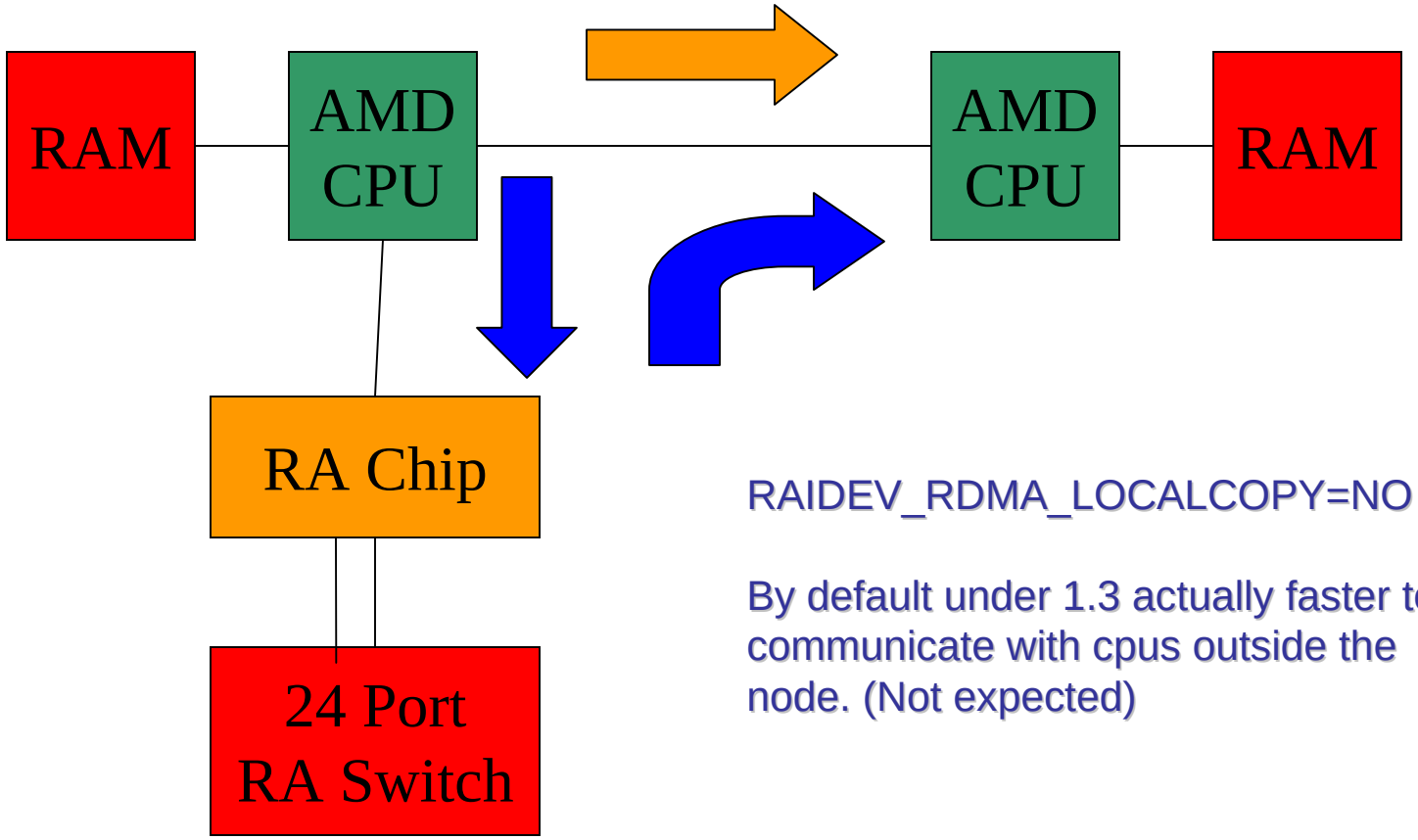


Not an efficient use of RAM

"Just reach out and Take it!"
Don't create a copy!

Problems with MPI on an SMP

RAIDEV_RDMA_LOCALCOPY=YES 20% Higher Bandwidth
(Intranode communication should be faster than Internode!)



RAIDEV_RDMA_LOCALCOPY=NO

By default under 1.3 actually faster to communicate with cpus outside the node. (Not expected)

Goals of the Project

Backup/compress a large amount of data (**Terabytes**) in a reasonable amount of time (**overnight**)

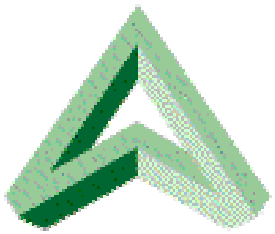
See how **fast** bzip2 can go

Be as **efficient** as possible

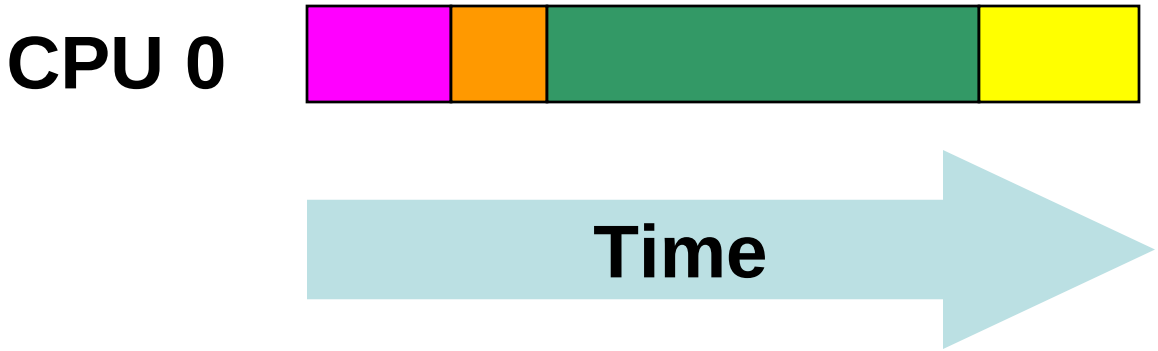
Match algorithm to XD1 hardware

Baby Step 0

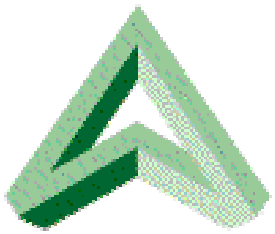
```
//Read a file into a buffer (yes the whole file…)
InputFile.read (inbuffer,filelength);

 // Init  a bzstream object
 bz_stream my_bzstream;
 my_bzstream.bzalloc=NULL; my_bzstream.bzfree=NULL;
     my_bzstream.opaque=NULL;
 BZ2_bzCompressInit (&my_bzstream,9,4,0);
 my_bzstream.next_in=inbuffer;
 my_bzstream.avail_in=filelength;
 my_bzstream.next_out=outbuffer;
 my_bzstream.avail_out=filelength;

// Compress Buffer
while (BZ2_bzCompress (&my_bzstream,BZ_FINISH) != BZ_STREAM_END ) {
       cout << my_bzstream.next_out ;
}

// Write out compressed file.
ofstream OutputFile("bible.bz2");
OutputFile.write(outbuffer,filelength - my_bzstream.avail_out);
OutputFile.close();

BZ2_bzCompressEnd(&my_bzstream);
```



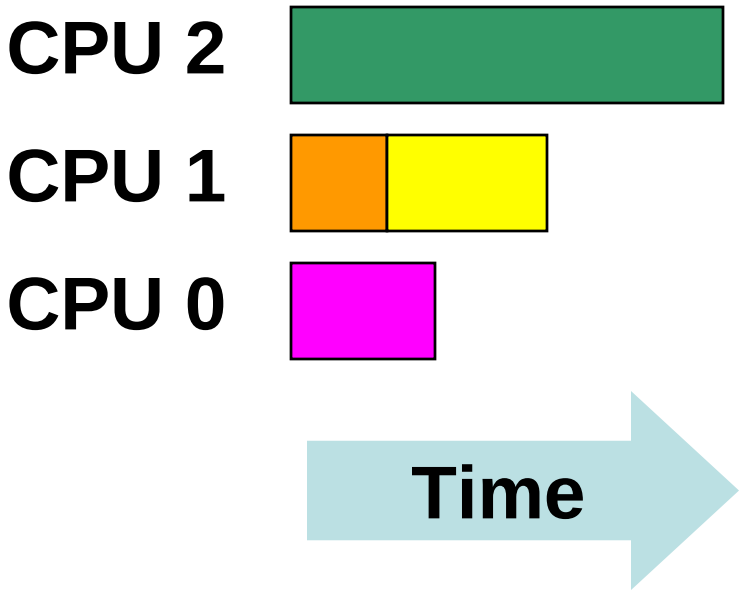
Compressing in Serial



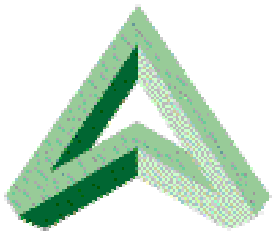
File is split into equal sized pieces,
each of which are compressed serially



Compressing in Parallel



File is split into equal sized pieces,
each of which are compressed in parallel

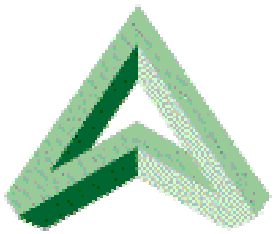


1st parallel version (MPI Only)

Master Process

```
while (!eof()) {
    Read from file
    Send 1 MB buffers to each slave to compress
    Compress Master's 1 MB piece
    Recv from all pieces (MPI_Gatherv)
    Write compressed buffers out
}
```

Achieved an 8x speedup with 20 cpus



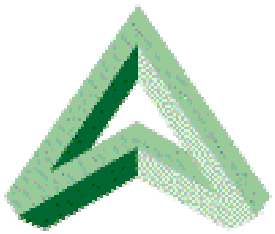
Problems with MPI only version

Master process blocked on I/O at the worst possible times

Slaves were not working when buffers were in transit

Compressing a 1MB buffer varied from 0.1 – 1.0 second

Resulted in Slaves compressing less than 50% of the time



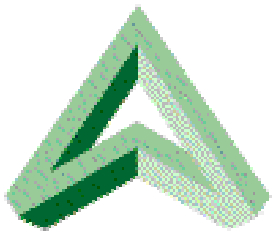
Version 2 Ideas
(We need more than just MPI)

Dynamic load balancing - Accounts for variable compression time

Need to turn around a Slave's request for work immediately

Asynchronous I/O – We can spin the disk and fill memory buffers before we need to use them

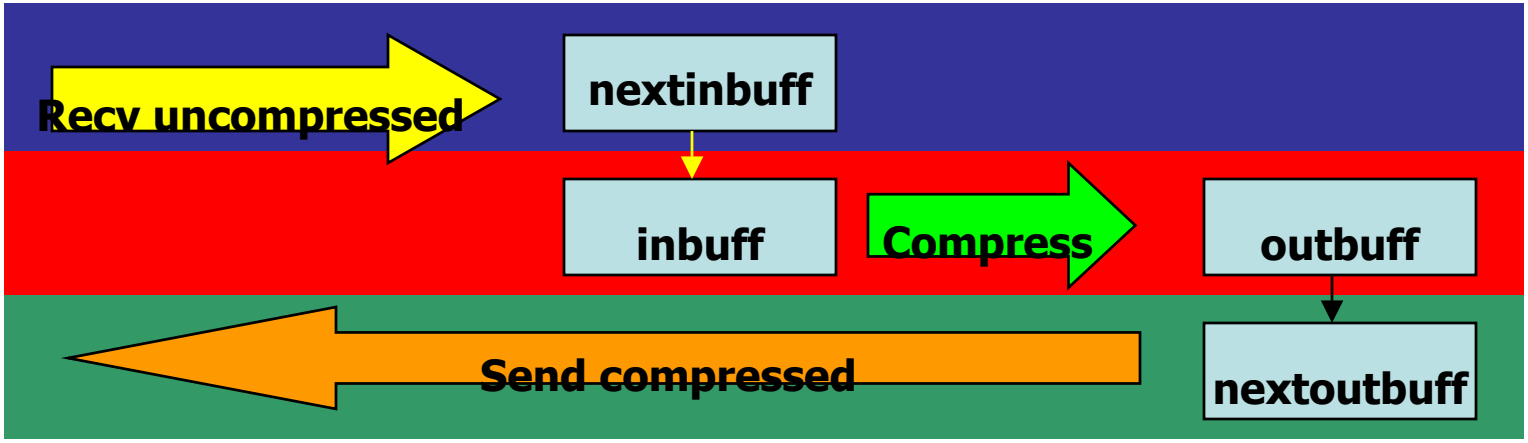
Overlap Communication and Compression - Slaves should be able to compress one packet while sending/receiving

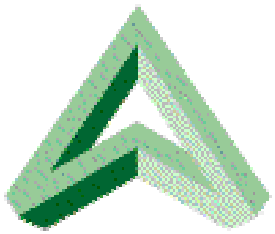


Threaded MPI Slave Process

Overlap Communication and Compression

- 1 Thread to keep next buffer to compress full
- 1 Thread to send compressed buffer back
- 1 Thread to compress buffer





Threaded MPI Master Process

Asynchronous I/O

1 Thread to keep inbuffers full

1 Thread to write outbuffers to file

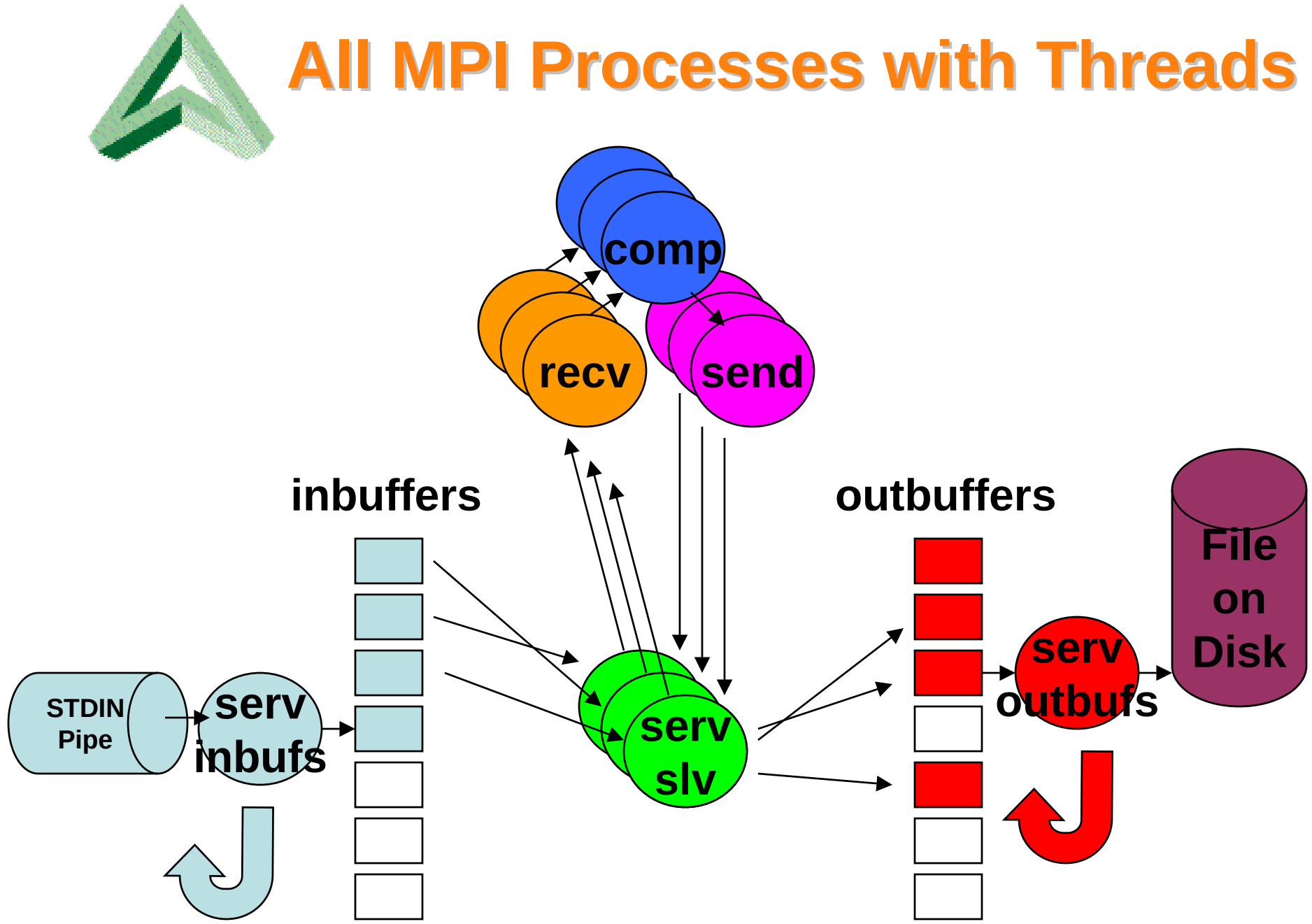
Dynamic load balancing

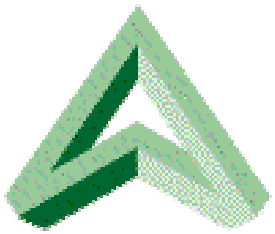
1 Thread per MPI Slave to manage communication

(Receive compressed packets out of order)

(Allow some nodes to turn around more packets than others)

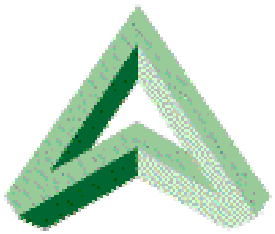
(should only block on spinning disks…)


All MPI Processes with Threads
comp
recv
send
inbuffers
outbuffers
File
on
Disk
STDIN
Pipe
serv
inbufs
serv
slv
serv
outbufs



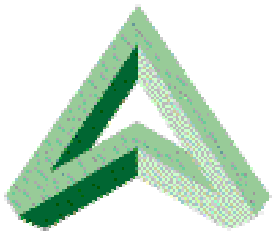
Debugging the Hybrid Version

uahrcw@c275-6:~/project> cat qsub.sh.e10011
xd1launcher: executing /home/uahrcw/project/pbzip
xd1launcher: executing /home/uahrcw/project/pbzip
xd1launcher: executing /home/uahrcw/project/pbzip
xd1launcher: executing /home/uahrcw/project/pbzip
MPI Slave #1 sending work request.
MPI Slave #3 sending work request.
MPI Slave #2 sending work request.
MPI Slave #1 : recvd work to compress of length = 1048576
MPI Slave #1 0.560455 0.8201 0.259645
MPI Slave # 1 sending compressed buffer back to master of length 208019
MPI Slave #1 sending work request.
MPI Slave #2 : recvd work to compress of length = 1048576
pbzip: /tmp/igorodet/rpm/BUILD/mpich-1.2.6/mpid/rai/dreg.c:307: dreg_decr_refcount: Assertion `d->refcount > 0' failed.
mpiexec: Error: read_full: EOF, only 0 of 4 bytes.



Checking if MPI is Thread Safe

```
MPI_Init_thread(&argc,&argv,MPI_THREAD_MULTIPLE,&provided);

if ( provided == MPI_THREAD_MULTIPLE )
        cout << "This version of MPI is thread safe" << endl << flush;
```



MPI_INIT_THREAD options

{ **MPI_THREAD_SINGLE**}

Only one thread will execute

{ **MPI_THREAD_FUNNELED**}

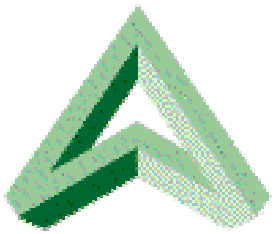
The process may be multi-threaded, but only the main thread will make MPI calls (all MPI calls are funneled to the main thread)

{ **MPI_THREAD_SERIALIZED**}

The process may be multi-threaded, and multiple threads may make MPI calls, but only one at a time: MPI calls are not made concurrently from two distinct threads (all MPI calls are serialized)

{ **MPI_THREAD_MULTIPLE**}

Multiple threads may call MPI, with no restrictions



Putting MPICH2 on the XD1

XD1's **MPICH** isn't thread safe

Generally **MPICH1** isn't thread safe

To compile **MPICH2** to be thread safe*

```
./configure --prefix=/opt/asn/apps/mpich2 \
            --enable-threads \
            --with-thread-package=posix
```

*not RapidArray optimized

Running my job with MPICH2

Script1 – qsub-mpich2.sh

```
#!/bin/bash
#PBS -l nodes=1:ppn=1:cpp=2+28:ppn=1,mem=1gb,cput=00:30:00 -joe
mpiprocs=29
cd /home/uahrcw/project
# Start Daemon used for MPICH2 communications.
mpdboot  -f $PBS_NODEFILE -n $mpiprocs
# Run the program
mpiexec -np 1 mpich2.sh : -np $((mpiprocs-1)) ./pbzip
# Stop Daemon used for MPICH2 communications.
mpdallexit
```

Running my job with MPICH2

mpich2.sh (only required for master)

```
#!/bin/bash
cd /home/uahrcw/project
tar -cf - /genomes/H_sapiens | ./pbzip –o hs.tar.bz2
```

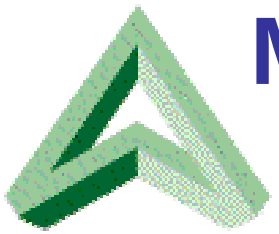
Submitting job to the queue system

qsub qsub-mpich2.sh

Watching it run

qstat –an


service_inbuffers : read in index # 0
service_slave1 : Sending inbuff[0] to compress length = 1048576
service_inbuffers : read in index # 1
service_slave2 : Sending inbuff[1] to compress length = 1048576
service_inbuffers : read in index # 2
service_slave4 : Sending inbuff[2] to compress length = 1048576
service_inbuffers : read in index # 3
service_slave6 : Sending inbuff[3] to compress length = 1048576
service_inbuffers : read in index # 4
service_slave7 : Sending inbuff[4] to compress length = 1048576
service_inbuffers : read in index # 5
service_slave5 : Sending inbuff[5] to compress length = 1048576
service_inbuffers : read in index # 6
service_slave8 : Sending inbuff[6] to compress length = 1048576
service_inbuffers : read in index # 7
service_slave3 : Sending inbuff[7] to compress length = 1048576
service_inbuffers : read in index # 8
service_inbuffers : read in index # 9
service_inbuffers : read in index # 10
service_inbuffers : read in index # 11
service_inbuffers : read in index # 12
service_inbuffers : read in index # 13
service_inbuffers : read in index # 14
service_inbuffers : read in index # 15
service_slave1 : Recving compressed buffer of length 275268
service_slave1 : Recving work request
service_slave1 : Sending inbuff[8] to compress length = 1048576
service_outbuffers : writing out 0
service_outbuffers : sleeping on nextout 1
service_slave2 : Recving compressed buffer of length 277579
service_slave2 : Recving work request
service_slave2 : Sending inbuff[9] to compress length = 1048576
service_outbuffers : writing out 1
service_outbuffers : sleeping on nextout 2
service_slave4 : Recving compressed buffer of length 296340
service_slave4 : Recving work request
service_slave4 : Sending inbuff[10] to compress length = 1048576
service_outbuffers : writing out 2
service_outbuffers : sleeping on nextout 3
service_slave6 : Recving compressed buffer of length 287806
service_slave6 : Recving work request
9 cpu run
Service_slaves wake up when Inbuffer is available
Keeping disk busy
Out of order recv
Out of order recv



MPI Slave process Bandwidth Requirements

Measured that **compression of 1MB** buffer ranged from 0.1-1 second

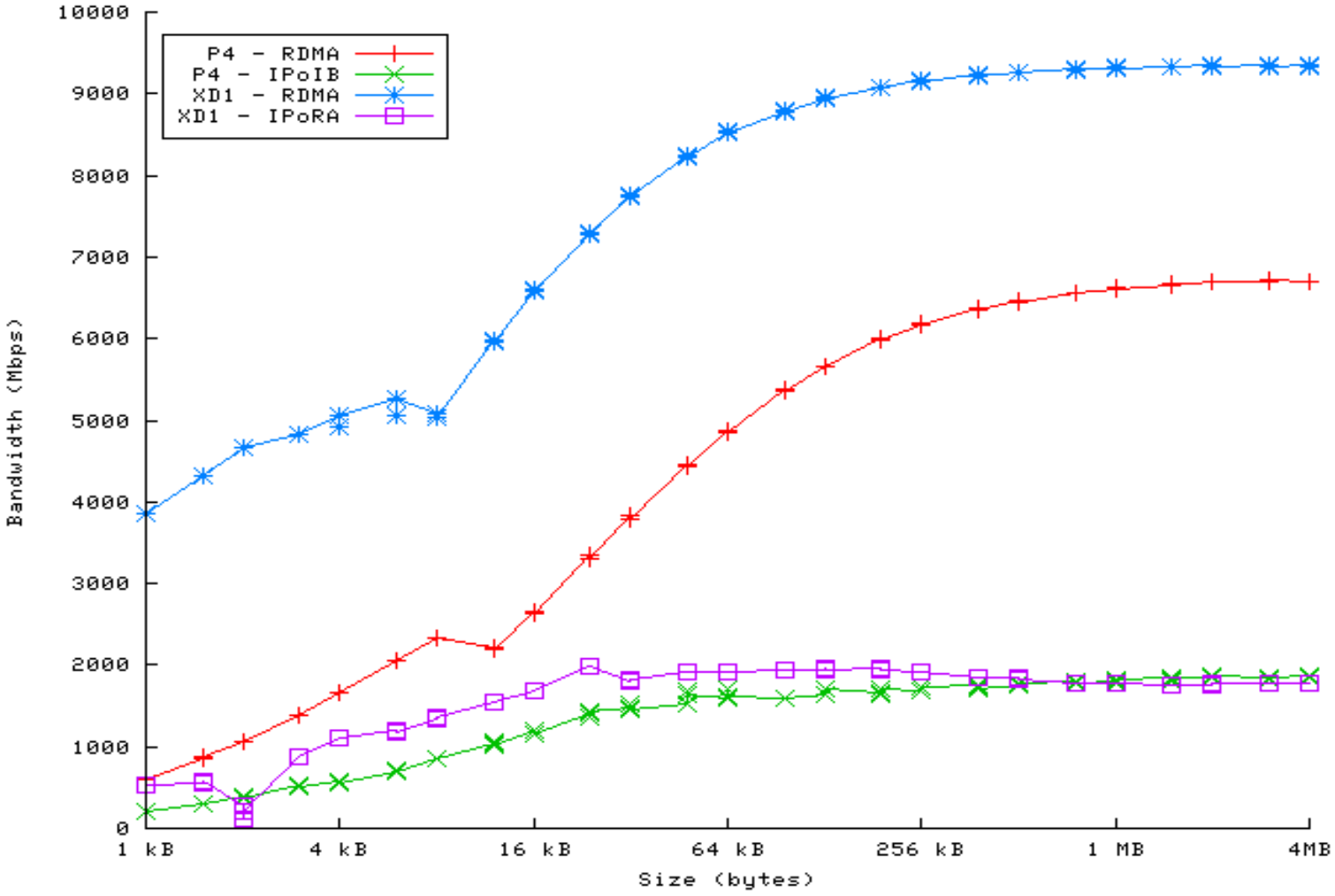
Maximum Bandwidth a compute node would require

(1MB*2xfers*8Mb/MB)/.1sec = 160 Mbits/sec

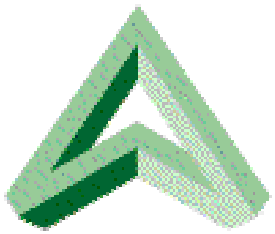
RapidArray Native Master should be capable of **supporting 62 Slaves** (10000Mbits/160Mbits)

RA Native Master might be capable of **supporting 300+ Slaves** if we estimate compress time average is .5 seconds

MPICH2 Network Performance



http://www.osc.edu/~dennis/rdma/rdma.html



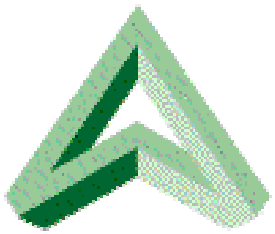
Scalability under MPICH2

Using IPoRA results in **100% CPU utilization**, limiting bandwidth to 1800 Mbits/sec

An **MPICH2** master process should support about **11 Slaves minimum (1800/160)**

Should support about 75 typical slaves

Ran into problems using 8 slaves - why?



Profiling the Code

TAU http://www.cs.uoregon.edu/research/tau/home.php

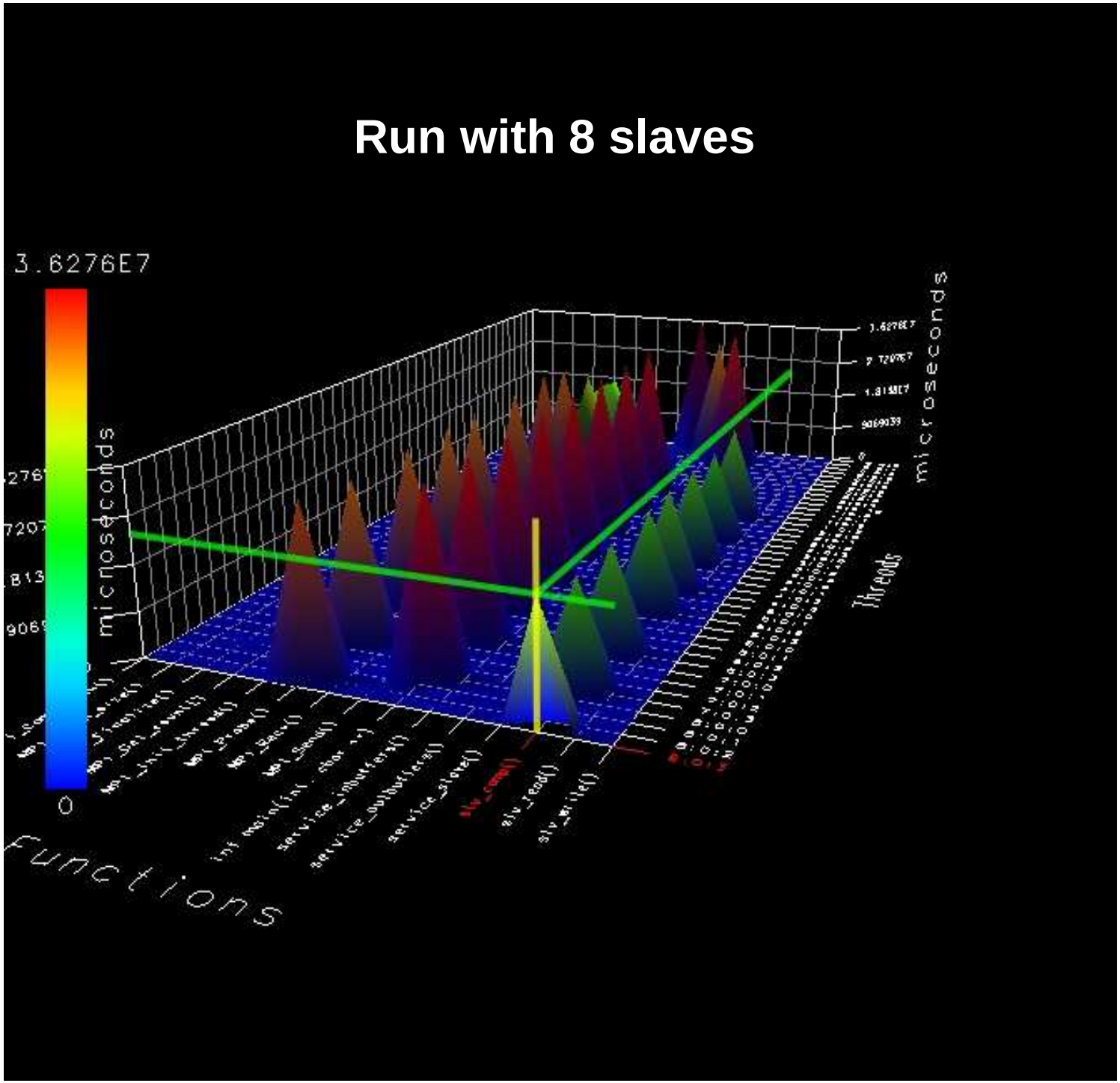
Idea

Src Code -> auto instrumentor -> New Src Code -> Compile/Execute

When pthreads are involved

Src Code -> instrument by hand -> New Src Code -> Compile/Execute

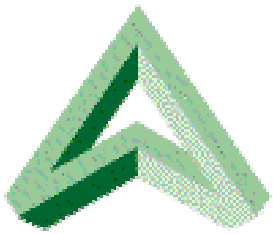
Update 5/9/2006 – Sameer Shende says auto instrumentor should now be fixed to handle pthreads. (Thanks)

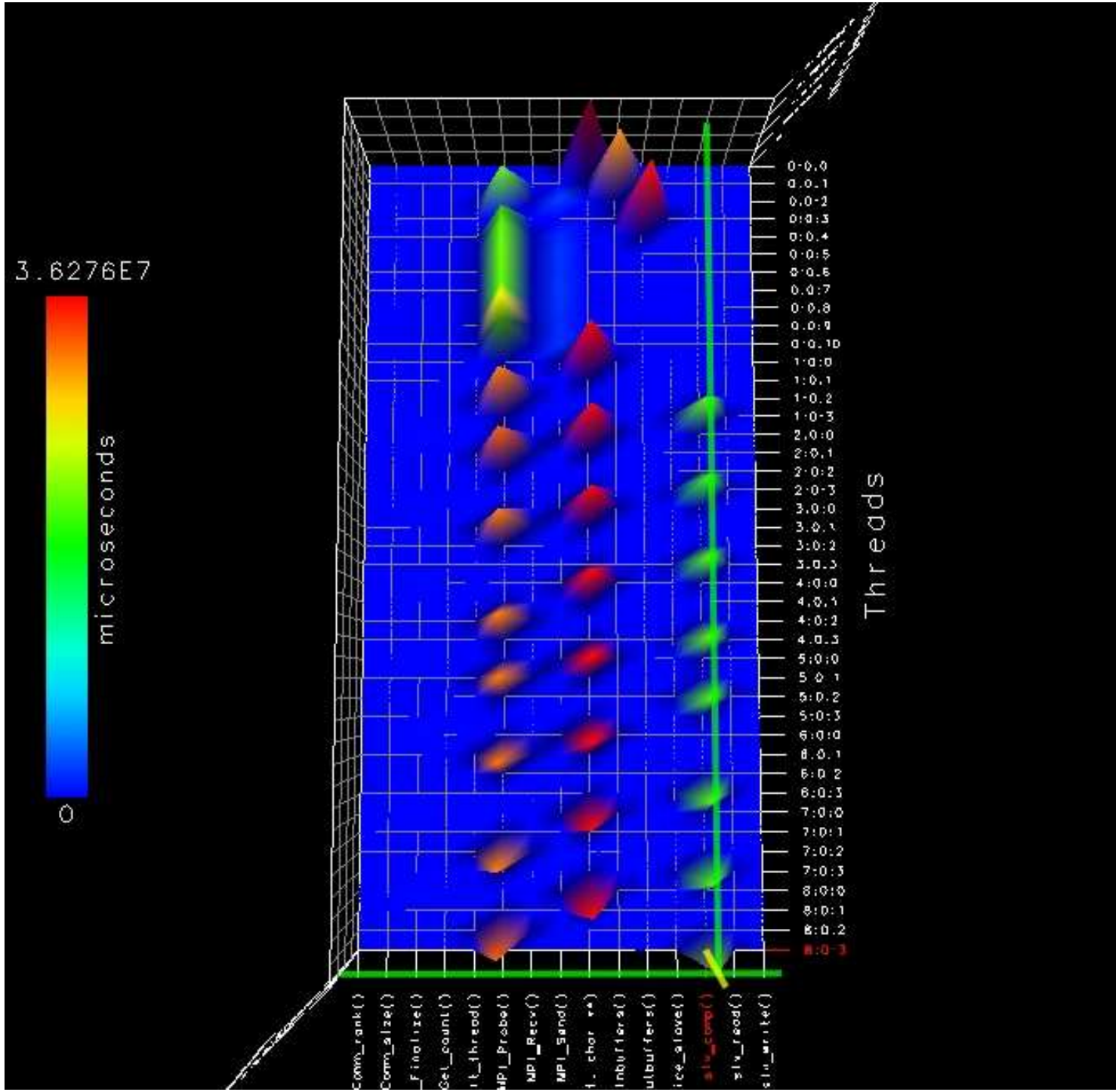


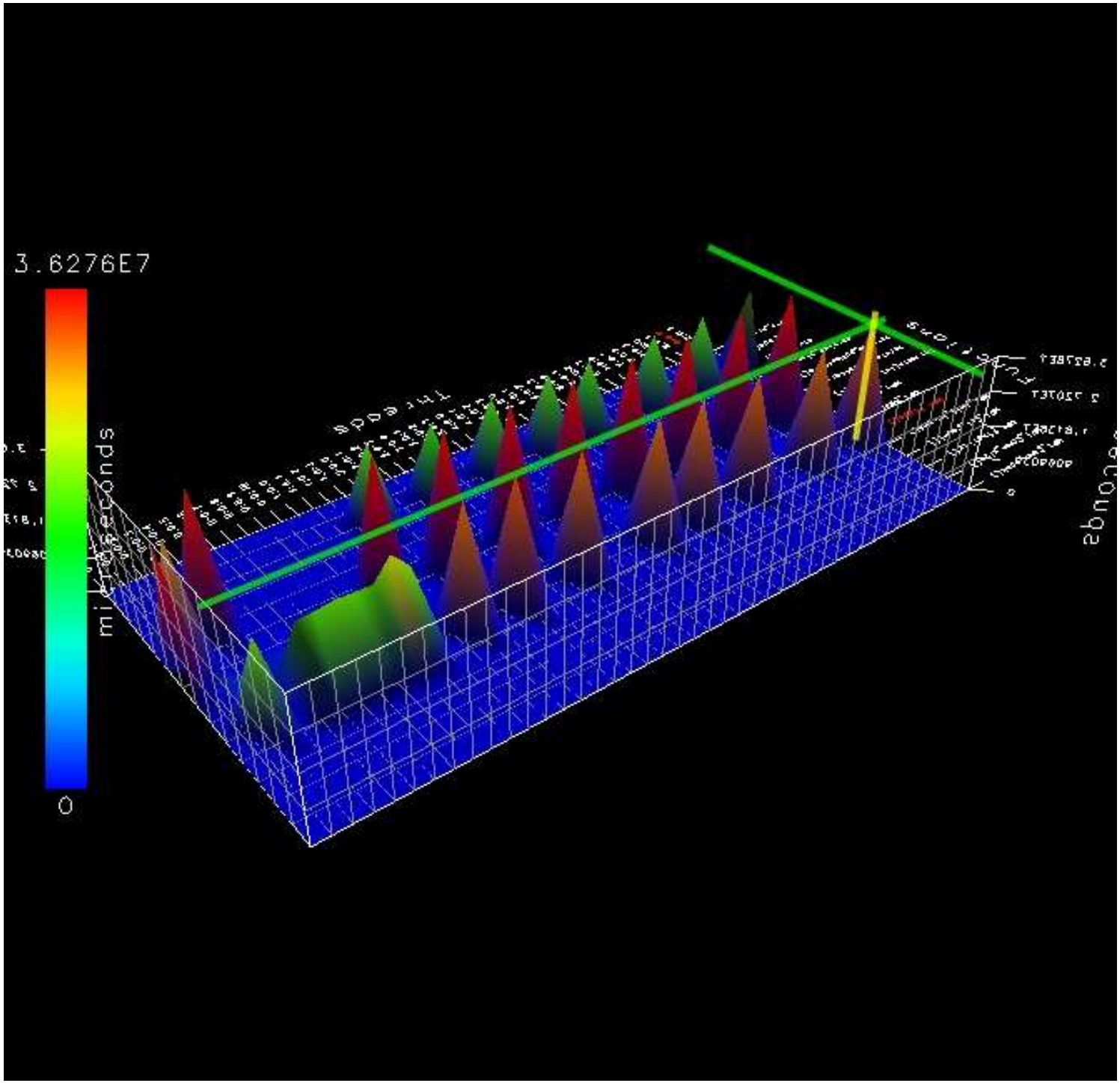
Orange Trees
MPI_Probe

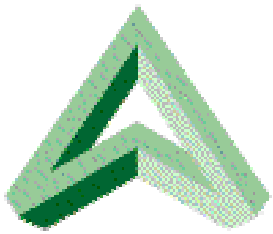
Red Trees
Total Time
Thread Lives

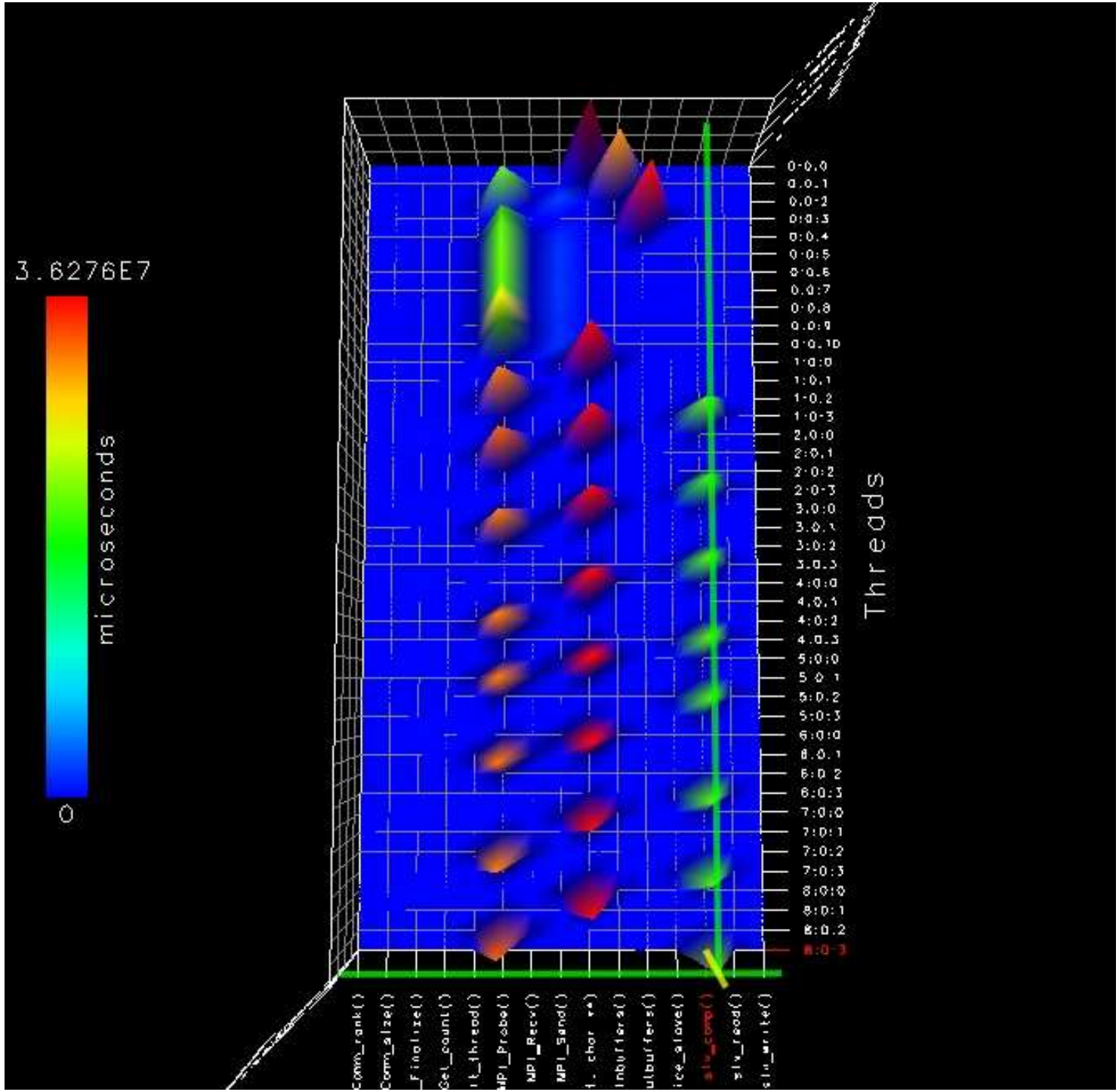
Green Trees
Time Spent
Compressing



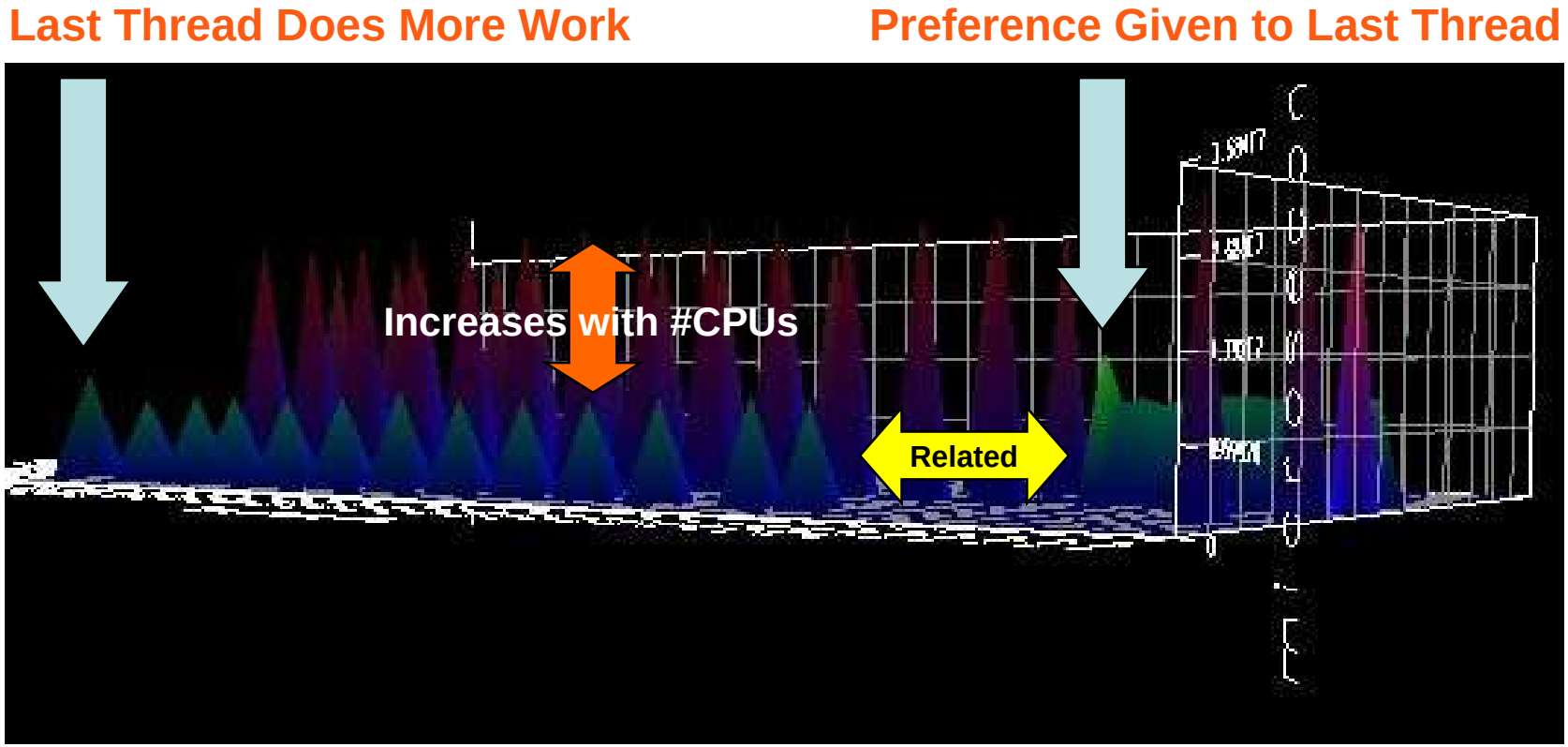

3.6276E7
microseconds
0
Threads
MPI_Recv()
MPI_Send()
slv_read()
slv_write()

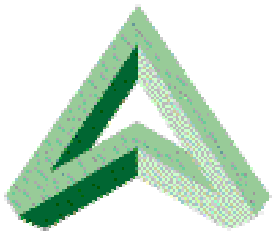

3.6276E7
0
microseconds




3.6276E7
microseconds
0
Threads
MPI_Recv()
MPI_Send()
slv_read()
slv_write()

Time spent in MPI_Probe is limited by other stuff…
=> Eliminate as much other stuff as possible…

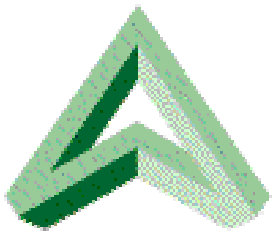




Top at first glance

One master process shows that all threads combined are using 160% cpu
Which threads are taking the most time?

```
top - 15:33:36 up 39 days, 14:06,  1 user,  load average: 0.45, 0.17, 0.36
Tasks: 164 total,   2 running, 162 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.6% us, 50.2% sy,  0.0% ni,  7.3% id,  4.0% wa,  9.0% hi, 17.9% si
Mem:   2051132k total,  2035960k used,    15172k free,     7580k buffers
Swap:  4192924k total,   945204k used,  3247720k free,  1383812k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2815 uahrcw    25   0  551m 281m 1252 S  160 14.0   0:19.86 ./pbzip -o /scr
 2813 uahrcw    16   0  9240 1404 1068 S    0  0.1   0:00.01 /bin/bash ./mpi
 2812 uahrcw    16   0 35368 7284 2756 S    0  0.4   0:00.00 python2.3 /opt/
 2811 uahrcw    15   0 38904 8616 2848 S    0  0.4   0:00.24 python2.3 /opt/
 2798 uahrcw    16   0 35416 7280 2720 S    0  0.4   0:00.00 python2.3 /opt/
 2792 uahrcw    19   0  9240 1400 1068 S    0  0.1   0:00.00 /bin/bash /var/
 2766 uahrcw    16   0  4728  384  292 S    0  0.0   0:00.00 pbs_demux
 2731 uahrcw    19   0  9240 1572 1200 S    0  0.1   0:00.00 -bash
```

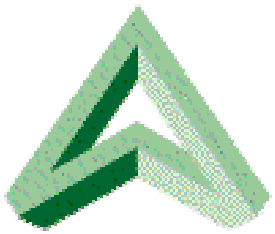


Xd1's Top Doesn't Show Threads

Compile your own procps package to show threads

Start top, filter by user, show threads, sort by pid

Pid numbers show order threads were created, thus you can identify individual threads

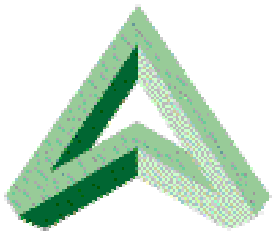


Actual Bottleneck

Service_inbuffer Thread:

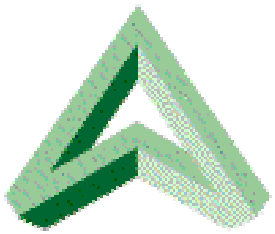
Default cin buffer was too small resulting in 99% cpu usage.

```
char mybuffer [bufferlength];
cin.rdbuf()->pubsetbuf(mybuffer,bufferlength);
```



Speedup and Efficiency

Version	Speedup	Ncpus	Notes
MPI only	8x	20	40% efficient
MPI+pthreads	8.9x	20	45% efficient
MPI+pthreads	8.1x	9	90% efficient
MPI+pthreads+	19.78x	20	98.9% efficient



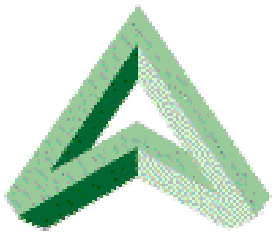
Accidental Superlinear Speedup

Bzip2 does a lot of **sorting**

Using Bzip2 library in parallel results in less sorting?

Compression Ratio is slightly effected

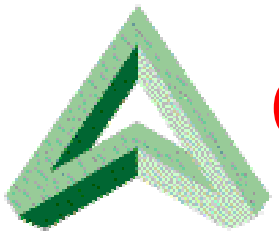
For 4.4 Gig test file
serial bzip compression ratio vs. pbzip
2.6179:1 vs 2.6135:1



Conclusions

Combining MPI and pthreads can have some real advantages

In the bzip2 program the **advantages overshadowed** the **performance penalty** of using MPICH2 compiled without RapidArray support

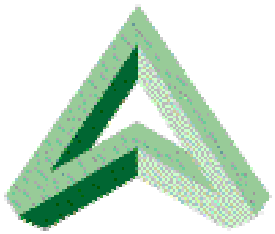


Cray's Position on RA + MPICH2

I would like Cray to provide a **thread safe RapidArray optimized version** of **MPICH2**

"No current plans to implement"

I encourage other XD1 sites to **ask Cray for a MPICH2 + RapidArray native implementation**



Questions/Comments?

charles@asc.edu