Profiling and Performance Tuning

Apr 2012

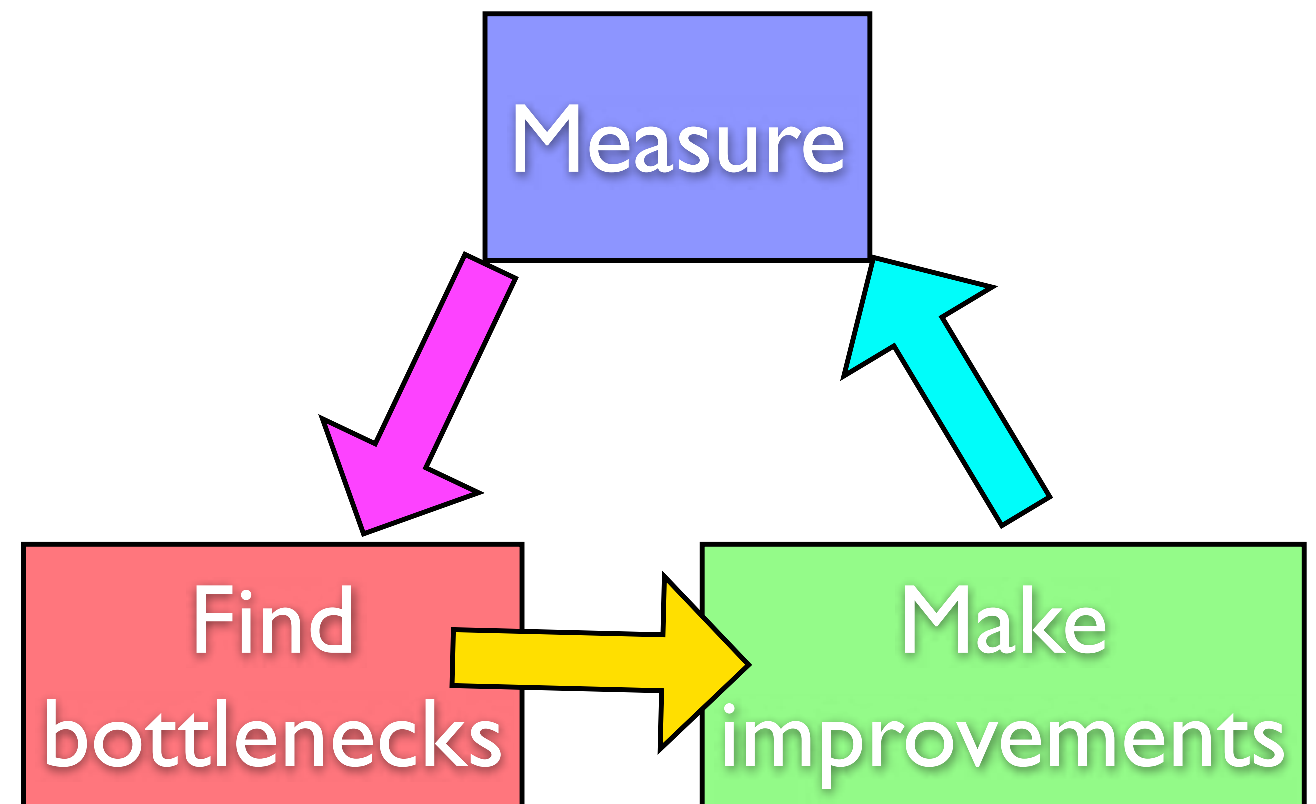
SciNet

Outline

- Basics of profiling, and open-source profiling tools
- Intel profiling tools
- Allinea MAP profiling tool
- Auto-tuning MPI performance

How to improve Performance?

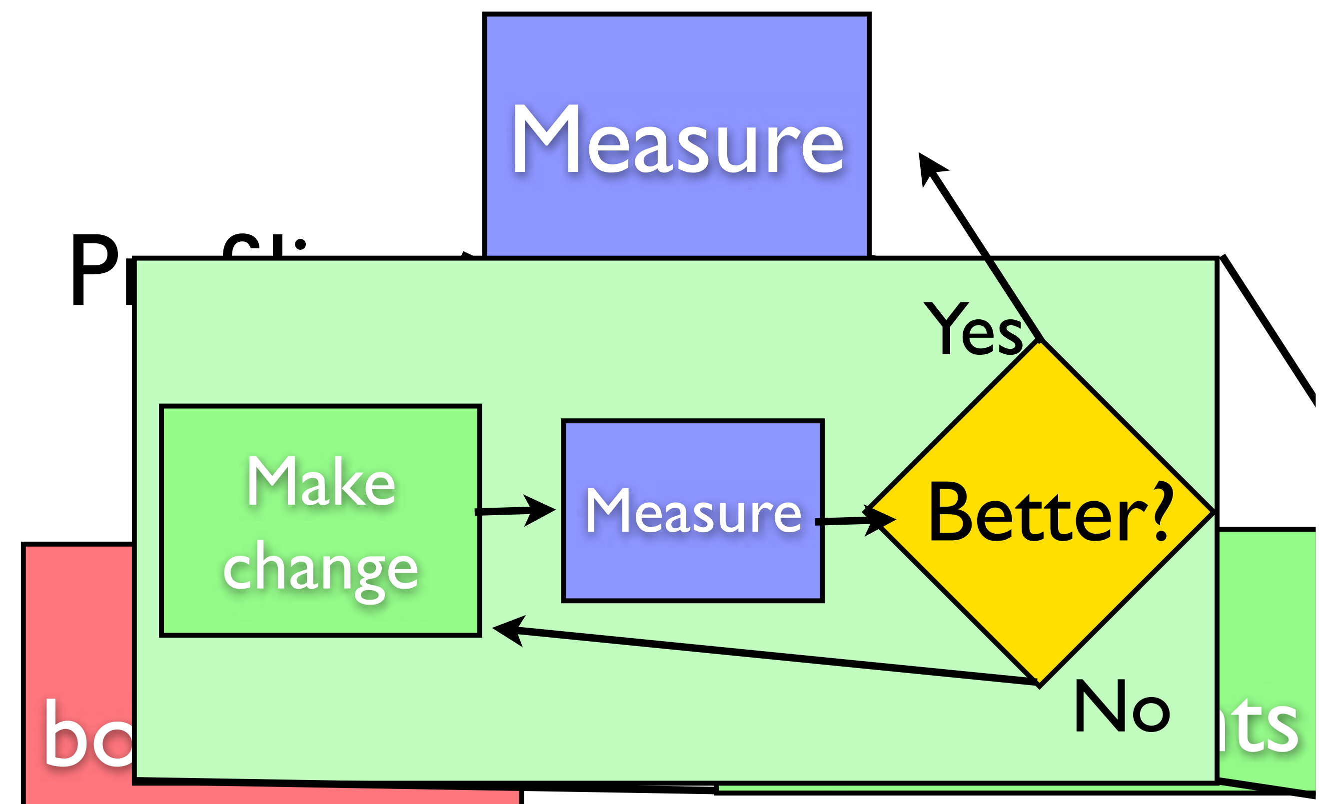
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

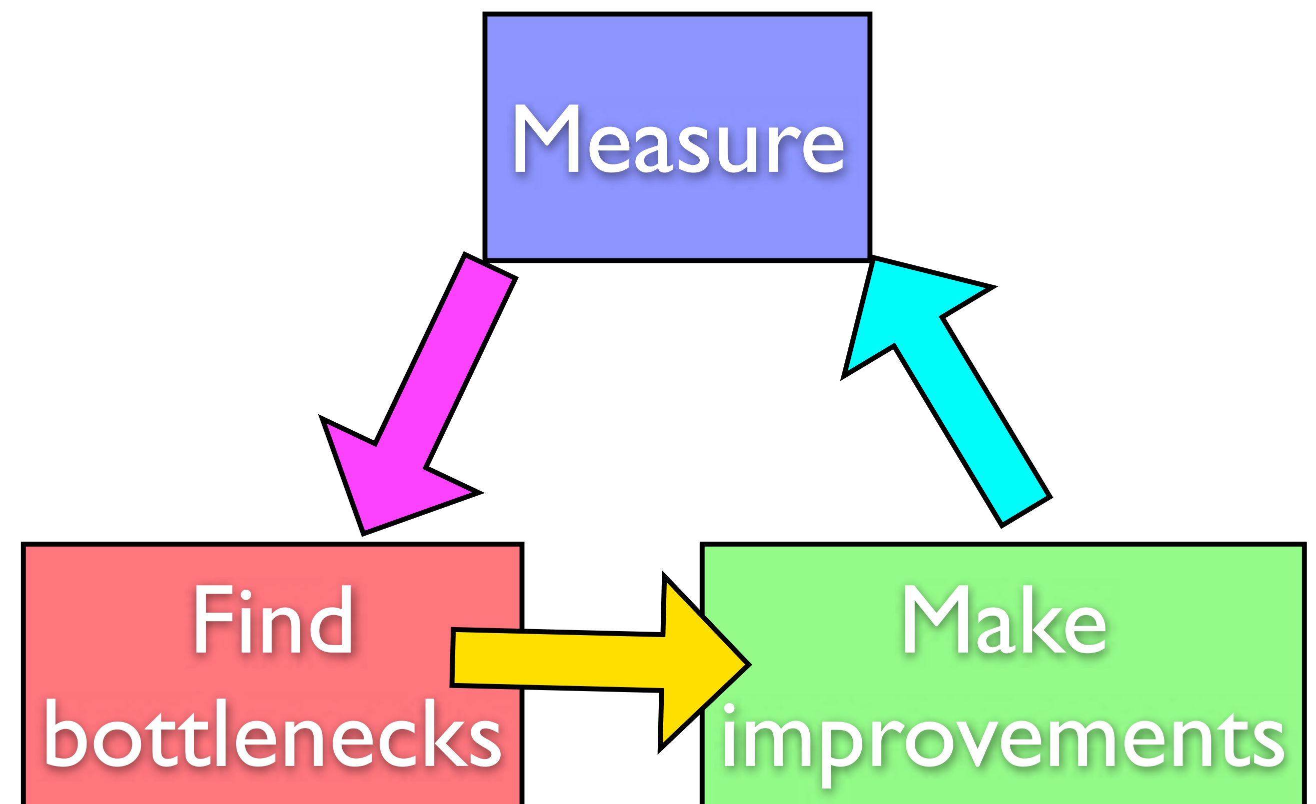
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

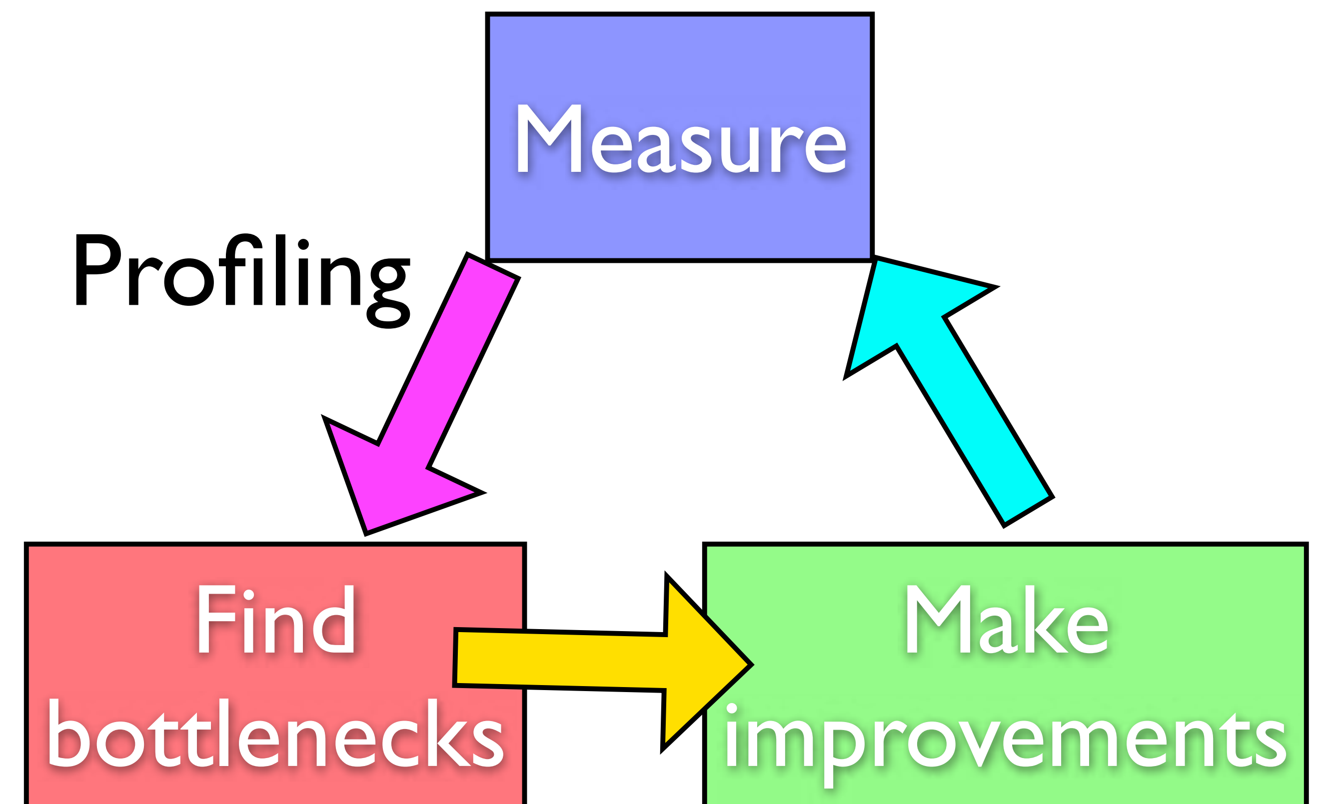
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

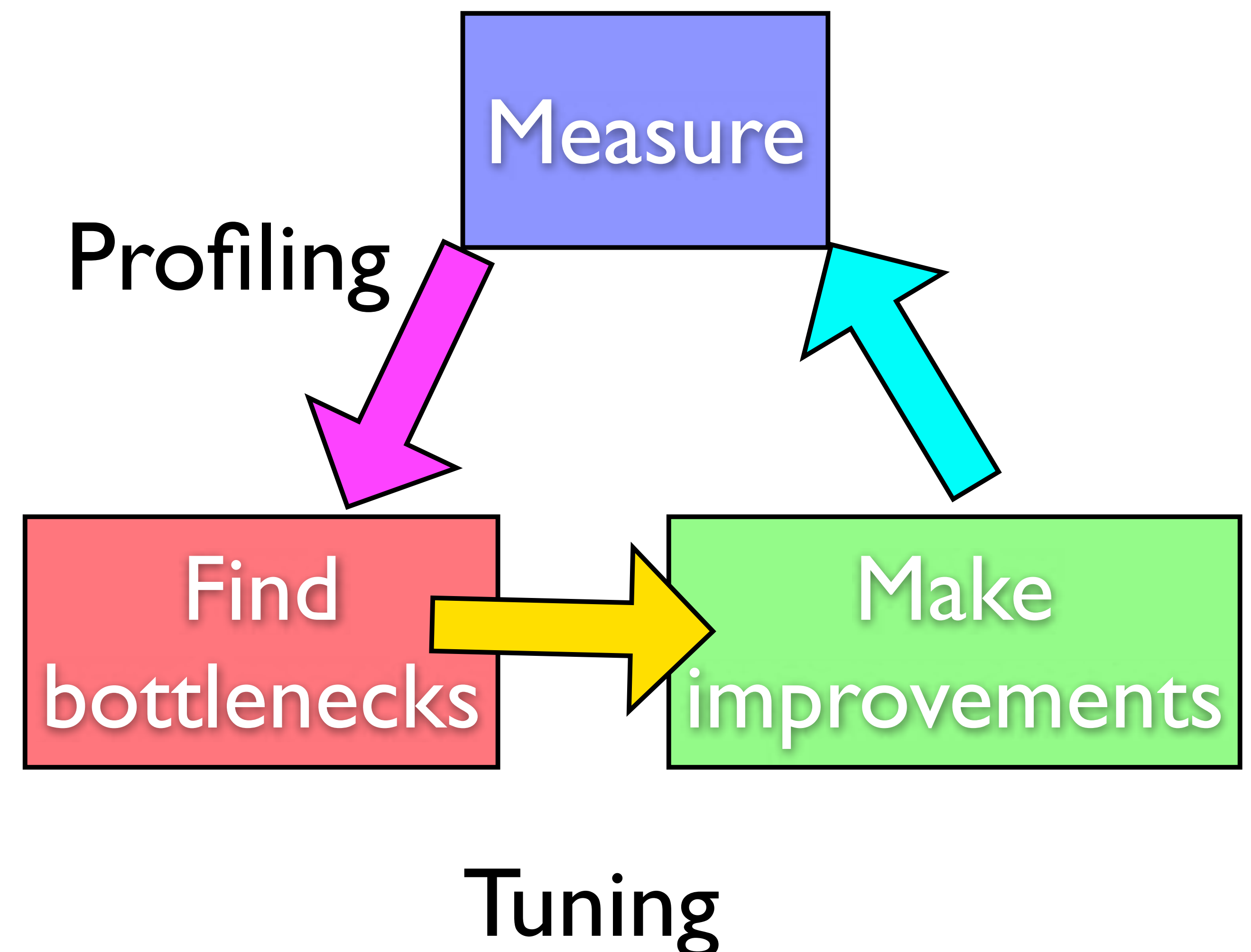
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

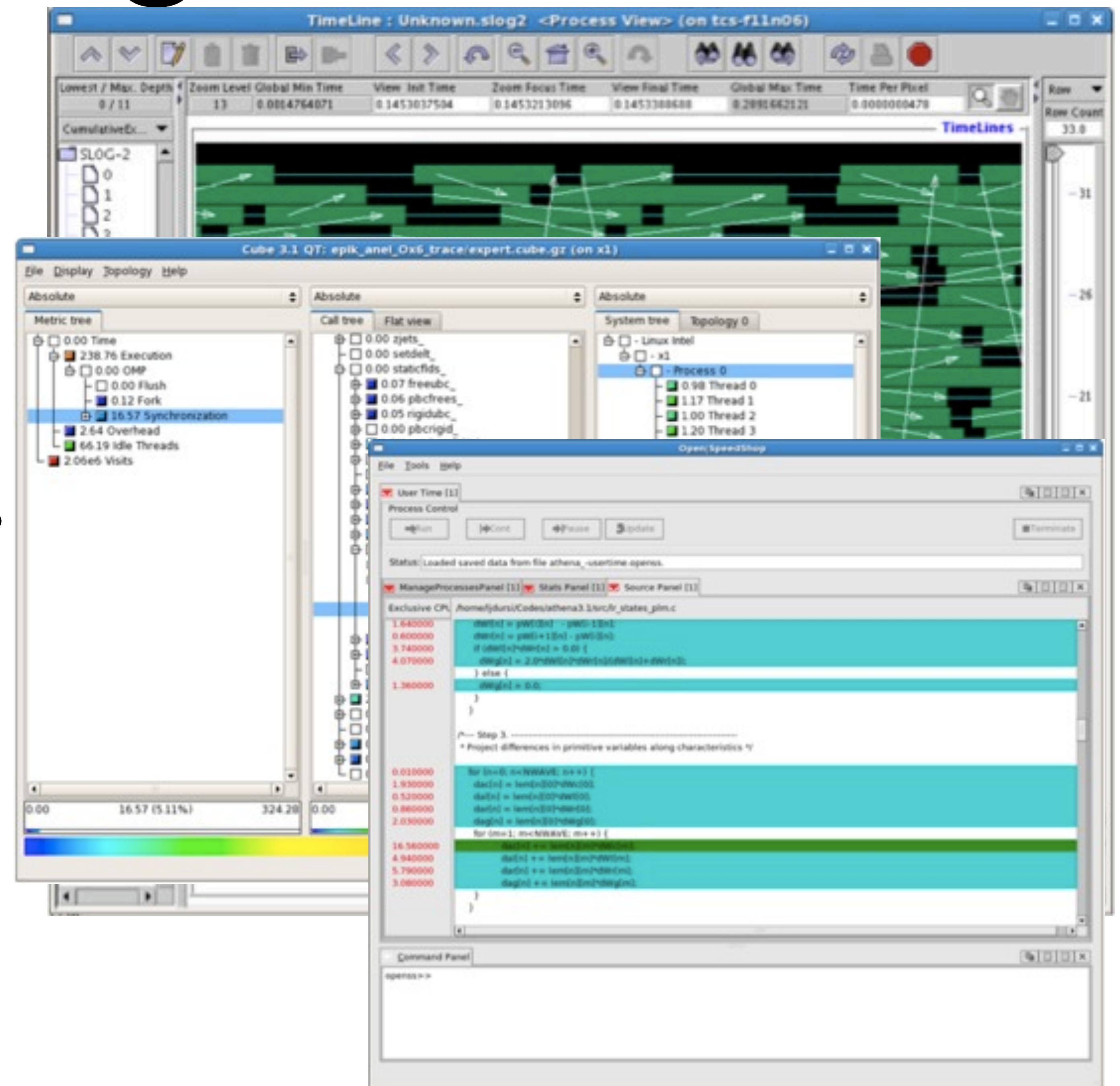
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

Profiling Tools

- Here we'll focus on profiling.
- Tuning - each problem might have different sorts of performance problem
- Tools are general
- Range of tools on GPC



SciNet

Profiling A Code

- Where in your program is time being spent?
- Find the expensive parts
  - Don't waste time optimizing parts that don't matter
- Find bottlenecks.

```c
    case SIM_PROJECTILE:
        ymin = xmin = 0.;
        ymax = xmax = 1.;
        dx = (xmax-xmin)/npts;
        dy = (ymax-ymin)/npts;
        init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
        projectile_initvalues(&d, psize, pdens, pvel);
        outputvar = DENSVAR;
        break;
}

/* apply boundary conditions and make thermodynamically consistant */
bcs[0] = xbc; bcs[1] = xbc;
bcs[2] = ybc; bcs[3] = ybc;
apply_all_bcs(&d,bcs);
domain_backward_dp_eos(&d);
domain_ener_internal_to_tot(&d);


/* main loop */

tick(&tt);
if (output) domain_plot(&d);
printf("Step\tdt\ttime\n");
for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

    printf("%d\t%g\t%g\n", step, dt, time);

    if (output && ((step % outevery) == 0) ) {
        sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
        sprintf(binfilename,"dens_test_%d.bin", outnum);
        sprintf(h5filename,"dens_test_%d.h5", outnum);
        sprintf(ncdffilename,"dens_test_%d.nc", outnum);
        domain_output_ppm(&d, outputvar, ppmfilename);
        domain_output_bin(&d, binfilename);
        domain_output_hdf5(&d, h5filename);
        domain_output_netcdf(&d, ncdffilename);
        domain_plot(&d);
        outnum++;
    }
    kl_timestep_xy(&d, bcs, dt);
    apply_all_bcs(&d,bcs);

    kl_timestep_yx(&d, bcs, dt);
    apply_all_bcs(&d,bcs);
}
tock(&tt);
```

SciNet

Profiling A Code

- Timing *vs.* Sampling *vs.* Tracing
- Instrumenting the code vs. Instrumentation-free

```
    case SIM_PROJECTILE:
        ymin = xmin = 0.;
        ymax = xmax = 1.;
        dx = (xmax-xmin)/npts;
        dy = (ymax-ymin)/npts;
        init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
        projectile_initvalues(&d, psize, pdens, pvel);
        outputvar = DENSVAR;
        break;
}

/* apply boundary conditions and make thermodynamically consistant */
bcs[0] = xbc; bcs[1] = xbc;
bcs[2] = ybc; bcs[3] = ybc;
apply_all_bcs(&d,bcs);
domain_backward_dp_eos(&d);
domain_ener_internal_to_tot(&d);


/* main loop */

tick(&tt);
if (output) domain_plot(&d);
printf("Step\tdt\ttime\n");
for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

    printf("%d\t%g\t%g\n", step, dt, time);

    if (output && ((step % outevery) == 0) ) {
        sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
        sprintf(binfilename,"dens_test_%d.bin", outnum);
        sprintf(h5filename,"dens_test_%d.h5", outnum);
        sprintf(ncdffilename,"dens_test_%d.nc", outnum);
        domain_output_ppm(&d, outputvar, ppmfilename);
        domain_output_bin(&d, binfilename);
        domain_output_hdf5(&d, h5filename);
        domain_output_netcdf(&d, ncdffilename);
        domain_plot(&d);
        outnum++;
    }
    kl_timestep_xy(&d, bcs, dt);
    apply_all_bcs(&d,bcs);

    kl_timestep_yx(&d, bcs, dt);
    apply_all_bcs(&d,bcs);
}
tock(&tt);
```

SciNet

Timing whole program

- Very simple; can run any command, incl in batch job
- In serial, real = user+sys
- In parallel, ideally user = (nprocs)x (real)

```
$ time ./a.out

[ your job output ]

real    0m2.448s
user    0m2.383s
sys     0m0.027s
```

Elapsed "walltime"

Actual user time

System time: Disk, I/O...

SciNet

Time in PBS *.o file

```
----------------------------------------
Begin PBS Prologue Tue Sep 14 17:14:48 EDT 2010 1284498888
Job ID:      3053514.gpc-sched
Username:    ljdursi
Group:       scinet
Nodes:       gpc-f134n009 gpc-f134n010 gpc-f134n011 gpc-f134n012
gpc-f134n043 gpc-f134n044 gpc-f134n045 gpc-f134n046 gpc-f134n047 gpc-f134n048
[...]
End PBS Prologue Tue Sep 14 17:14:50 EDT 2010 1284498890
----------------------------------------
[ Your job's output here... ]
----------------------------------------
Begin PBS Epilogue Tue Sep 14 17:36:07 EDT 2010 1284500167
Job ID:      3053514.gpc-sched
Username:    ljdursi
Group:       scinet
Job Name:    fft_8192_procs_2048
Session:     18758
Limits:      neednodes=256:ib:ppn=8,nodes=256:ib:ppn=8,walltime=01:00:00
Resources:   cput=713:42:30,mem=3463854672kb,vmem=3759656372kb,walltime=00:21:07
Queue:       batch_ib
Account:
Nodes:   gpc-f134n009 gpc-f134n010 gpc-f134n011 gpc-f134n012 gpc-f134n043
[...]
Killing leftovers...
gpc-f141n054:    killing gpc-f141n054 12412

End PBS Epilogue Tue Sep 14 17:36:09 EDT 2010 1284500169
----------------------------------------
```

SciNet

Can use ‘top’ on running jobs

```
$ checkjob 3802660
job 3802660

AName: GoL
State: Running
Creds:  user:ljdursi  group:scinet  [...]
WallTime:   00:00:00 of 00:20:00
SubmitTime: Tue Dec  7 21:53:41
  (Time Queued  Total: 00:00:22  Eligible: 00:00:22)

StartTime: Tue Dec  7 21:54:03
Total Requested Tasks: 16

Req[0]  TaskCount: 16  Partition: torque
Opsys: centos53computeA  Arch: ---  Features: compute-eth

Allocated Nodes:
[gpc-f109n001:8][gpc-f109n002:8]
```

```
gpc-f103n084-$ ssh gpc-f109n001
gpc-f109n001-$ top
```

```
top - 21:56:45 up  5:56,  1 user,  load average: 5.55, 1.73, 0.88
Tasks: 234 total,   1 running, 233 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.4%us, 36.2%sy,  0.0%ni, 52.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  16410900k total,  1542768k used, 14868132k free,        0k buffers
Swap:        0k total,        0k used,        0k free,   294628k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
22479 ljdursi   18   0  108m 4816 3212 S 98.5  0.0   1:04.81  6 gameoflife
22480 ljdursi   18   0  108m 4856 3260 S 98.5  0.0   1:04.85 13 gameoflife
22482 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.83  2 gameoflife
22483 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.82  8 gameoflife
22484 ljdursi   18   0  108m 4832 3232 S 98.5  0.0   1:04.80  9 gameoflife
22481 ljdursi   18   0  108m 4856 3256 S 98.2  0.0   1:04.81  3 gameoflife
22485 ljdursi   18   0  108m 4808 3208 S 98.2  0.0   1:04.80  4 gameoflife
22478 ljdursi   18   0  117m 5724 3268 D 69.6  0.0   0:46.07 15 gameoflife
 8042 root       0 -20 2235m 1.1g  16m S  2.3  6.8   0:30.59  8 mmfsd
10735 root      15   0  3792  452  372 S  1.3  0.0   0:16.80  0 cat
```

More system then user time -- not very efficient.
(Idle ~50% is ok -- hyperthreading)

SciNet

```
gpc-f103n084-$ ssh gpc-f109n001
gpc-f109n001-$ top
```

```
top - 21:56:45 up  5:56,  1 user,  load average: 5.55, 1.73, 0.88
Tasks: 234 total,   1 running, 233 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.4%us, 36.2%sy,  0.0%ni, 52.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  16410900k total,  1542768k used, 14868132k free,        0k buffers
Swap:        0k total,        0k used,        0k free,   294628k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
22479 ljdursi   18   0  108m 4816 3212 S 98.5  0.0   1:04.81  6 gameoflife
22480 ljdursi   18   0  108m 4856 3260 S 98.5  0.0   1:04.85 13 gameoflife
22482 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.83  2 gameoflife
22483 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.82  8 gameoflife
22484 ljdursi   18   0  108m 4832 3232 S 98.5  0.0   1:04.80  9 gameoflife
22481 ljdursi   18   0  108m 4856 3256 S 98.2  0.0   1:04.81  3 gameoflife
22485 ljdursi   18   0  108m 4808 3208 S 98.2  0.0   1:04.80  4 gameoflife
22478 ljdursi   18   0  117m 5724 3268 D 69.6  0.0   0:46.07 15 gameoflife
 8042 root       0 -20 2235m 1.1g  16m S  2.3  6.8   0:30.59  8 mmfsd
10735 root      15   0  3792  452  372 S  1.3  0.0   0:16.80  0 cat
```

Also, load-balance issues; one processor under utilized (~70% use as vs 98.2%)

Insert timers into regions of code

- *Instrumenting* code
- Simple, but incredibly useful
- Runs every time your code is run
- Can trivially see if changes make things better or worse

```c
    struct timeval calc;

    tick(&calc);
     /* do work */
    calctime = tock(&calc);

    printf("Timing summary:\n");
    /* other timers.. */
    printf("Calc: %8.5f\n", calctime);



void tick(struct timeval *t) {
    gettimeofday(t, NULL);
}


double tock(struct timeval *t) {
    struct timeval now;
    gettimeofday(&now, NULL);
    return (double)(now.tv_sec - t->tv_sec) +
      ((double)(now.tv_usec - t->tv_usec)/1000000.);
}
```

C

SciNet

Insert timers into regions of code

- *Instrumenting* code
- Simple, but incredibly useful
- Runs every time your code is run
- Can trivially see if changes make things better or worse

```
    integer :: calc
    real    :: calctime

    call tick(calc);
     ! do work
    calctime = tock(calc);

    print *, 'Timing summary:'
    ! other timers..
    print *, "Calc: ", calctime


subroutine tick(t)
    integer, intent(OUT) :: t
    call system_clock(t)
end subroutine tick

real function tock(t)
    integer, intent(IN) :: t
    integer :: now, clock_rate

    call system_clock(now, clock_rate)
    return real(now - t)/real(clock_rate)
end function tock
```

FORTRAN90

SciNet

Matrix-Vector multiply

- Simple mat-vec multiply
- Initializes data, does multiply, saves result
- Look to see where it spends its time, speed it up.
- Options for how to access data, output data.

```
/* initialize data */

tick(&init);
gettimeofday(&t, NULL);
seed = (unsigned int)t.tv_sec;

for (int i=0; i<size; i++) {
        x[i] = (double)rand_r(&seed)/RAND_MAX;
        y[i] = 0.;
}

if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
}
inittime = tock(&init);


/* do multiplication */

tick(&calc);
if (transpose) {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
calctime = tock(&calc);

/* Now output files */
tick(&io);
if (binoutput) {
    out = fopen("Mat-vec.dat","wb");
```

mat-vec-mult.c

/scinet/courses/profiling

Matrix-Vector multiply

- Can get an overview of the time spent easily, because we instrumented our code (~12 lines!)
- I/O huge bottleneck.

```
$ mvm --matsize=2500

Timing summary:
   Init:   0.00952 sec
   Calc:   0.06638 sec
   I/O :   5.07121 sec
```

mat-vec-mult.c

Matrix-Vector multiply

- I/O being done in ASCII
- having to loop over data, convert to string, write to output.
- 6,252,500 write operations!
- Let's try a --binary option:

```
out = fopen("Mat-vec.dat","w");
fprintf(out,"%d\n",size);

for (int i=0; i<size; i++)
    fprintf(out,"%f ", x[i]);

fprintf(out,"\n",out);

for (int i=0; i<size; i++)
    fprintf(out,"%f ", y[i]);

fprintf(out,"\n",out);

for (int i=0; i<size; i++) {
    for (int j=0; j<size; j++) {
        fprintf(out,"%f ", a[i][j]);
    }
    fprintf(out,"\n",out);
}
fclose(out);
```

mat-vec-mult.c

Matrix-Vector multiply

- Let's try a --binary option:
- Shorter...

```
out = fopen("Mat-vec.dat","wb");
fwrite(&size, sizeof(int),   1,          out);
fwrite(x,     sizeof(float), size,       out);
fwrite(y,     sizeof(float), size,       out);
fwrite(&(a[0][0]),     sizeof(float), size*size, out);
fclose(out);
```

mat-vec-mult.c

SciNet

Binary I/O

- Much (36x!) faster.
- And ~4x smaller.
- Still slow, but writing to disk is slower than a multiplication.
- On to Calc..

```
$ mvm --matsize=2500
--binary

Timing summary:
   Init:   0.00976 sec
   Calc:   0.06695 sec
   I/O :   0.14218 sec
```

```
$ ./mvm --binary
$ du -h Mat-vec.dat
89M     Mat-vec.dat

$ ./mvm --binary
$ du -h Mat-vec.dat
20M     Mat-vec.dat
```

SciNet

Sampling for Profiling

- How to get finer-grained information about where time is being spent?
- Can't instrument every single line.
- Compilers have tools for *sampling* execution paths.

SciNet

Program Counter Sampling

- As program executes, every so often (~100ms) a timer goes off, and the current location of execution is recored
- Shows where time is being spent.

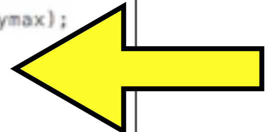
```c
    case SIM_PROJECTILE:
        ymin = xmin = 0.;
        ymax = xmax = 1.;
        dx = (xmax-xmin)/npts;
        dy = (ymax-ymin)/npts;
        init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
        projectile_initvalues(&d, psize, pdens, pvel);
        outputvar = DENSVAR;
        break;
}

/* apply boundary conditions and make thermodynamically consistant */
bcs[0] = xbc; bcs[1] = xbc;
bcs[2] = ybc; bcs[3] = ybc;
apply_all_bcs(&d,bcs);
domain_backward_dp_eos(&d);
domain_ener_internal_to_tot(&d);


/* main loop */

tick(&tt);
if (output) domain_plot(&d);
printf("Step\tdt\ttime\n");
for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

    printf("%d\t%g\t%g\n", step, dt, time);

    if (output && ((step % outevery) == 0) ) {
        sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
        sprintf(binfilename,"dens_test_%d.bin", outnum);
        sprintf(h5filename,"dens_test_%d.h5", outnum);
        sprintf(ncdffilename,"dens_test_%d.nc", outnum);
        domain_output_ppm(&d, outputvar, ppmfilename);
        domain_output_bin(&d, binfilename);
        domain_output_hdf5(&d, h5filename);
        domain_output_netcdf(&d, ncdffilename);
        domain_plot(&d);
        outnum++;
    }
    kl_timestep_xy(&d, bcs, dt);
    apply_all_bcs(&d,bcs);

    kl_timestep_yx(&d, bcs, dt);
    apply_all_bcs(&d,bcs);
}
tock(&tt);
```

SciNet

Program Counter Sampling

- Advantages:
  - Very low overhead
  - No extra instrumentation
- Disadvantages:
  - Don’t know why code is there
  - Statistics - have to run long enough job

```c
    case SIM_PROJECTILE:
        ymin = xmin = 0.;
        ymax = xmax = 1.;
        dx = (xmax-xmin)/npts;
        dy = (ymax-ymin)/npts;
        init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
        projectile_initvalues(&d, psize, pdens, pvel);
        outputvar = DENSVAR;
        break;
}

/* apply boundary conditions and make thermodynamically consistant */
bcs[0] = xbc; bcs[1] = xbc;
bcs[2] = ybc; bcs[3] = ybc;
apply_all_bcs(&d,bcs);
domain_backward_dp_eos(&d);
domain_ener_internal_to_tot(&d);


/* main loop */

tick(&tt);
if (output) domain_plot(&d);
printf("Step\tdt\ttime\n");
for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

    printf("%d\t%g\t%g\n", step, dt, time);

    if (output && ((step % outevery) == 0) ) {
        sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
        sprintf(binfilename,"dens_test_%d.bin", outnum);
        sprintf(h5filename,"dens_test_%d.h5", outnum);
        sprintf(ncdffilename,"dens_test_%d.nc", outnum);
        domain_output_ppm(&d, outputvar, ppmfilename);
        domain_output_bin(&d, binfilename);
        domain_output_hdf5(&d, h5filename);
        domain_output_netcdf(&d, ncdffilename);
        domain_plot(&d);
        outnum++;
    }
    kl_timestep_xy(&d, bcs, dt);
    apply_all_bcs(&d,bcs);

    kl_timestep_yx(&d, bcs, dt);
    apply_all_bcs(&d,bcs);
}
tock(&tt);
```



SciNet

gprof for sampling

```
$ gcc -O3 -pg -g mat-vec-mult.c --std=c99
$ icc -O3 -pg -g mat-vec-mult.c -c99
```

turn on profiling

debugging symbols (optional, but more info)

```
$ ./mvm-profile --matsize=2500
[output]
$ ls
Makefile  Mat-vec.dat  gmon.out
mat-vec-mult.c   mvm-profile
```

SciNet

gprof examines gmon.out

```
$ gprof mvm-profile gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.24      0.41     0.41                             main
  0.00      0.41     0.00        3     0.00     0.00  tick
  0.00      0.41     0.00        3     0.00     0.00  tock
  0.00      0.41     0.00        2     0.00     0.00  alloc1d
  0.00      0.41     0.00        2     0.00     0.00  free1d
  0.00      0.41     0.00        1     0.00     0.00  alloc2d
  0.00      0.41     0.00        1     0.00     0.00  free2d
  0.00      0.41     0.00        1     0.00     0.00  get_options
[...]
```

Gives data by function -- usually handy, not so useful in this toy problem

SciNet

gprof --line examines gmon.out by line

```
gpc-f103n084-$ gprof --line mvm-profile gmon.out  | more
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 68.46      0.28     0.28                             main (mat-vec-mult.c:82 @ 401
 14.67      0.34     0.06                             main (mat-vec-mult.c:113 @ 40
  7.33      0.37     0.03                             main (mat-vec-mult.c:63 @ 401
  4.89      0.39     0.02                             main (mat-vec-mult.c:112 @ 40
  4.89      0.41     0.02                             main (mat-vec-mult.c:113 @ 40
  0.00      0.41     0.00        3     0.00     0.00  tick (mat-vec-mult.c:159 @ 40
  0.00      0.41     0.00        3     0.00     0.00  tock (mat-vec-mult.c:164 @ 40
  0.00      0.41     0.00        2     0.00     0.00  alloc1d (mat-vec-mult.c:152 @
  0.00      0.41     0.00        2     0.00     0.00  free1d (mat-vec-mult.c:171 @
  0.00      0.41     0.00        1     0.00     0.00  alloc2d (mat-vec-mult.c:130 @
  0.00      0.41     0.00        1     0.00     0.00  free2d (mat-vec-mult.c:144 @
  0.00      0.41     0.00        1     0.00     0.00  get_options (mat-vec-mult.c:1
400a30)
```

SciNet

Then can compare to source

- Code is spending most time deep in loops
- #1 - multiplication
- #2 - I/O (old way)

```
80          for (int j=0; j<size; j++) {
81              for (int i=0; i<size; i++) {
82                  y[i] += a[i][j]*x[j];      <--
83              }
84          }
                          ...
98          out = fopen("Mat-vec.dat","w");
99          fprintf(out,"%d\n",size);
100
101         for (int i=0; i<size; i++)
102             fprintf(out,"%f ", x[i]);
103
104         fprintf(out,"\n");
105
106         for (int i=0; i<size; i++)
107             fprintf(out,"%f ", y[i]);
108
109         fprintf(out,"\n");
110
111         for (int i=0; i<size; i++) {
112             for (int j=0; j<size; j++) {
113                 fprintf(out,"%f ", a[i][j]);      <--
114             }
115             fprintf(out,"\n");
116         }
117         fclose(out);
```

SciNet

gprof pros/cons

- Exists everywhere
- Easy to script, put in batch jobs
- Low overhead
- Works well with multiple processes - thread data all gets clumped together
- 1 file per proc (good for small #s, but hard to compare)

SciNet

Open|Speedshop

- GUI containing several different ways of doing performance experiments
- Includes pcsamp (like gprof - by function), usertime (by line of code and callgraph), I/O tracing, MPI tracing.
- Can run either in a sampling mode, or instrumenting/tracing ('online' mode - automatically instruments the binary).

SciNet

Open|Speedshop

Open|SpeedShop

File Tools Help

pc Sampling [4] | Source Panel [0]

Process Control

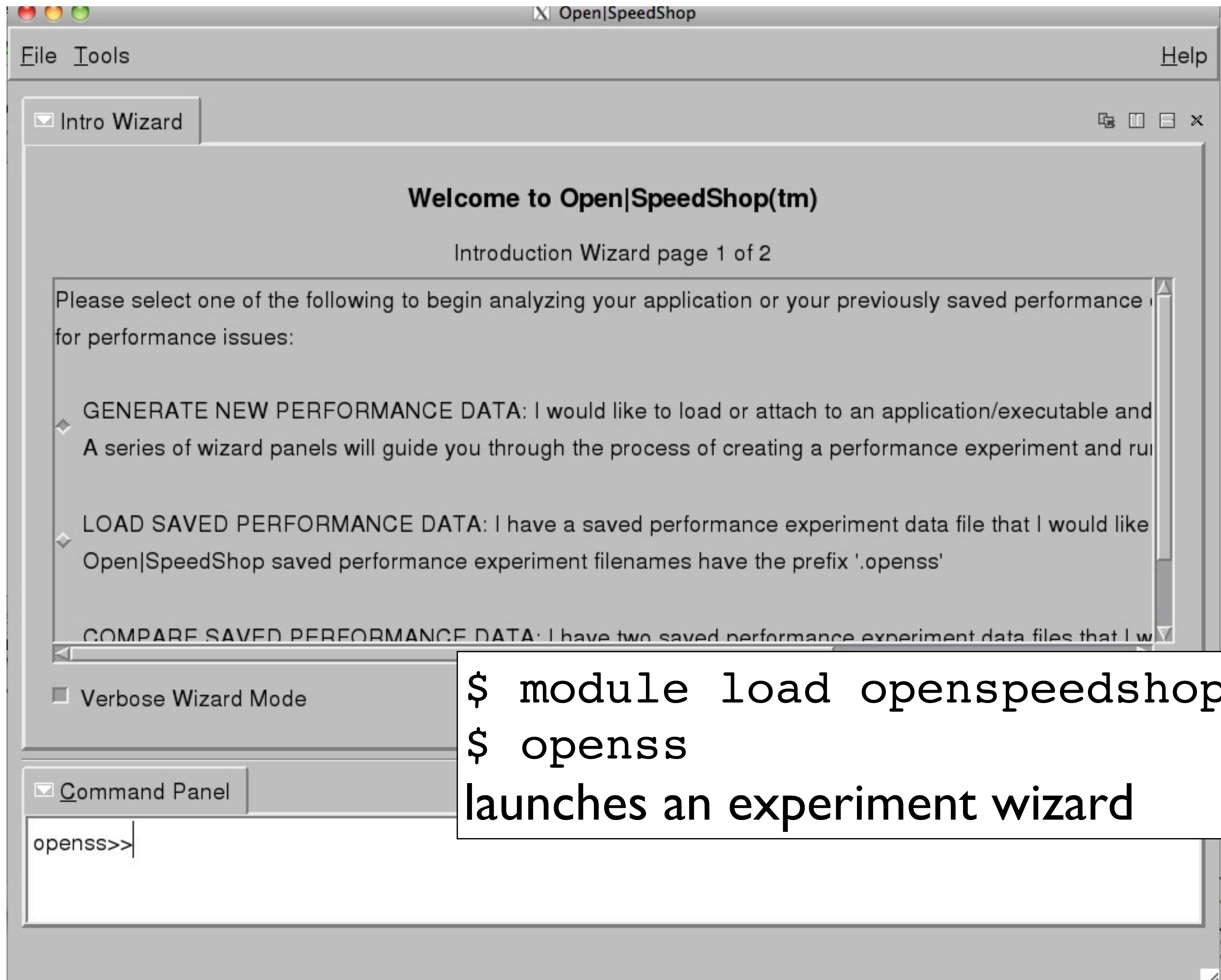
Run Cont Pause Update Terminate

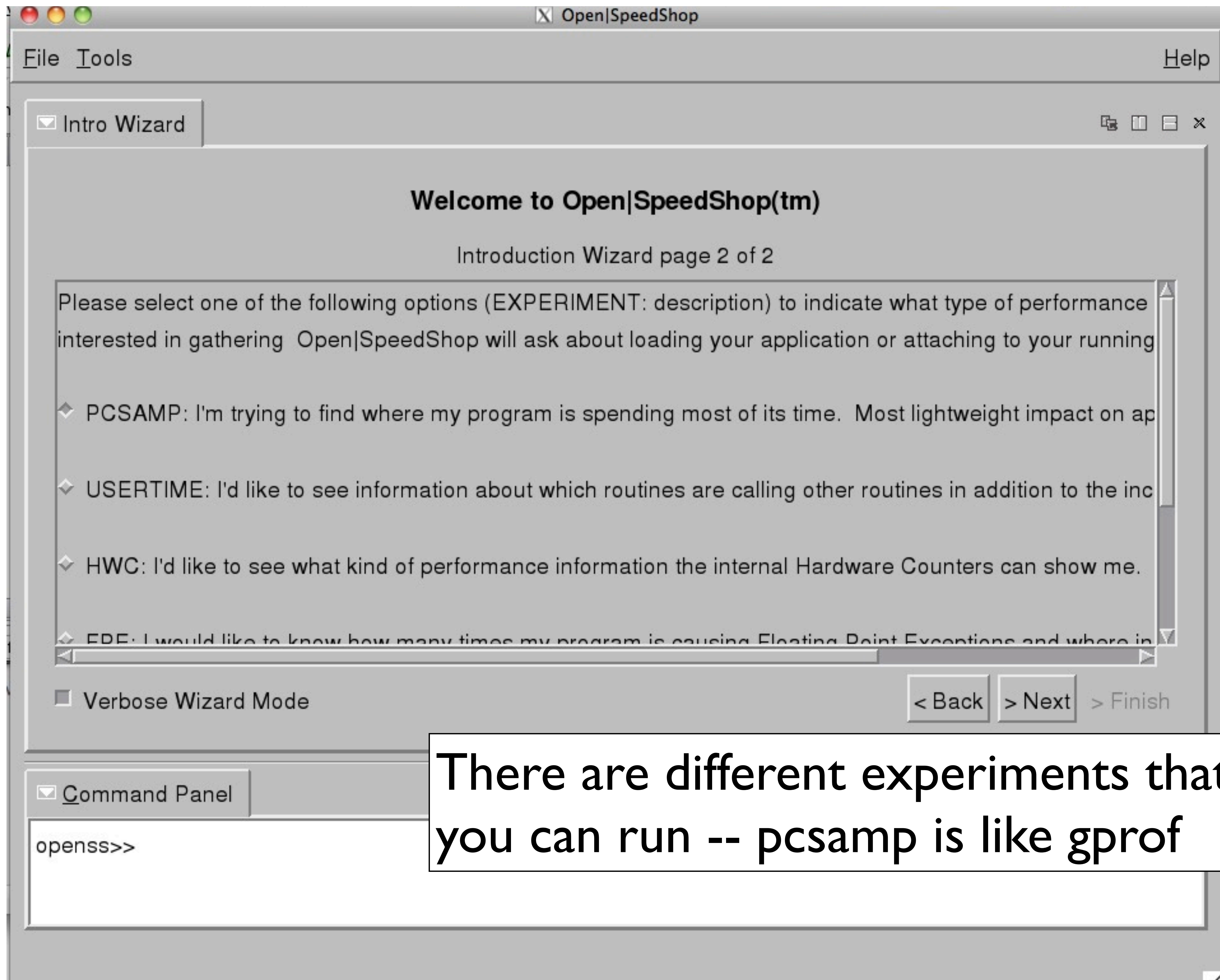
Status: Loaded saved data from file /scratch/ljdursi/Testing/profiling/mat-vec/mvm-pcsamp-4.openss.

Stats Panel [4] | ManageProcessesPanel [4] | Source Panel [4]

Exclusive C /scratch/ljdursi/Testing/profiling/mat-vec/mat-vec-mult.c

```
            80      for (int j=0; j<size; j++) {
0.020000    81          for (int i=0; i<size; i++) {
>> 1.42000  82              y[i] += a[i][j]*x[j];
            83          }
            84      }
            85  }
            86  calctime = tock(&calc);
            87
            88  /* Now output files */
            89  tick(&io);
            90  if (binoutput) {
```


Open|SpeedShop
File
Tools
Help
Intro Wizard
Welcome to Open|SpeedShop(tm)
Introduction Wizard page 1 of 2
Please select one of the following to begin analyzing your application or your previously saved performance
for performance issues:
GENERATE NEW PERFORMANCE DATA: I would like to load or attach to an application/executable and
A series of wizard panels will guide you through the process of creating a performance experiment and ru
LOAD SAVED PERFORMANCE DATA: I have a saved performance experiment data file that I would like
Open|SpeedShop saved performance experiment filenames have the prefix '.openss'
Verbose Wizard Mode
Command Panel
openss>>
$ module load openspeedshop
$ openss
launches an experiment wizard


Open|SpeedShop
File Tools
Help
Intro Wizard
Welcome to Open|SpeedShop(tm)
Introduction Wizard page 2 of 2
Please select one of the following options (EXPERIMENT: description) to indicate what type of performance
interested in gathering Open|SpeedShop will ask about loading your application or attaching to your running
PCSAMP: I'm trying to find where my program is spending most of its time. Most lightweight impact on ap
USERTIME: I'd like to see information about which routines are calling other routines in addition to the inc
HWC: I'd like to see what kind of performance information the internal Hardware Counters can show me.
Verbose Wizard Mode
< Back
> Next
> Finish
Command Panel
openss>>
There are different experiments that you can run -- pcsamp is like gprof

Open|SpeedShop

File Tools Help

pc Sampling [4] | Source Panel [0]

Process Control

Run | Cont | Pause | Update | Terminate

Status: Loaded saved data from file /scratch/ljdursi/Testing/profiling/mat-vec/mvm-pcsamp-4.openss.

Stats Panel [4] | ManageProcessesPanel [4] | Source Panel [4]

Showing Statements Report...

View/Display Choice: Functions | Statements | Linked Objects

Executables: mvm Host: gpc-f103n084 Pid/Rank/Thread: 47700862966512

% of CPU Time: 77.173913, 11.413043, 9.239130, 1.086957

Exclusive CPU time	% of CPU Time	Statement Location (Line Number)
1.420000	77.173913	mat-vec-mult.c(82)
0.210000	11.413043	mat-vec-mult.c(63)
0.170000	9.239130	mat-vec-mult.c(62)
0.020000	1.086957	mat-vec-mult.c(81)
0.010000	0.543478	interp.c(0)

Command Panel

openss>>

It will show top functions (or statements) by default; double-clicking takes to source line.

Open|SpeedShop

File Tools Help

pc Sampling [4] | Source Panel [0]

Process Control

Run Cont Pause Update Terminate

Status: Loaded saved data from file /scratch/ljdursi/Testing/profiling/mat-vec/mvm-pcsamp-4.openss.

Stats Panel [4] | ManageProcessesPanel [4] | Source Panel [4]

Exclusive C /scratch/ljdursi/Testing/profiling/mat-vec/mat-vec-mult.c

```
            80        for (int j=0; j<size; j++) {
0.020000    81            for (int i=0; i<size; i++) {
>> 1.42000  82                y[i] += a[i][j]*x[j];
            83            }
            84        }
            85    }
            86    calctime = tock(&calc);
            87
            88    /* Now output files */
            89    tick(&io);
            90    if (binoutput) {
```

SciNet

Open|SpeedShop

File Tools

Compare Experiments [5]

Status: Experiment 5 is being compared with experiment 7

Stats Panel [5] | Source Panel [5]

Showing Comparison Report...

Executables: mvm

View consists of comparison columns click on the metadata icon "I" for details.

-c 6, Exclusive CPU	-c 8, Exclusive CPU	Statement Location (Line Number)
0.060000	0.000000	mat-vec-mult.c(82)
0.000000	0.010000	interp.c(0)
0.000000	0.010000	mat-vec-mult.c(74)

Command Panel

openss>>

It will also let you compare experiments. Here we try two ways of doing the matrix multiplication; the first (line 82) requires .06 seconds, the second (line 74) requires only 0.01 -- a 6x speedup!

Compare Experiments [5]

Status: Experiment 5 is being compared with experiment 7

Stats Panel [5] | Source Panel [5]

, Exclus /scratch/ljdursi/Testing/profiling/mat-vec/mat-vec-mult.c

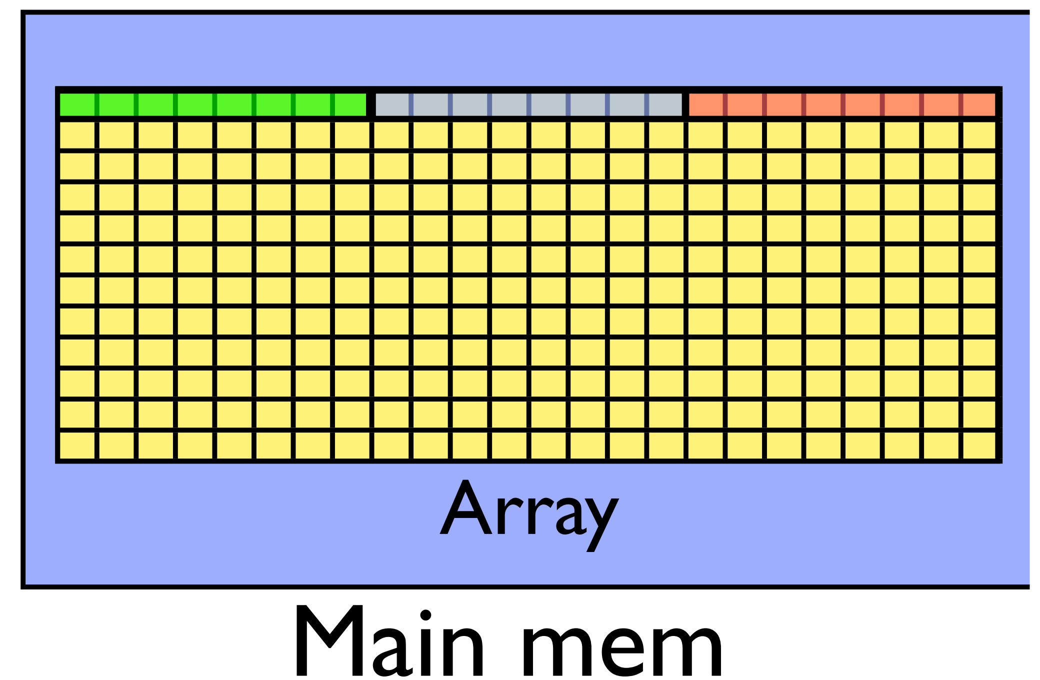
```
         70  tick(&calc);
         71  if (transpose) {
         72      #pragma omp parallel for default(none) shared(x,y,a,size)
         73      for (int i=0; i<size; i++) {
0.01000  74          for (int j=0; j<size; j++) {
         75              y[i] += a[i][j]*x[j];
         76          }
         77      }
         78  } else {
         79      #pragma omp parallel for default(none) shared(x,y,a,size)
         80      for (int j=0; j<size; j++) {
         81          for (int i=0; i<size; i++) {
         82              y[i] += a[i][j]*x[j];
         83          }
```

SciNet

Cache Thrashing

Cache

- Memory bandwidth is key to getting good performance on modern systems
- Main Mem - big, slow
- Cache - small, fast
  - Saves recent accesses, a line of data at a time



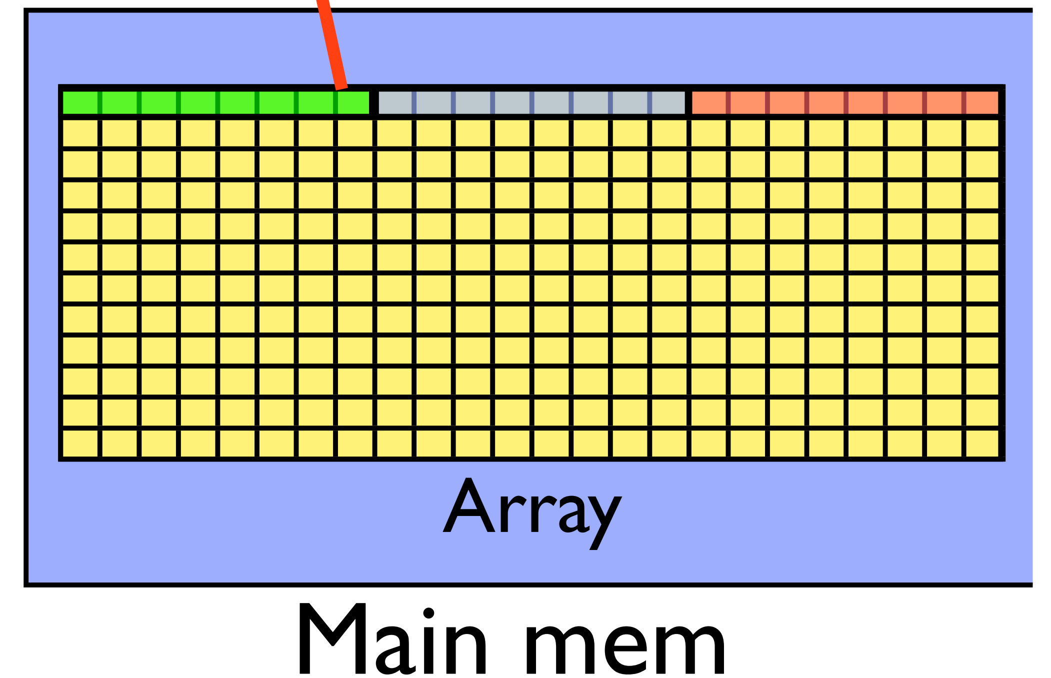
Main mem

SciNet

Cache Thrashing

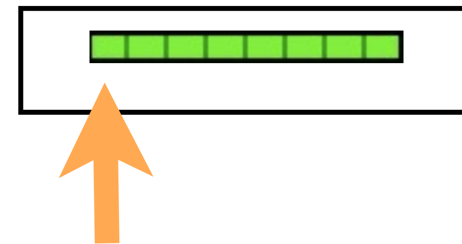
Cache

- When accessing memory in order, only one access to slow main mem for many data points
- Much faster

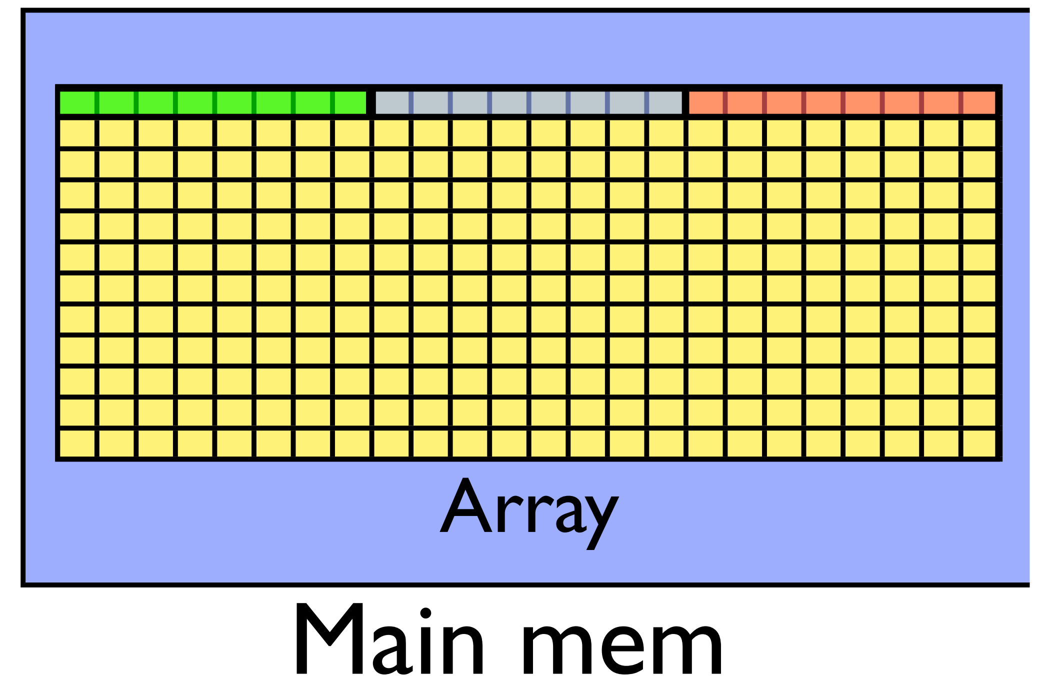


SciNet

Cache Thrashing

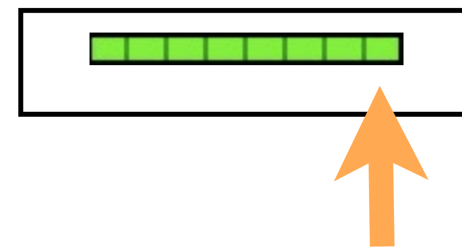


- When accessing memory in order, only one access to slow main mem for many data points
- Much faster

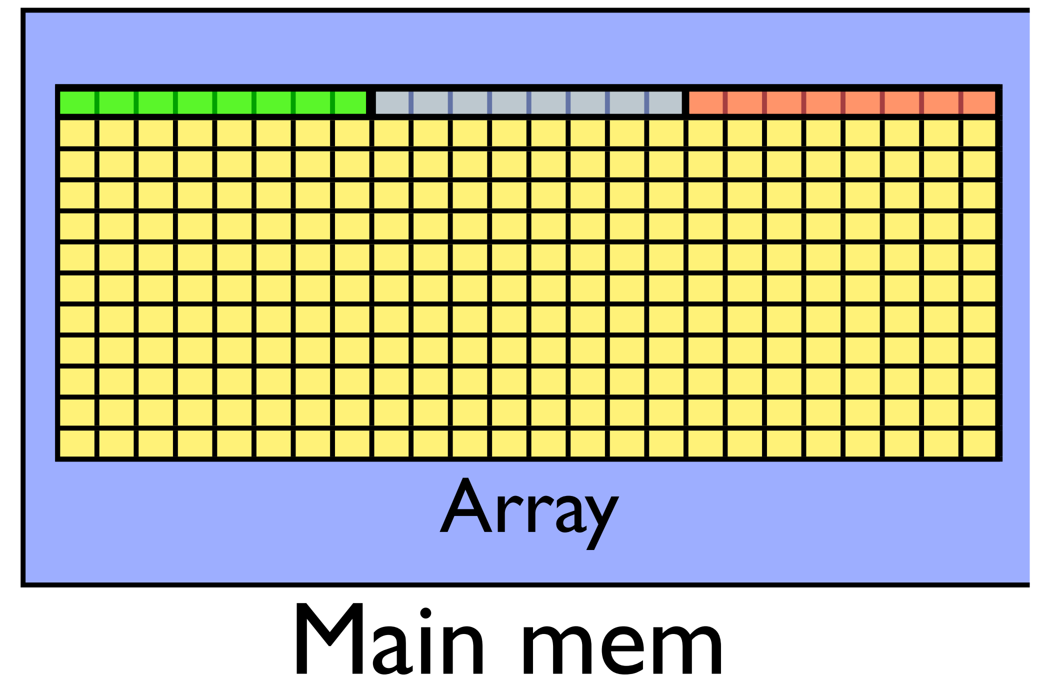


SciNet

Cache Thrashing



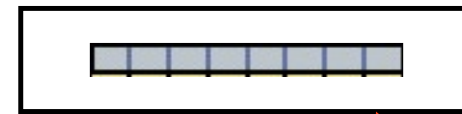
- When accessing memory in order, only one access to slow main mem for many data points
- Much faster



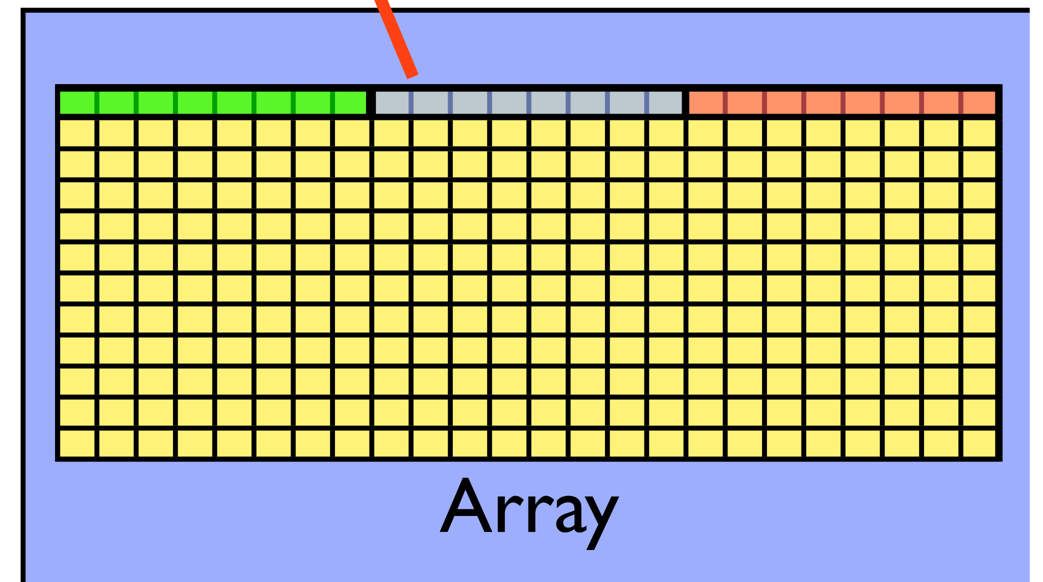
SciNet

Cache Thrashing

Cache



- When accessing memory in order, only one access to slow main mem for many data points
- Much faster

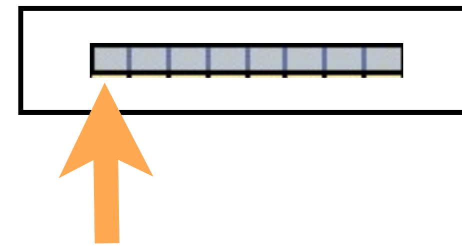


Main mem

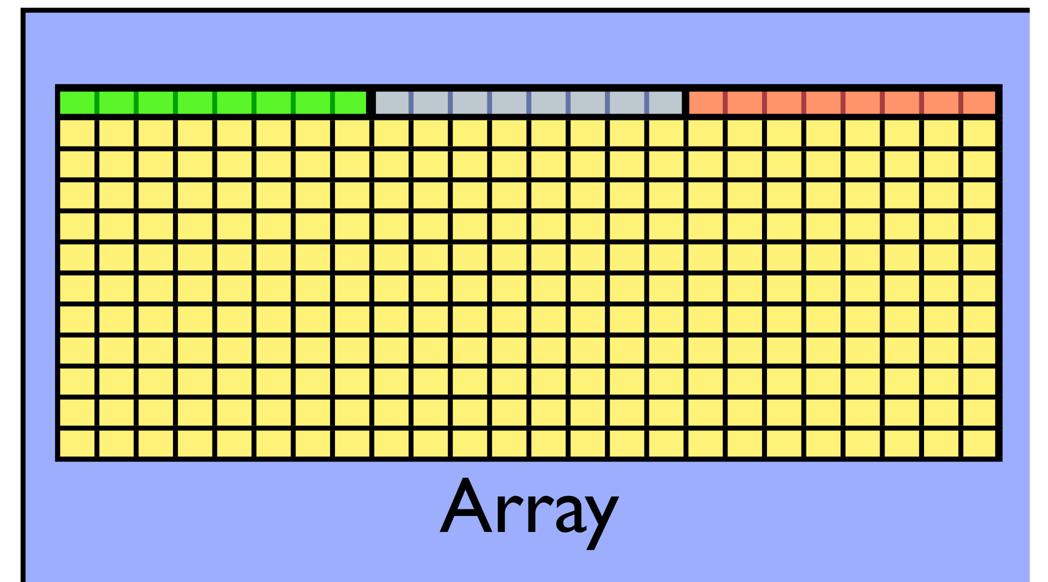
SciNet

Cache Thrashing

Cache



- When accessing memory in order, only one access to slow main mem for many data points
- Much faster

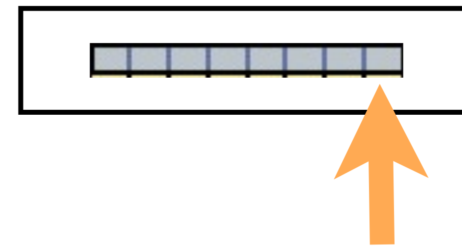


Main mem

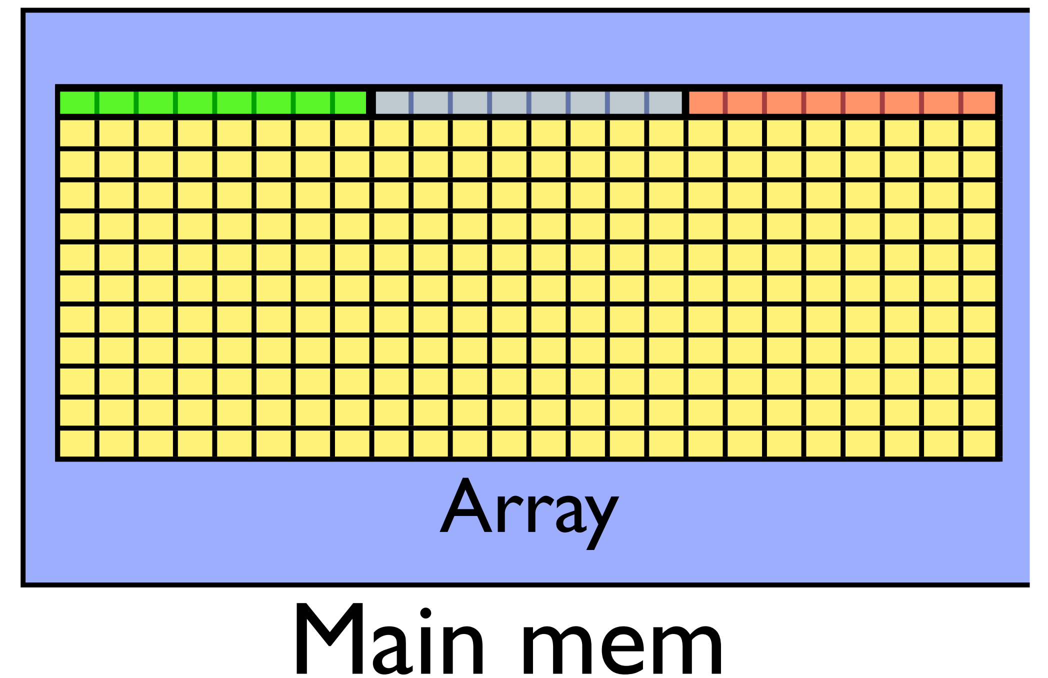
SciNet

Cache Thrashing

Cache



- When accessing memory in order, only one access to slow main mem for many data points
- Much faster



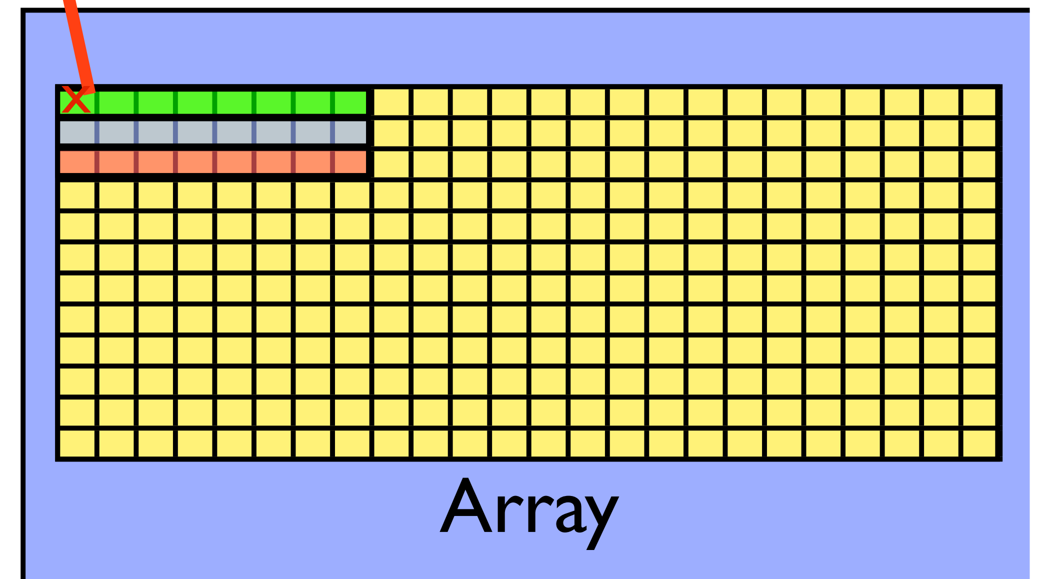
Main mem

SciNet

Cache Thrashing

Cache

- When accessing memory out of order, much worse
- Each access is new cache line (cache miss)- slow access to main memory

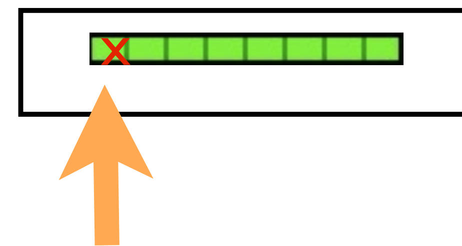


Main mem

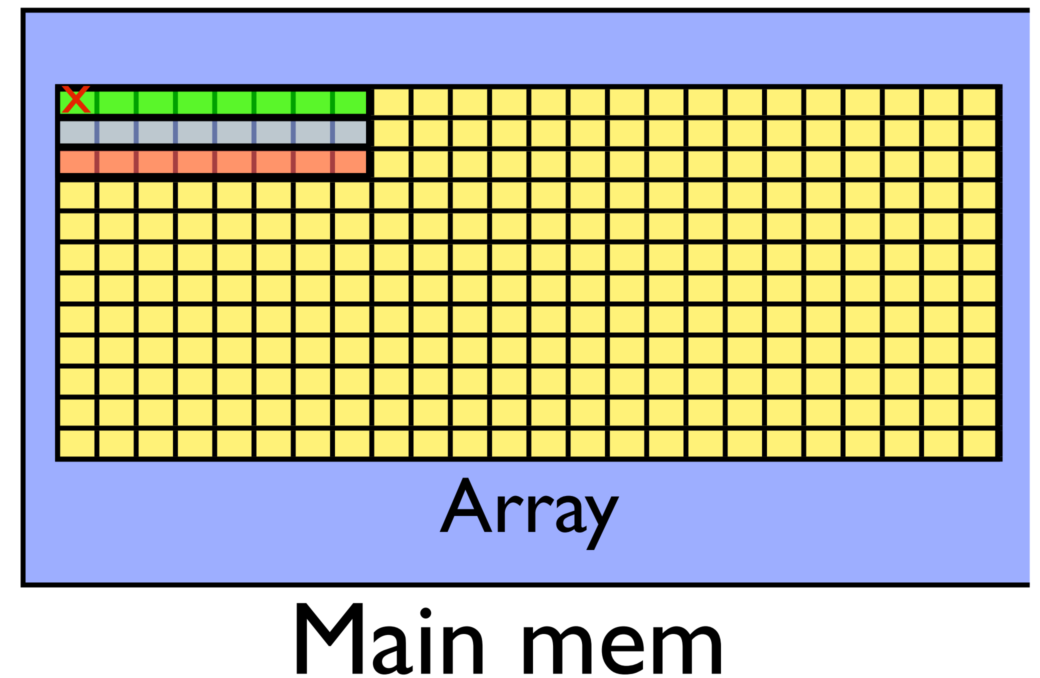
SciNet

Cache Thrashing

Cache



- When accessing memory out of order, much worse
- Each access is new cache line (cache miss)- slow access to main memory

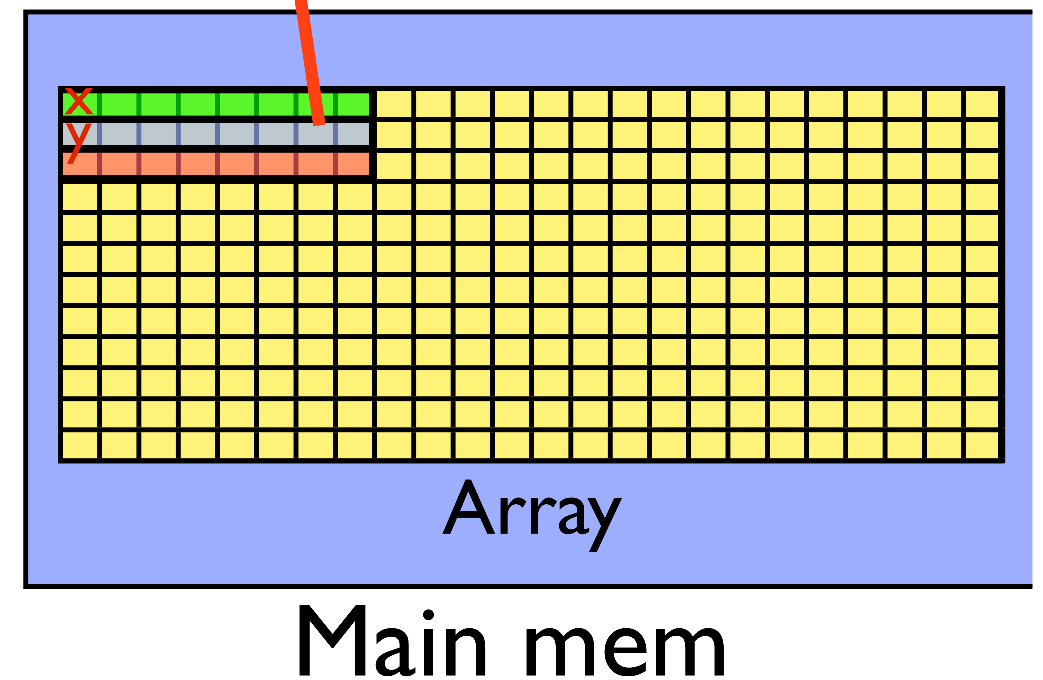


SciNet

Cache Thrashing

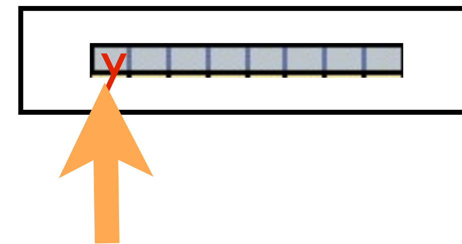
Cache

- When accessing memory out of order, much worse
- Each access is new cache line (cache miss)- slow access to main memory

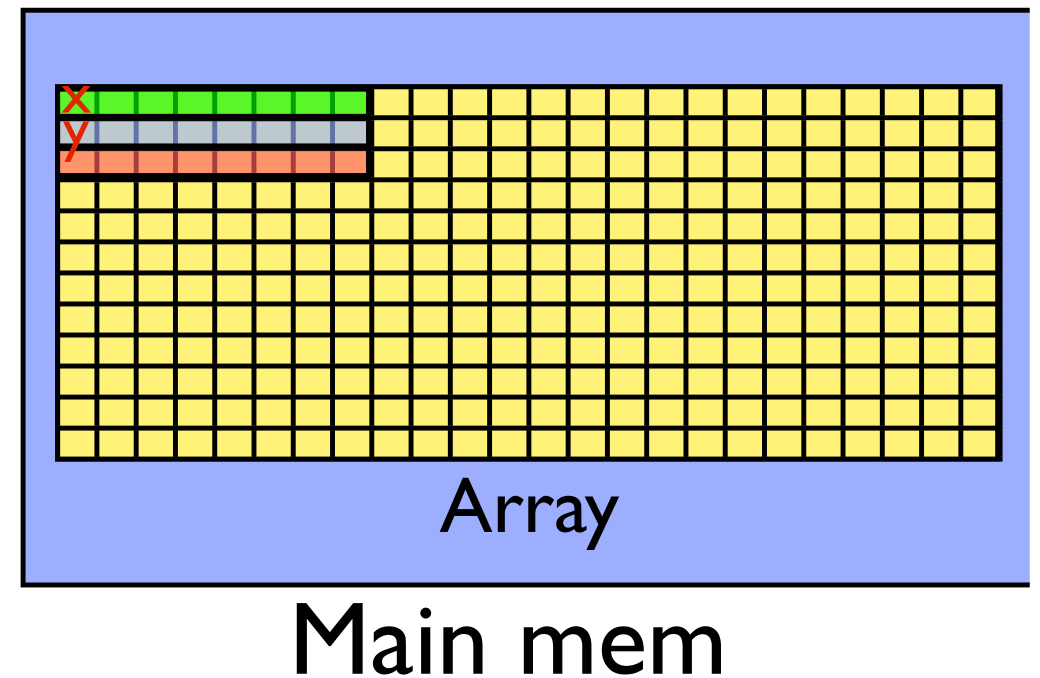


SciNet

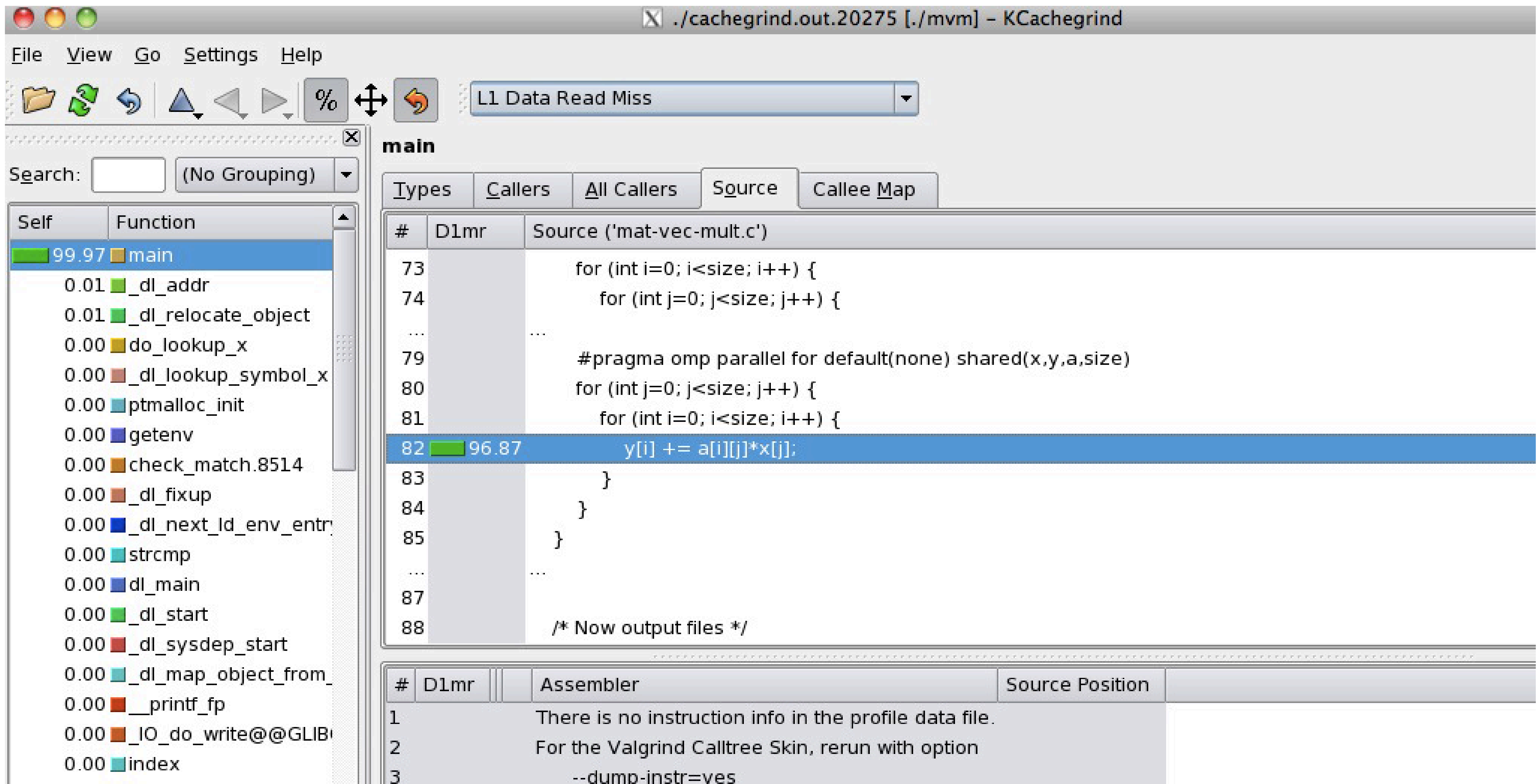
Cache Thrashing



- When accessing memory out of order, much worse
- Each access is new cache line (cache miss)- slow access to main memory



SciNet



kcachegrind viewing output of

```
$ module load valgrind
$ valgrind --tool=cachegrind ./mvm --matsize=2500
$ kcachegrind cachegrind.out.20275
```

Cache Trashing

- In C, cache-friendly order is to make last index most quickly varying

Good

```
tick(&calc);
if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
calctime = tock(&calc);
```

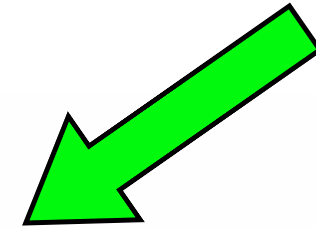
Bad

SciNet

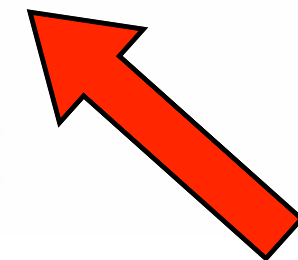
Cache Trashing

- In Fortran, cache-friendly order is to make first index most quickly varying...
- or in this case, just use `matmul`

Good



```
call tick(calc)
if (transpose) then
    do j=1,size
        do i=1,size
            y(i,j) = a(i,j)*x(j)
        enddo
    enddo
else
    do i=1,size
        do j=1,size
            y(i,j) = a(i,j)*x(j)
        enddo
    enddo
endif
calctime = tock(calc)
```



Bad

SciNet

```
gpc-f103n084-$ export OMP_NUM_THREADS=1
gpc-f103n084-$ ./mvm-omp --matsize=2500 --transpose --binary
Timing summary:
    Init:   0.00947 sec
    Calc:   0.00811 sec
    I/O :   0.14881 sec

gpc-f103n084-$ export OMP_NUM_THREADS=2
gpc-f103n084-$ ./mvm-omp --matsize=2500 --transpose --binary
Timing summary:
    Init:   0.00986 sec
    Calc:   0.00445 sec
    I/O :   0.01558 sec
```

Once cache thrashing is fixed (by transposing the order of the loops), OpenMPing the loop works fairly well -- but now initialization is a bottleneck. (Amdahl's law)

Tuning is iterative!

SciNet

Stats Panel [9] | ManageProcessesPanel [9] | Source Panel [9]

Showing Load Balance (min,max,ave) Report:

View/Display Choice: Functions · Statements · Linked Objects

Executables: mvm Host: gpc-f103n084 Pid/Rank/Thread: 47974948653808

Max Exclusive CPU	Posix ThreadId of M	Min Exclusive CPU	Posix ThreadId of M	Average Exclusive	Statement Location (Line Number)
0.070000	47974948653808	0.070000	47974948653808	0.070000	mat-vec-mult.c(63)
0.050000	47974948653808	0.050000	47974948653808	0.050000	mat-vec-mult.c(75)
0.020000	47974948653808	0.020000	47974948653808	0.020000	mat-vec-mult.c(74)
0.010000	47974948653808	0.010000	47974948653808	0.010000	interp.c(0)

Under Load Balance Overview, can also give top lines and their min/average/max time spent by thread. Good measure of load balance -- underused threads? Here, all #s equal -- very good load balance

Open|Speedshop

- Also has very powerful UNIX command line tools "openss -f `./mvm --transpose' pcsamp" and python scripting interface.
- Experiments: pcsamp (gprof), usertime (includes call graph), iot (I/O tracing - find out where I/O time is being spent), mpit (MPI tracing)

SciNet

Game of Life

- Simple MPI implementation of Conway game of life
- Live cell with 2,3 neighbours lives; 0-1starves; 4+ dies of overcrowding
- Empty cell w/ 3 neighbours becomes live

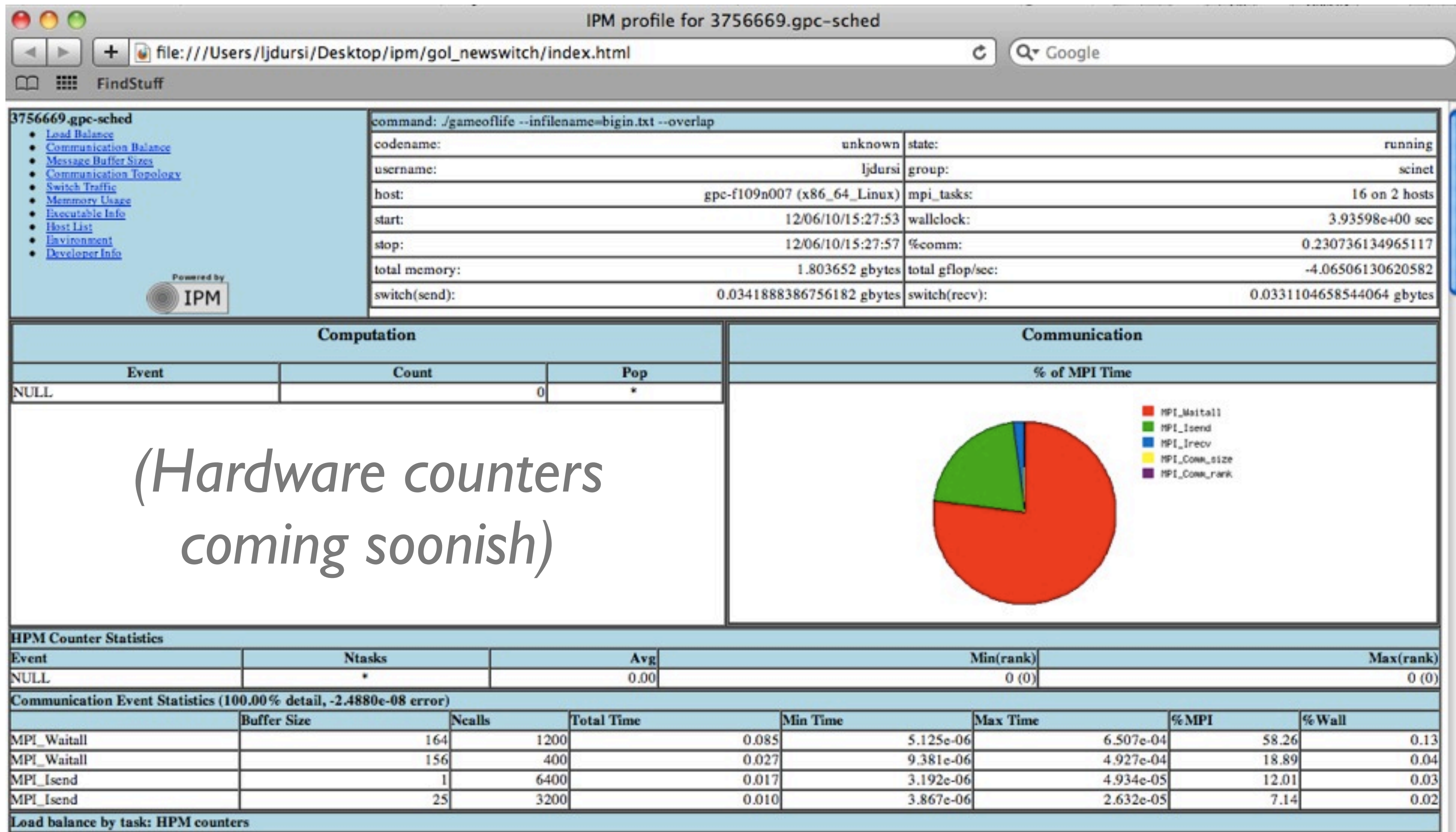
SciNet

IPM

- Integrated Performance Monitor
- Integrates a number of low-overhead counters for performance measurements of parallel codes (particularly MPI)
- Only installed for gcc+openmpi for now

```
$ module load ipm
$ export LD_PRELOAD=${SCINET_IPM_LIB}/libipm.so
$ mpirun ./gameoflife --infilename=bigin.txt
[generates big file with ugly name]
$ export LD_PRELOAD=
$ ipm_parse -html [uglyname]
```

SciNet

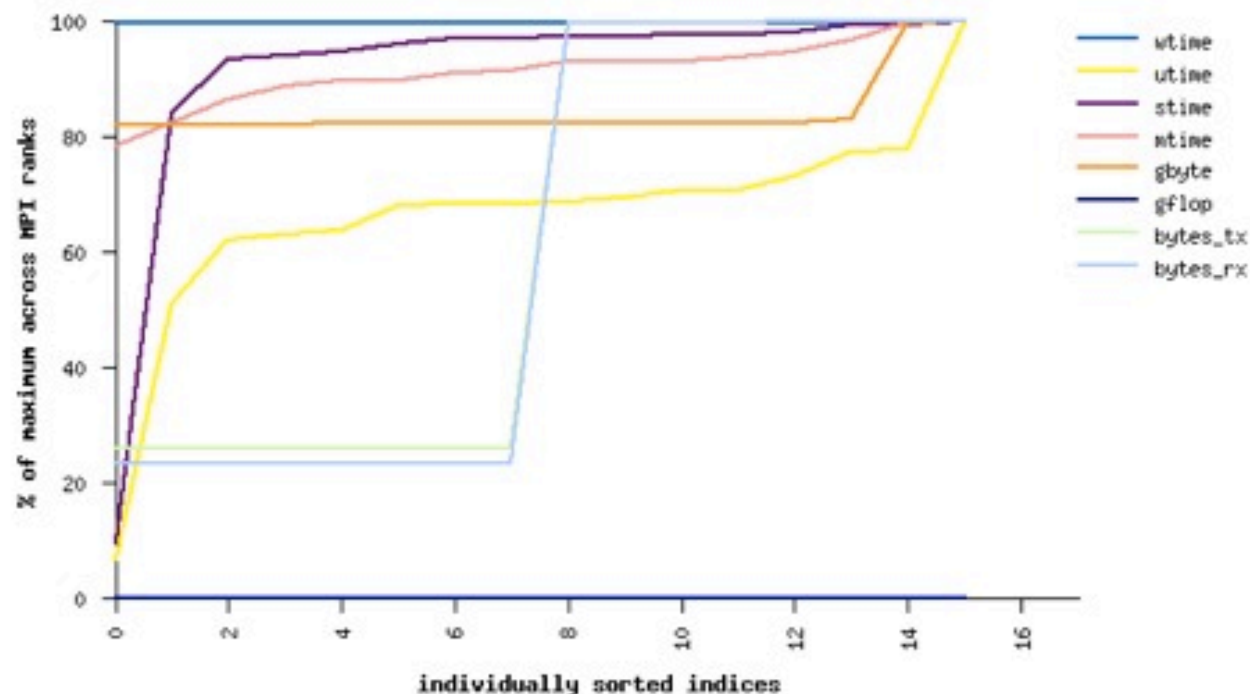


Overview: global stats, % of MPI time by call, buffer size

SciNet

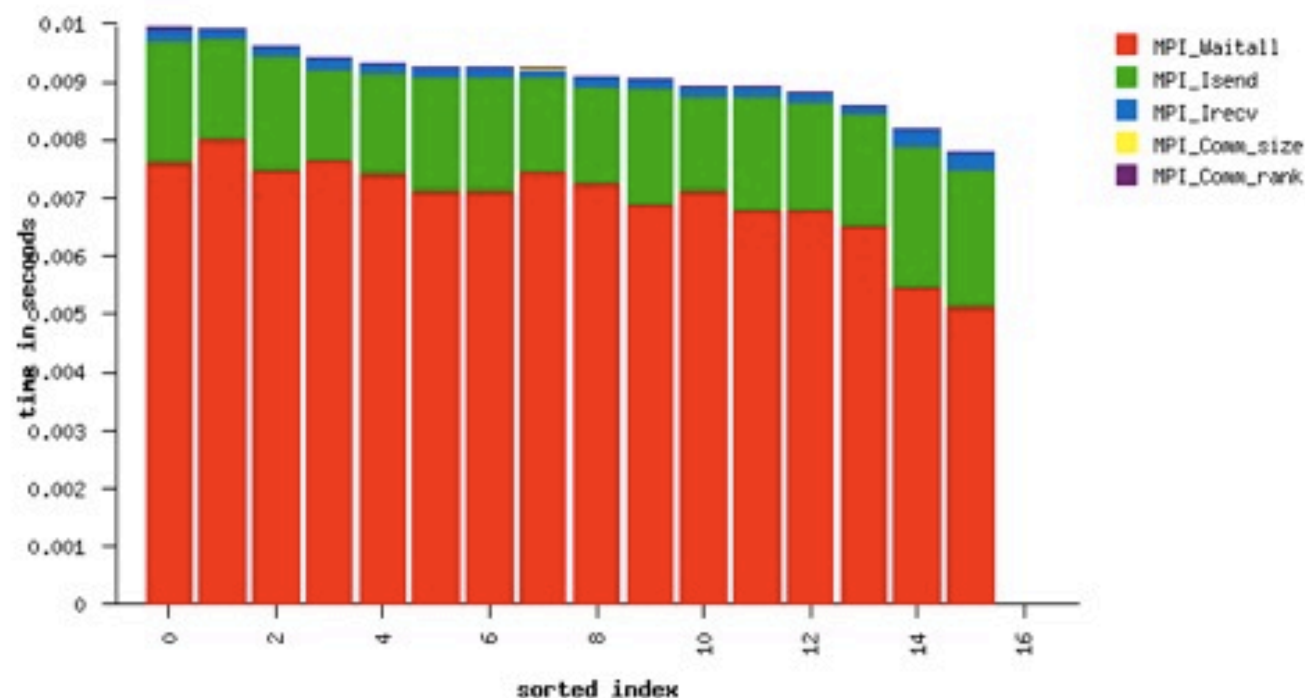
by MPI rank, by MPI time

Load balance by task: memory, flops, timings



by MPI rank, by MPI time

Communication balance by task (sorted by MPI time)

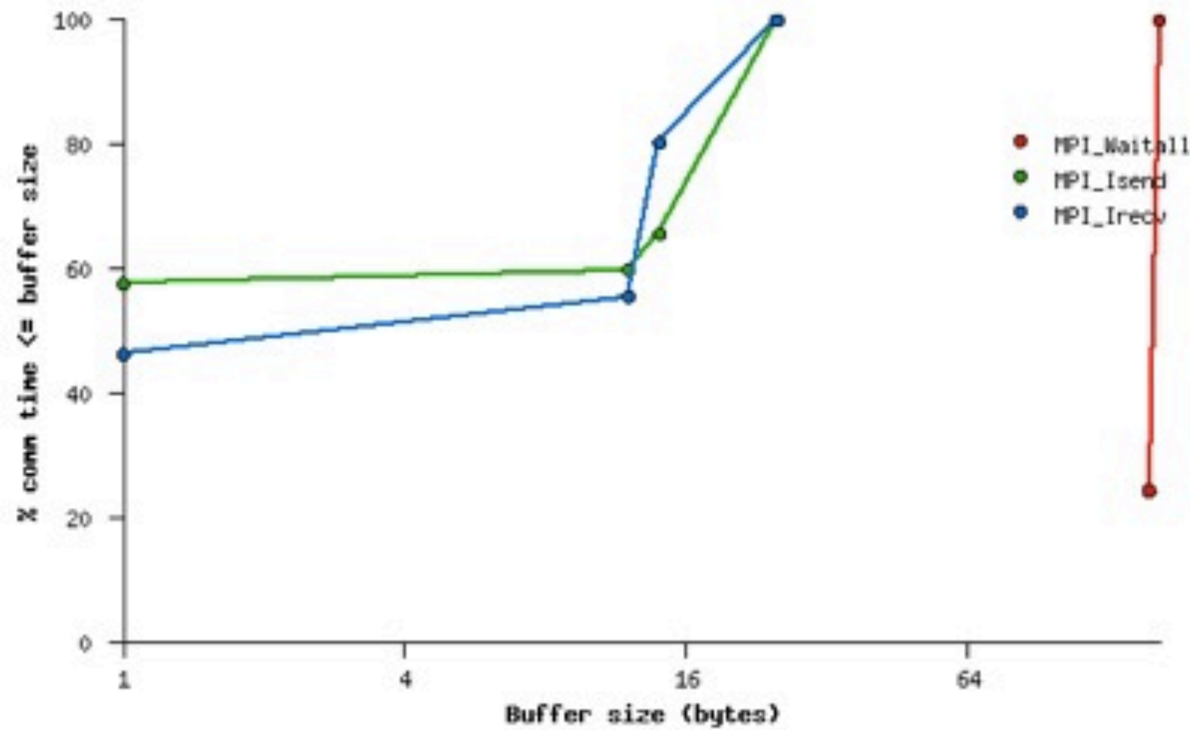


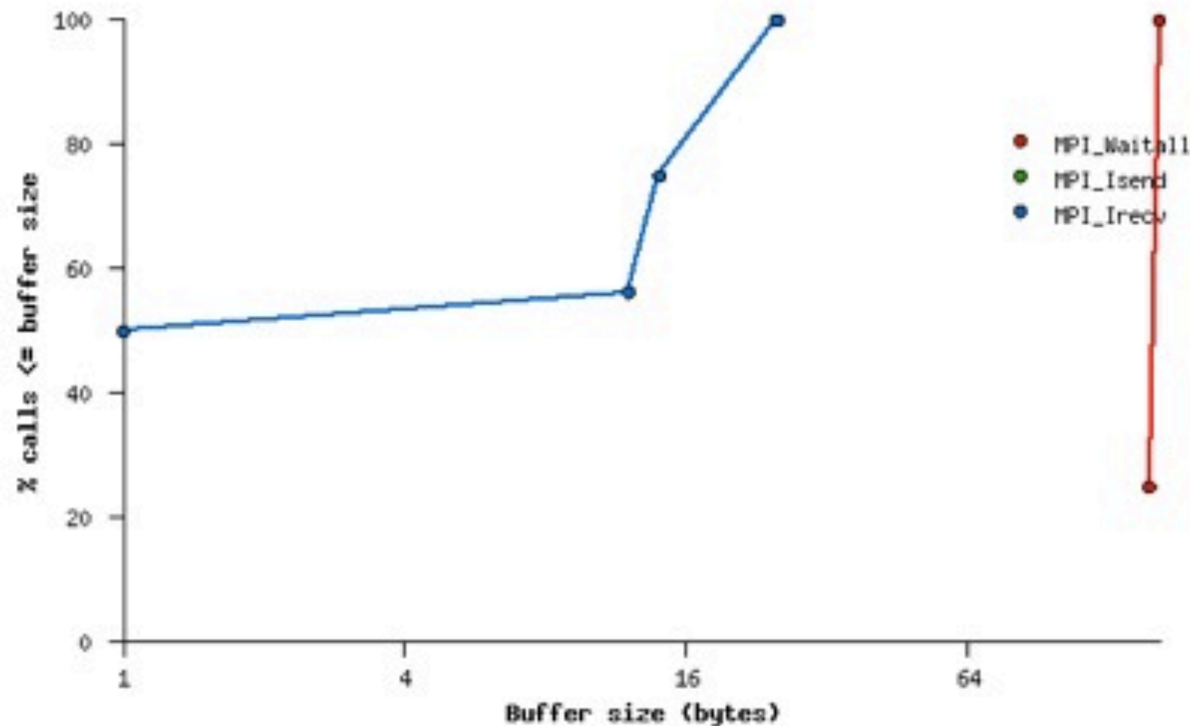
by MPI rank , time detail by MPI time , time detail by rank , call list

Message Buffer Size Distributions: time

Load balance view:
Are all tasks doing same amount of work?

SciNet

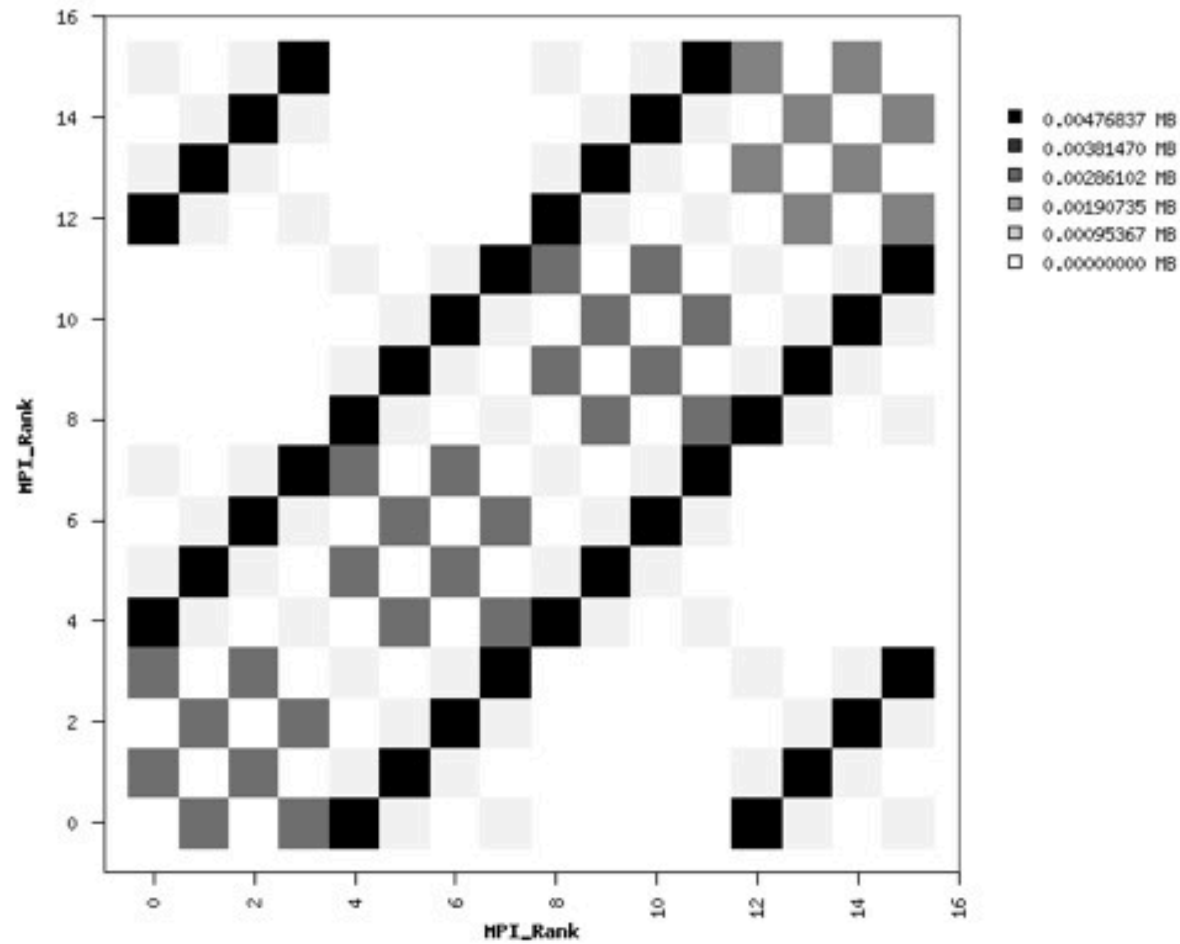




Distribution of time, # of calls by buffer size (here -- all very small messages!)

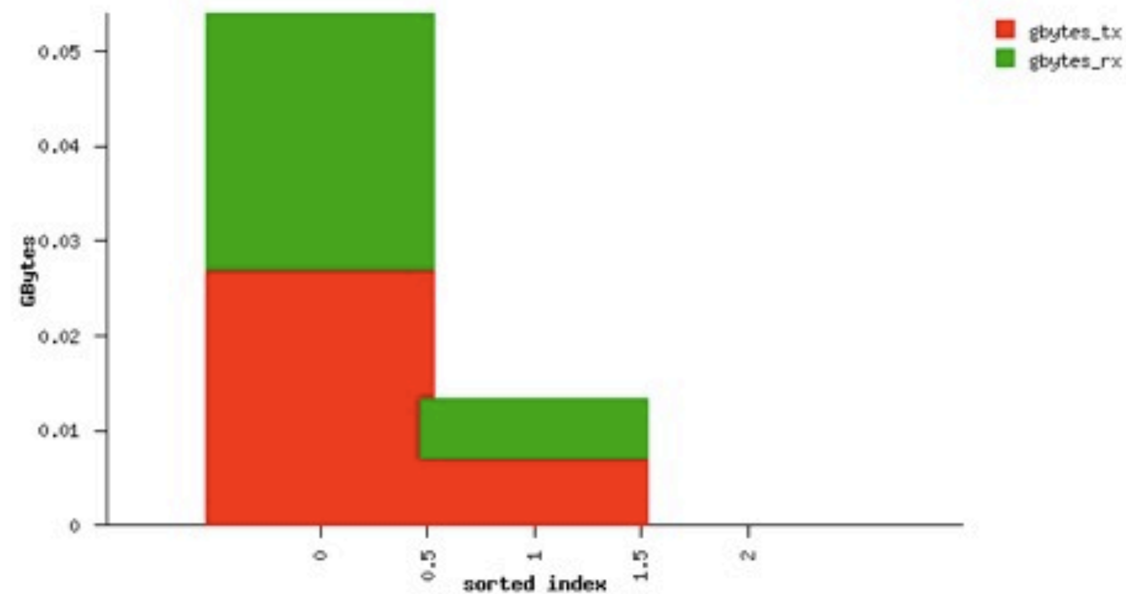
SciNet

Communication Topology : point to point data flow



data sent , data recv , time spent , map_data file map_adjacency file

Switch Traffic (volume by node)



Memory usage by node

Communications patterns, total switch traffic (I/O + MPI)

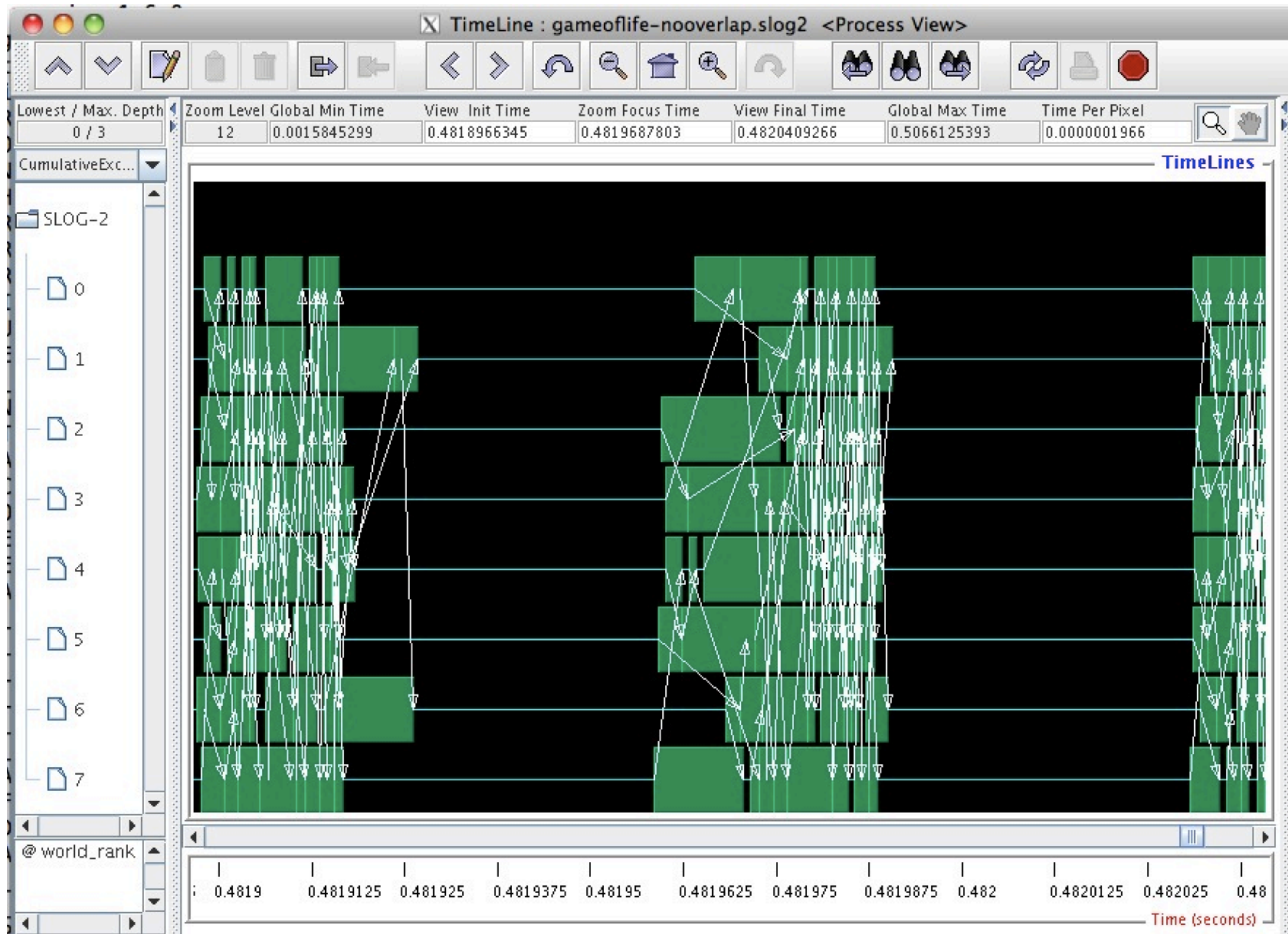
SciNet

MPE/Jumpshot

- More detailed view of MPI calls
- Rather than just counting, actually logs every MPI call, can then be visualized.
- Higher overhead - more detailed data.

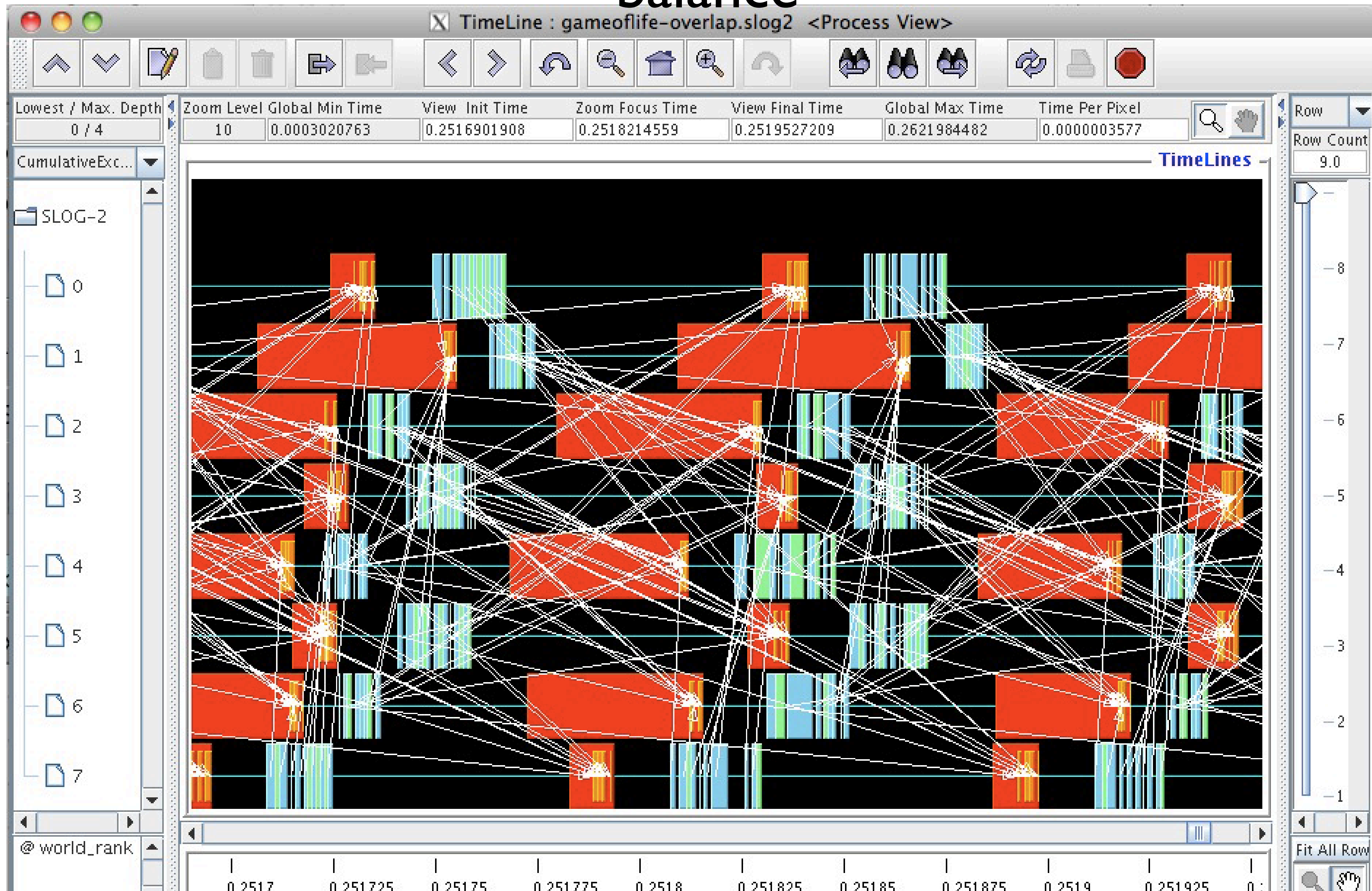
```
$ module load mpe
$ mpecc -mpilog -std=c99 gol.c -o gol
$ mpirun -np 8 ./gol
$ clog2TOslog2 gol.clog2
$ jumpshot gol.slog2
```

SciNet



SciNet

Overlapping communication & Computation: Much less synchronized (good); but shows poor load balance



Scalasca - Analysis

- Low-level automated instrumentation of code.
- High-level analysis of that data.
- Compile, run as normal, but prefix with:
  - compile: scalasca -instrument
  - run: scalasca -analyze
- Then scalasca -examine the resulting directory.

SciNet

Game of life: can take a look at data sent, received

Cube 3.3 QT: epik_gameoflife-scalasca_8_sum/summary.cube

File Display Topology Help

Absolute	Absolute	Absolute
Metric tree	Call tree / Flat view	System tree / Topology 0 / Topology 1

Metric tree

- 13.07 Time
- 2.71e7 Visits
- 0 Synchronizations
- 6.40e4 Communications
- 0 Bytes transferred
  - 0 Point-to-point
    - 4.32e5 Sent
    - 4.32e5 Received
  - 0 Collective
- 140 MPI file operations
- 0.01 Computational imbalance

Call tree

- 0 main
  - 0 init_rundata
  - 0 MPI_Init
  - 0 MPI_Comm_size
  - 0 nearsquare
  - 0 get_options
  - 0 MPI_Cart_create
  - 0 MPI_Comm_rank
  - 0 MPI_Cart_coords
  - 0 MPI_Cart_shift
  - 0 MPI_Cart_rank
  - 0 preadarray2d_mpiio
  - 0 pwritearray2d_mpiio
  - 0 guardcellexchange_nonblocking
    - 4.32e5 MPI_Isend
    - 0 MPI_Irecv
  - 0 iterate
  - 0 guardcellexchange_nonblocking_end
    - 0 MPI_Waitall
  - 0 finalize_rundata

System tree

- - Linux Cluster
  - - gpc-f109n002
    - 5.40e4 Process 0
    - 5.40e4 Process 1
    - 5.40e4 Process 2
    - 5.40e4 Process 3
    - 5.40e4 Process 4
    - 5.40e4 Process 5
    - 5.40e4 Process 6
    - 5.40e4 Process 7

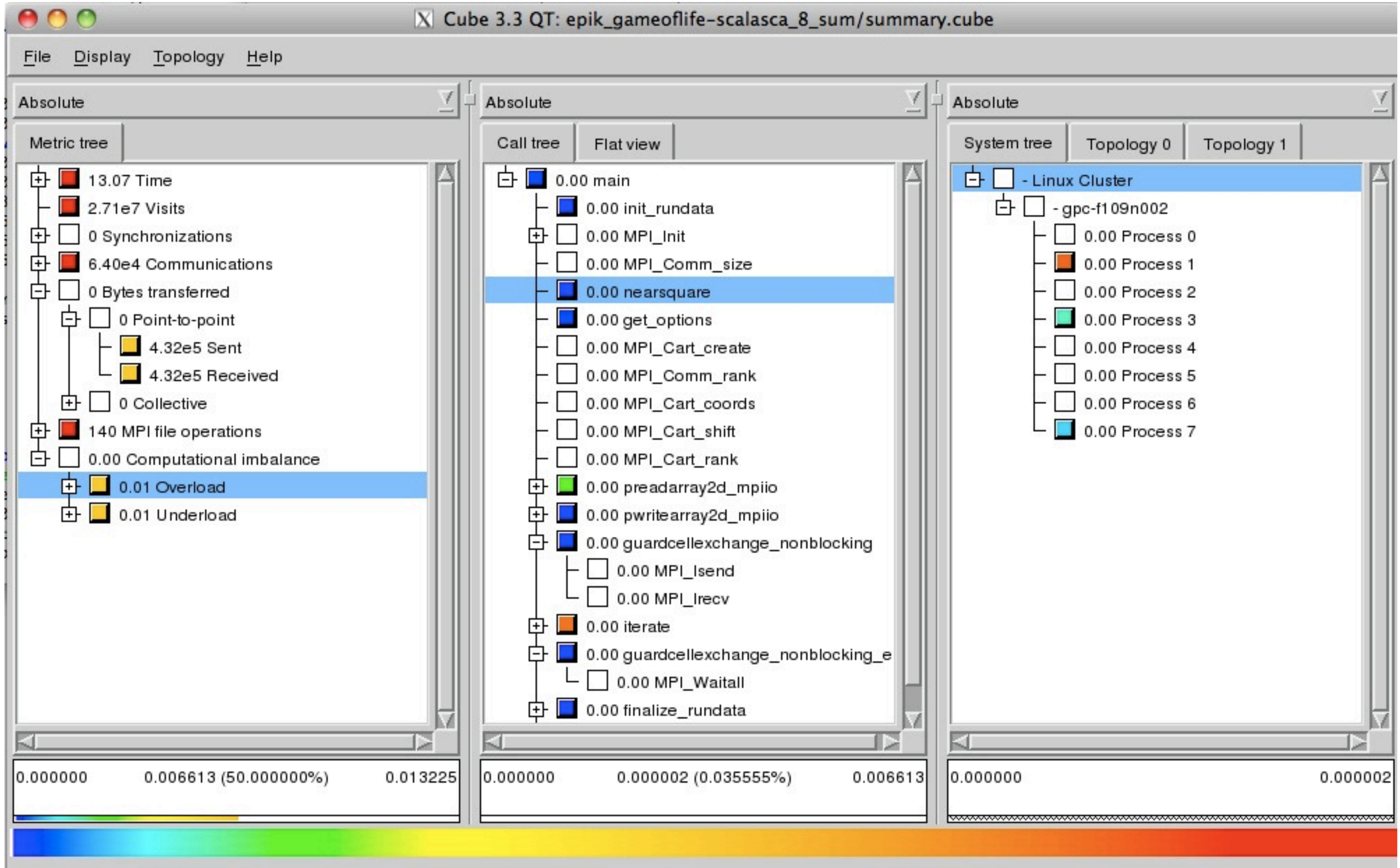
0 4.320000e5 (50.000000%) 8.640000e5

0 4.320000e5 (100.000000%) 4.320000e5

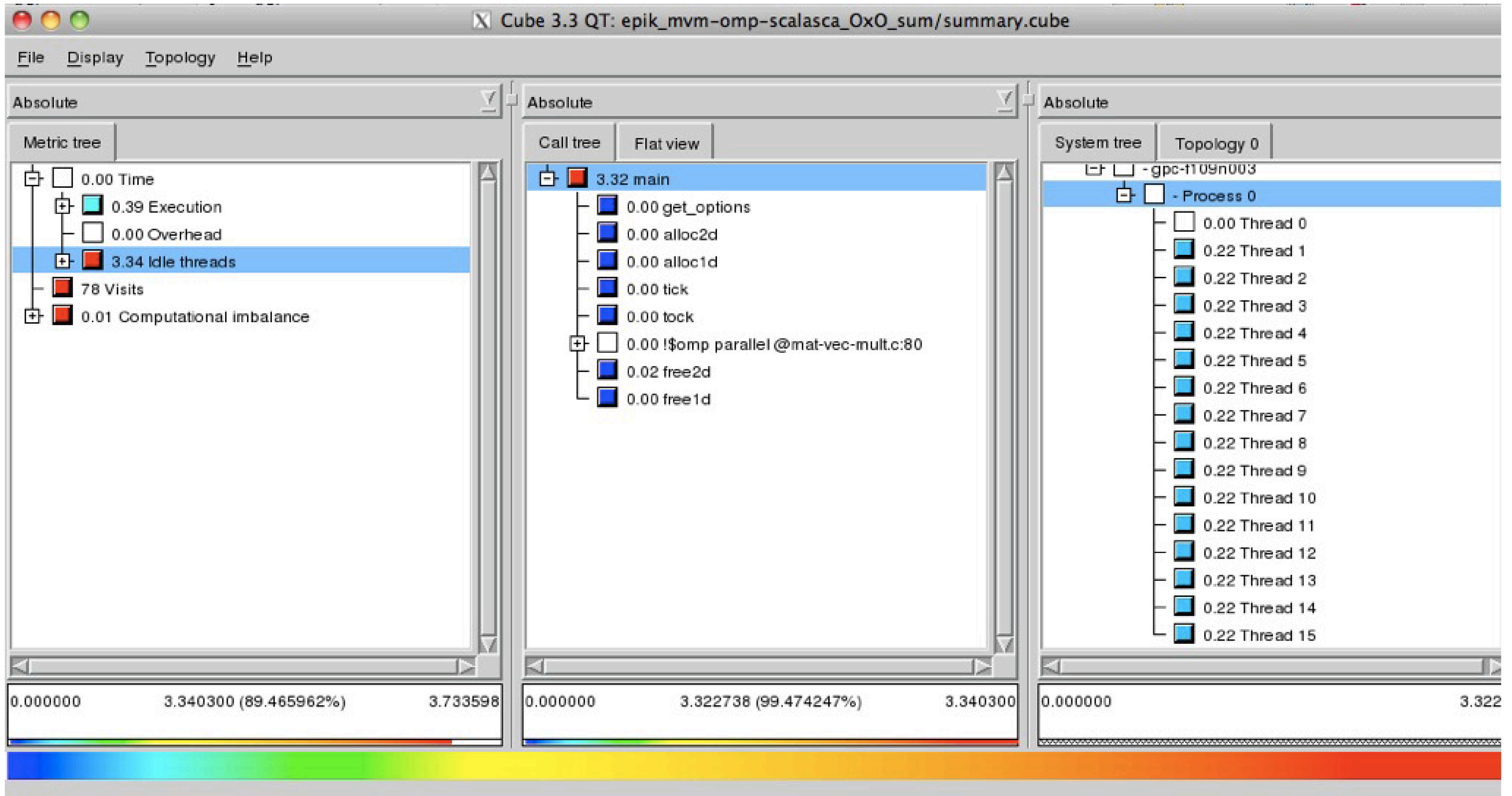
0 4.320000e5

SciNet

Can also see load imbalance -- by function, process



MVM - can show where threads are idle



(Thread 0 doing way too much work!)

SciNet

Coming Soon:

- Intel Trace Analyzer/Collector -- for MPI, like jumpshot + IPM.   A little easier to use
- Intel Vtune -- good thread performance analyzer

SciNet

Summary

- Use output .o files, or time, to get overall time - predict run time, notice if anything big changes
- Put your own timers in the code in important sections, find out where time is being spent
  - if something changes, know in what section

SciNet

Summary

- Gprof, or openss, are excellent for profiling serial code
- Even for parallel code, biggest wins often come from serial improvements
- Know important sections of code
- valgrind good for cache performance, memory checks.

SciNet

Summary

- Basically all MPI codes should be run with IPM
- Very low overhead, gives overview of MPI performance
- See communications structure, message statistics

SciNet

Summary

- OpenMP/pthreads code - Open|SpeedShop good for load balance issues
- MPI or OpenMP - Scalasca gives very good overview, shows common performance problems.

SciNet

Tuning Your MPI Application Without Writing Code

SNUG TechTalk, 8 Feb 2012

SciNet
compute • calcul
CANADA

Outline

- MPI Libraries
    - Eager vs Rendezvous, Collective Algorithms
- mpitune
- otpo
- Locality and Pinning

SciNet
compute • calcul
CANADA

Inside an MPI Library

You send a message, a miracle occurs, and the message is received on the other side.

- Jeff Squyres, Cisco/OpenMPI, OpenMPI Mailing list, Jan 2012

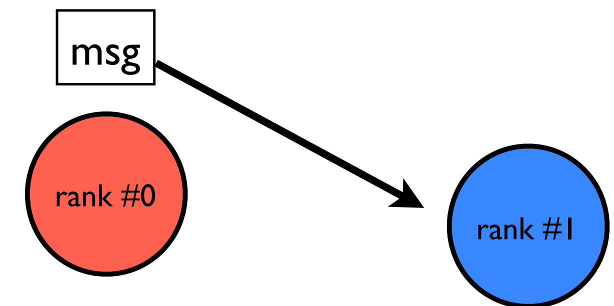
SciNet
compute • calcul
CANADA

Inside an MPI Library

- The MPI standard intentionally says nothing about *how* messages are sent between MPI tasks
- The implementation must decide
- Typically many behaviours, determined by threshold parameters.
- Parameters chosen for overall good performance - but your application may benefit from changing these.

SciNet
compute • calcul
CANADA

Point to Point

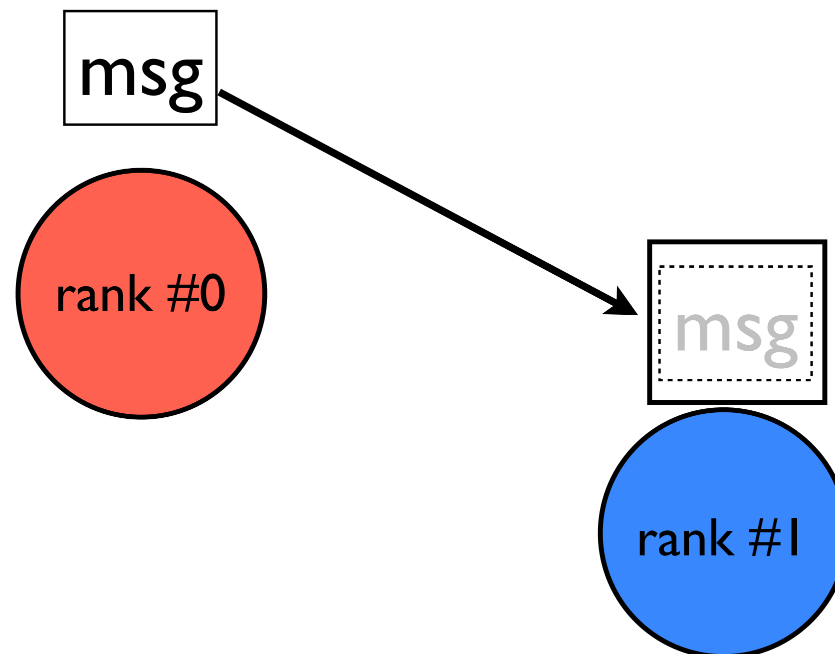


- Typically multiple protocols.
- None of this is in the standard; future implementations may use additional or different approaches entirely

SciNet
compute • calcul
CANADA

Point to Point: Eager

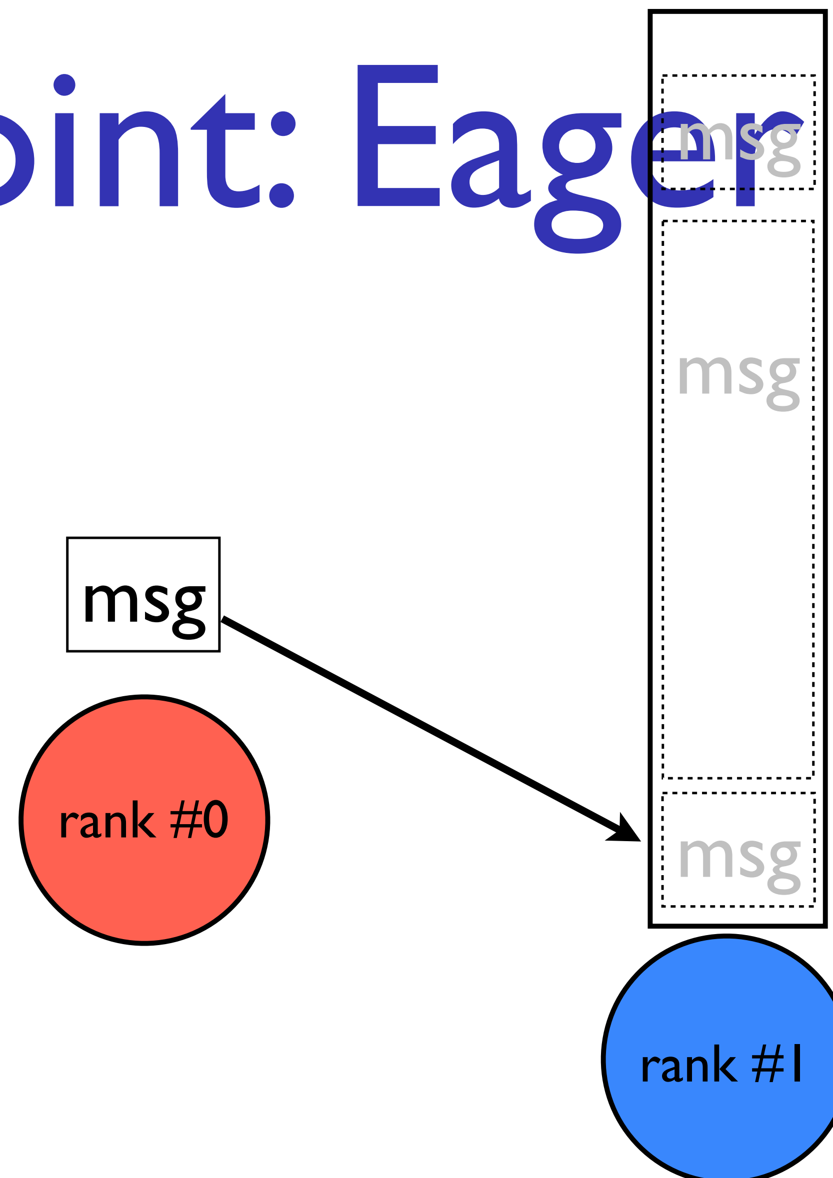
- Eager messages: sender plops message in MPI-defined system buffer on receive end.
- 1 transit of network



SciNet compute • calcul CANADA

Point to Point: Eager

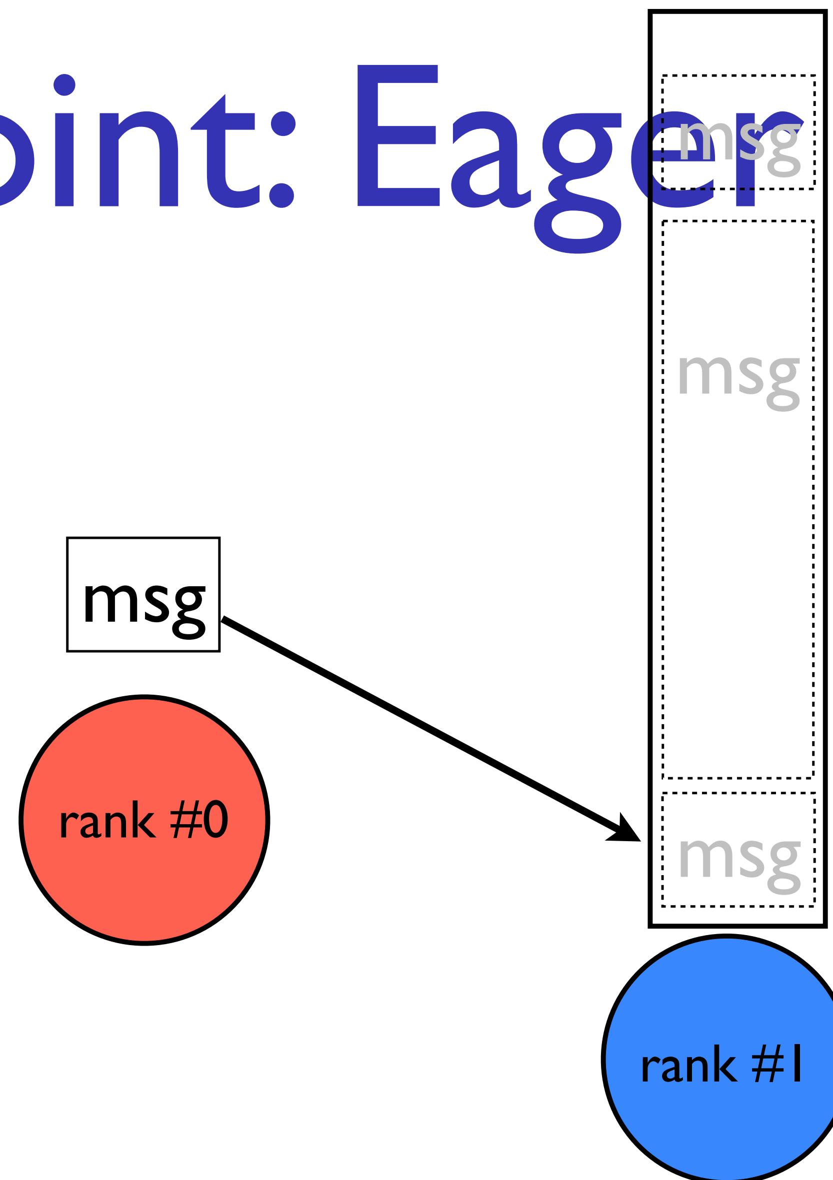
- But what if several messages pile up..
- And are quite large?
- (~100MB messages not uncommon in HPC)



SciNet
compute • calcul
CANADA

Point to Point: Eager

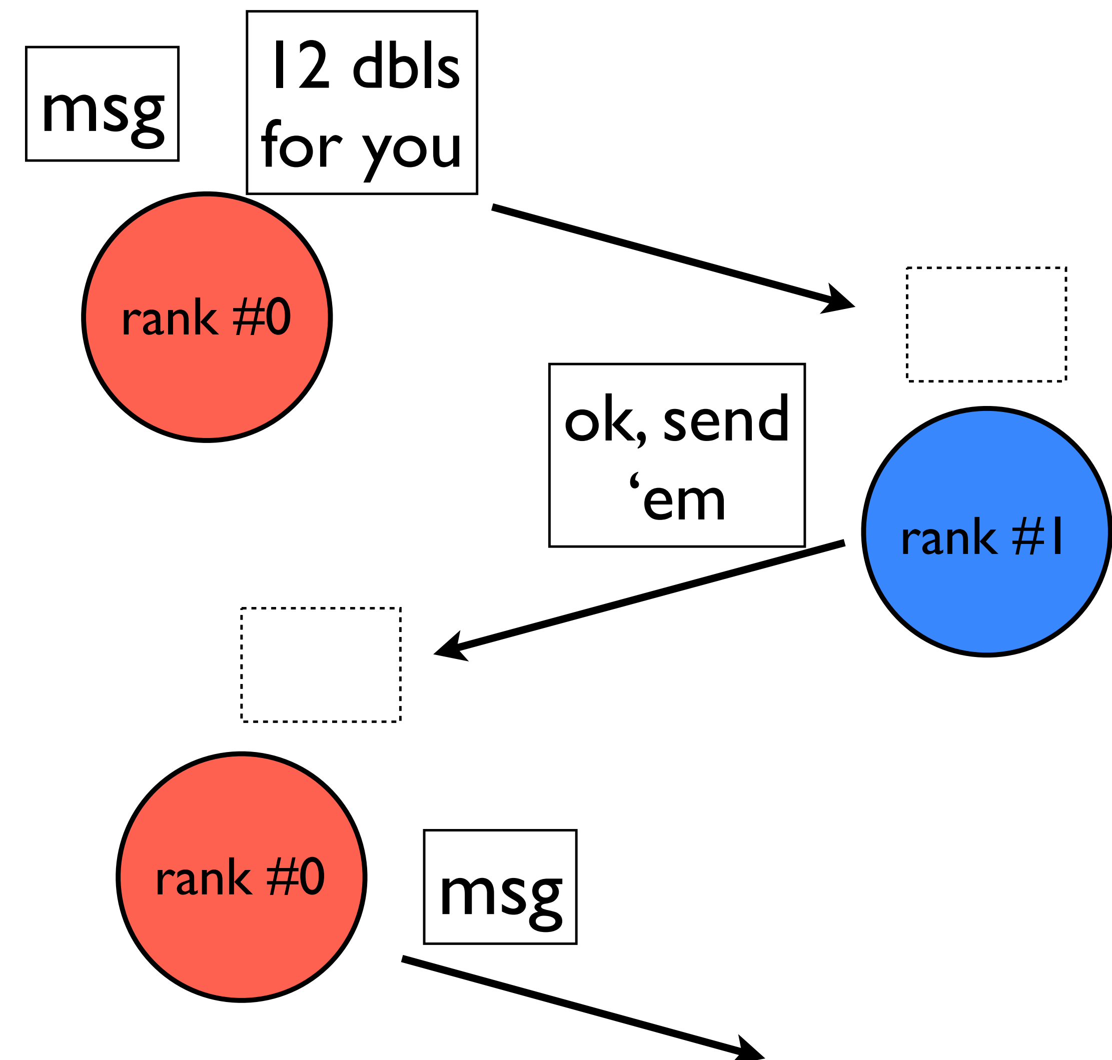
- Can't afford to dedicate large chunk of memory to receiveing data, "just in case"



SciNet compute • calcul CANADA

Point to Point: Rendezvous

- Rendezvous protocol: 3-way handshake
- First message is just "envelope" - describes contents
- Small, fits in memory.



Eager vs. Rendezvous

- Eager: faster, lower latency, requires big buffers on receive
- Rendezvous: much lower memory overhead for big messages, much larger latency esp. on slow networks
- But Rendezvous doesn't save any memory for messages approximately the size of the envelope..

SciNet
compute • calcul
CANADA

Eager vs. Rendezvous

- Send via Eager protocol for “small enough” messages
- Send via Rendezvous for “large” messages.
- “Eager Threshold” threshold tunable
- Typically one threshold per network type

SciNet
compute • calcul
CANADA

Protocols

- OpenMPI, MPICH2, etc implement much more just these two protocols
- Also transport-specific protocols/policies
- Policies for fragment sizes to use for large messages, pipelining, etc.
- But eager vs. handshake good distinction to know

SciNet compute • calcul CANADA

Setting eager thresholds

- OpenMPI:
  - --mca btl_sm_eager_limit [num] (default: 4k)
  - --mca btl_openib_eager_limit [num] (default: 12k)
  - --mca btl_tcp_eager_limit [num] (default: 64k)
  - Or: (eg)
    export OMPI_MCA_btl_sm_eager_limit=4096

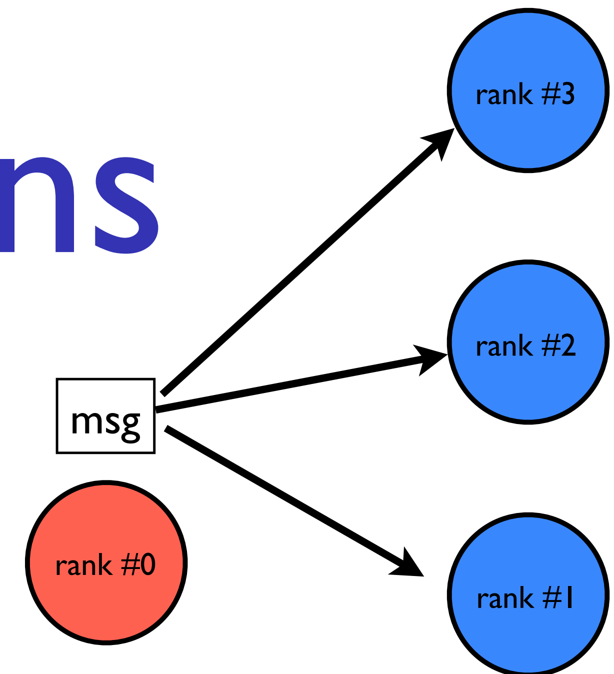
SciNet compute • calcul CANADA

Setting eager thresholds

- IntelMPI:
  - -genv I_MPI_EAGER_THRESHOLD [num] (default: 256k)
  - -genv I_MPI_INTRANODE_EAGER_THRESHOLD [num] (default: 256k)
  - -genv I_MPI_RDMA_EAGER_THRESHOLD [num] (default: 16k)
  - Or: (eg)
    export I_MPI_EAGER_THRESHOLD=4096

SciNet
compute • calcul
CANADA

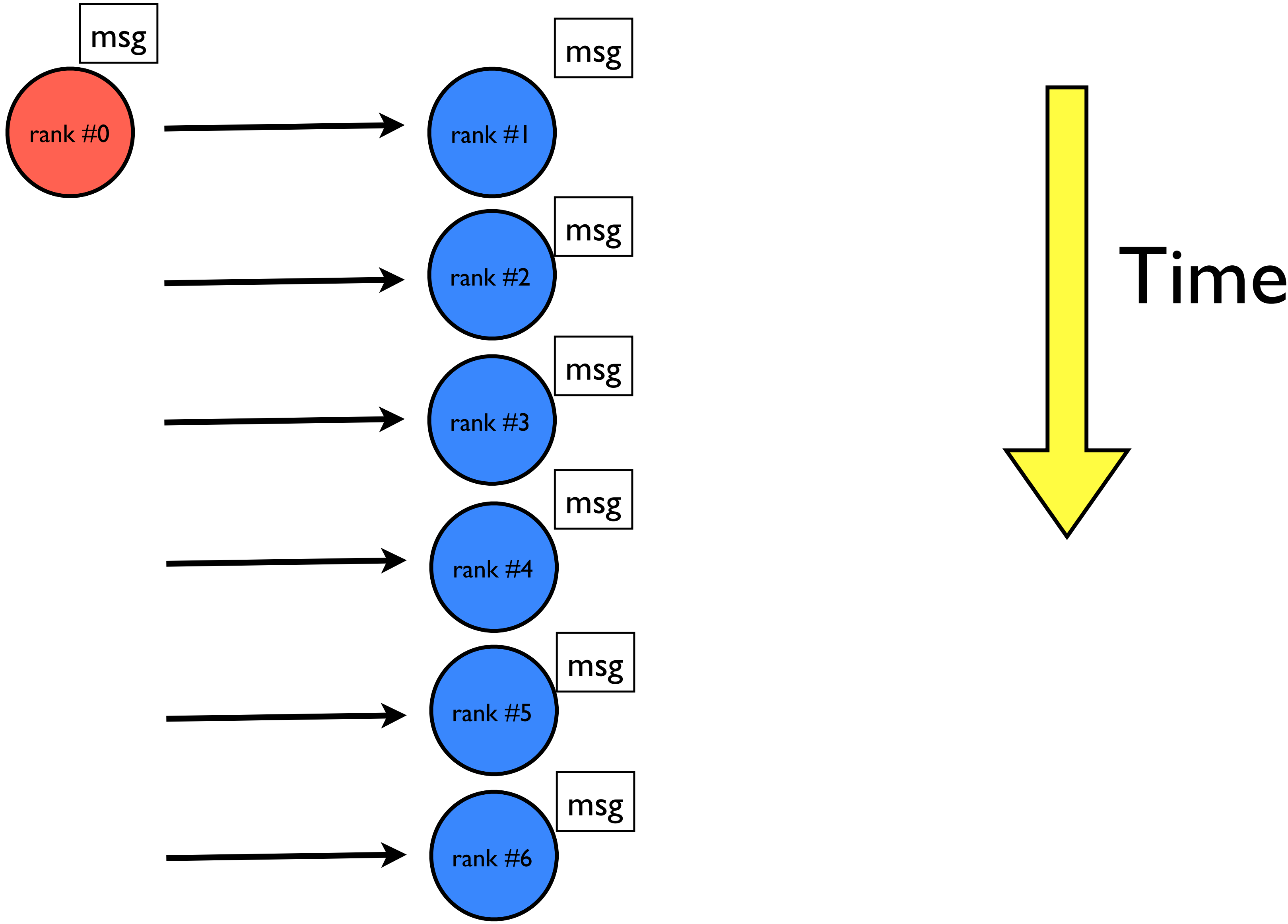
Collective Communications



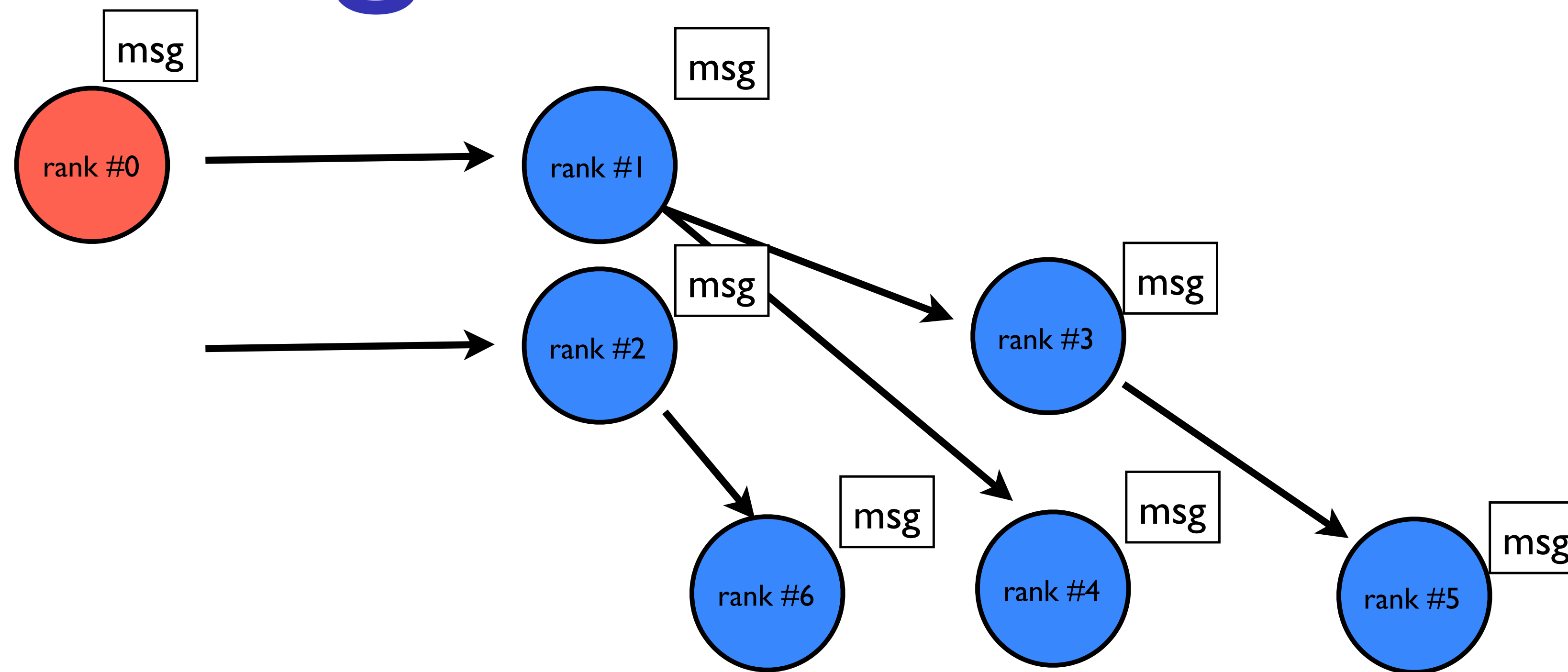
- Broadcast, Allreduce,...
- All the different ways above to send each individual message;
- *plus* decisions about which messages to send!

compute • calcul CANADA

Linear:

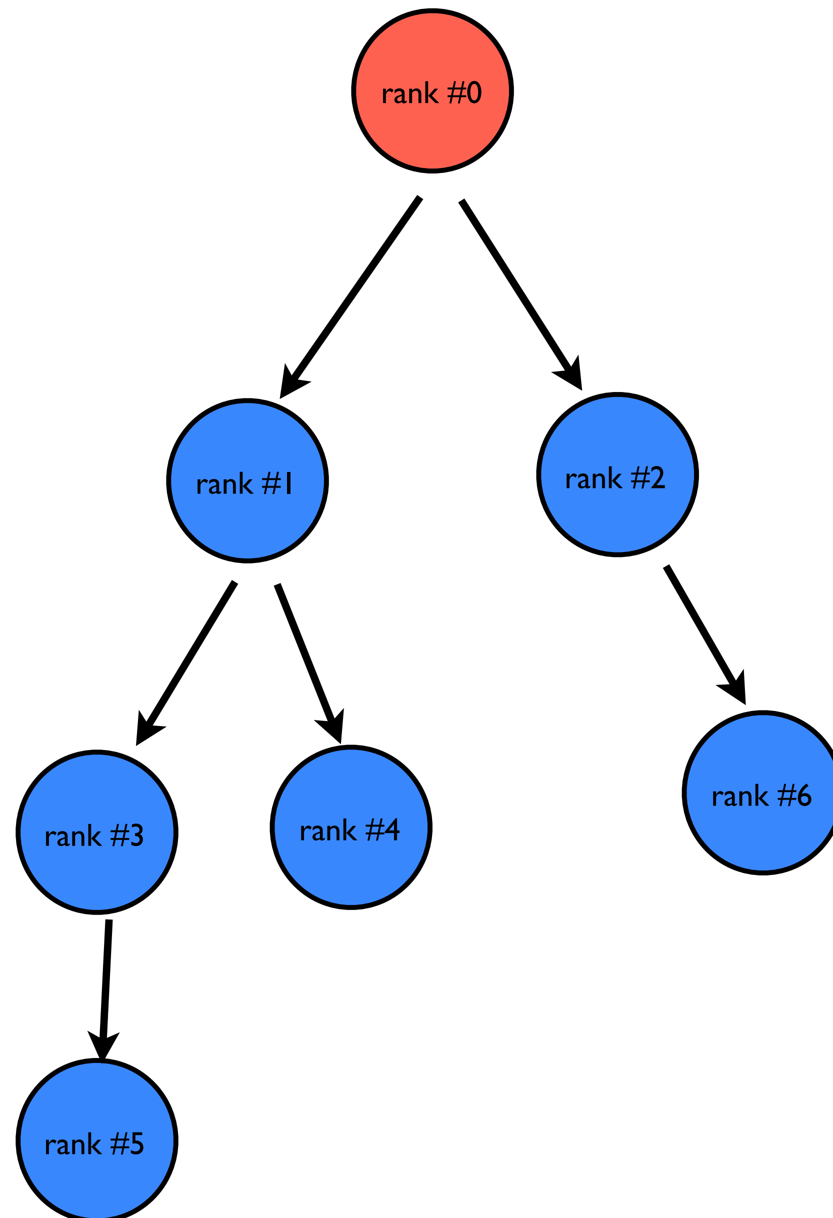


Logarithmic Tree



SciNet
compute • calcul
CANADA

Logarithmic Tree



SciNet compute • calcul CANADA

Linear vs. Logarithmic

- Hierarchical tree obviously scales much better
  - lg(P) steps vs. P
- But for small P, linear actually faster - lower overhead.

SciNet
compute • calcul
CANADA

Other considerations

- Modern clusters are hierarchial:
  - many cores in a node
  - many nodes on a switch
  - many switches in a cluster
- Modern MPI implementations have many collective algorithms, chosen depending on P, size of message, fabric...

SciNet
compute • calcul
CANADA

Adjusting algorithms

- IntelMPI
  - I_MPI_ADJUST_ALLREDUCE [num] (eg)
    - choose algorithm #[num]
- OpenMPI
  - --mca coll (many)
  - ompi_info --param coll all

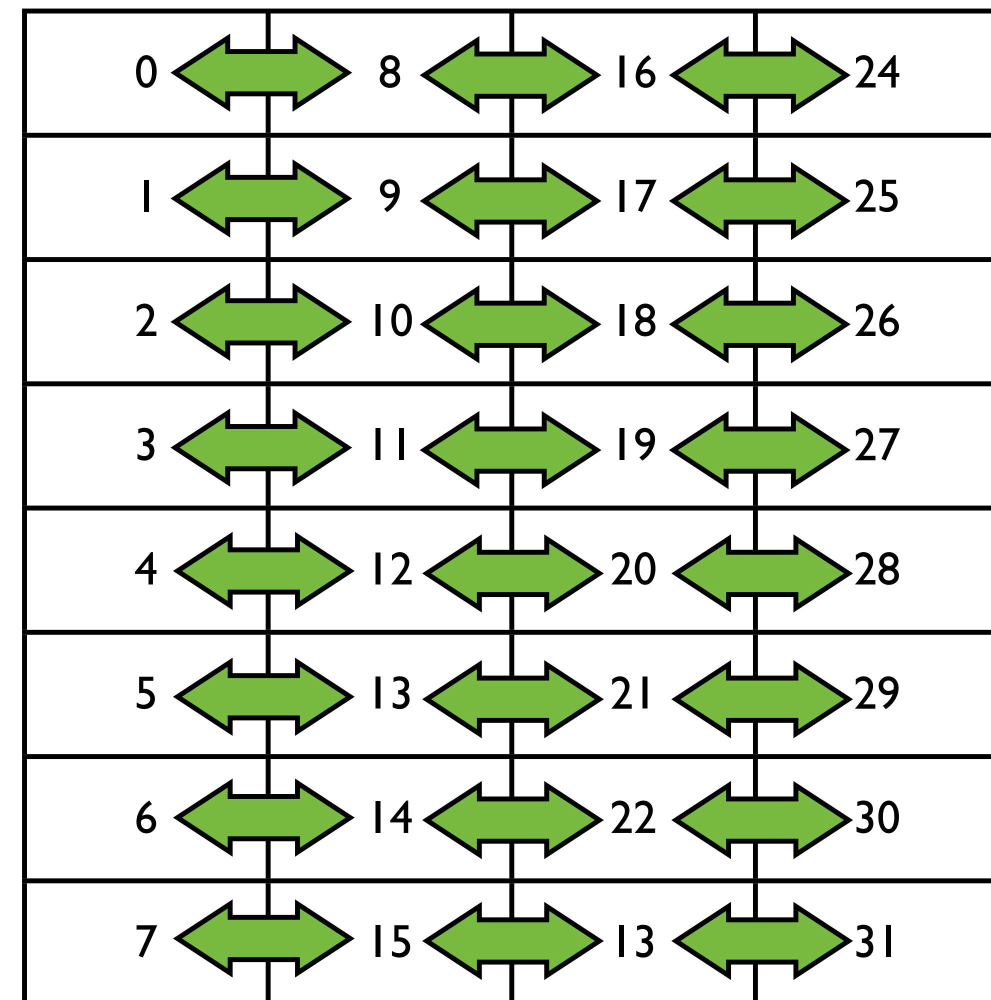
Utilities to test parameters for you

- mpitune (IntelMPI)
- otpo (OpenMPI: not nearly as full featured, mainly for sysadmins)

SciNet
compute • calcul
CANADA

Analysis.c

- Example program
- 256^2 x 32 array
- Pipeline data across rows, auto-correlation
- Allreduce answers



SciNet
compute • calcul
CANADA

```
MPI_Dims_create(size, 2, dims);   /* eg, an 8x4 grid on 4 nodes */

int left  = (rank - dims[0] + size) % size;
int right = (rank + dims[0]) % size;

int rows = (problemsize + (row/dims[0]))/dims[0];
int cols = (problemsize + (col/dims[1]))/dims[1];

int ndata = problemdepth*rows*cols;
double *data = malloc(ndata*sizeof(double));
double *extdata = malloc(ndata*sizeof(double));
double *result  = malloc(ndata*sizeof(double));
/* ... */

for (int iter=0; iter<5; iter++) {
    /* ... */

    /* calculate on local data */

    /* get external data and calculate on it */
    for (int i=1; i<dims[1]; i++) {
        MPI_Sendrecv(data, ndata, MPI_DOUBLE, (rank + i*dims[0])%size, i,
                     extdata, ndata, MPI_DOUBLE, MPI_ANY_SOURCE, i,
                     MPI_COMM_WORLD, &status);

        /* do something with data */
    }

   /* get some local max */
    MPI_Allreduce(&locmaxres, &maxres, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
}

MPI_Finalize();
```

SciNet compute • calcul CANADA

Analysis.c

- Run for 5 iterations with IPM
- module avail ipm
- mpicc .... -L${SCINET_IPM_LIB} -lipm
- Can’t improve performance if you don’t measure it...
- Start with Intel MPI library defaults

SciNet
compute • calcul
CANADA

Analysis.c

```
$ mpirun -genv I_MPI_FABRICS shm:tcp
    -np 32 ./analysis

$ ipm_parse -html ljdursi.*
```

SciNet compute • calcul CANADA

```
##IPMv0.983####################################################################
#
# command : ./analysis  (completed)
# host    : gpc-f104n043/x86_64_Linux       mpi_tasks : 32 on 4 nodes
# start   : 02/06/12/08:17:44               wallclock : 3.769331 sec
# stop    : 02/06/12/08:17:47               %comm     : 99.43
# gbytes  : 3.56665e+00 total               gflop/sec : 1.13818e-02 total
#
##############################################################################
# region  : *       [ntasks] =     32
#
#                          [total]         <avg>           min           max
# entries                       32             1             1             1
# wallclock                120.527       3.76648       3.75933       3.76933
# user                     94.4216       2.95068       1.71474       3.36049
# system                   27.3028      0.853214      0.443932       2.08168
# mpi                      119.934       3.74795       3.74724       3.74866
# %comm                                  99.4327       99.4332       99.6855
# gflop/sec              0.0113818   0.000355681   0.000352649   0.000357786
# gbytes                   3.56665      0.111458      0.111458      0.111458
#
# PAPI_FP_OPS          4.29017e+07   1.34068e+06   1.32925e+06   1.34861e+06
# PAPI_FP_INS          4.28852e+07   1.34016e+06   1.32874e+06   1.34812e+06
# PAPI_DP_OPS          8.57646e+07   2.68014e+06   2.65732e+06   2.69606e+06
# PAPI_VEC_DP          4.28795e+07   1.33998e+06   1.32857e+06   1.34794e+06
#
#                           [time]       [calls]        <%mpi>       <%wall>
# MPI_Sendrecv             70.4843           480         58.77         58.48
# MPI_Allreduce            49.4501           160         41.23         41.03
# MPI_Comm_rank        1.01876e-05            32          0.00          0.00
# MPI_Comm_size        7.42101e-06            32          0.00          0.00
###############################################################################
```

99% of time spent in communications

Computation		
Event	Count	Pop
	0	*
PAPI_DP_OPS	85764638	*
PAPI_FP_INS	42885154	*
PAPI_FP_OPS	42901729	*
PAPI_VEC_DP	42879484	*

Communication

% of MPI Time

MPI_Sendrecv
MPI_Allreduce
MPI_Comm_rank
MPI_Comm_size

HPM Counter Statistics

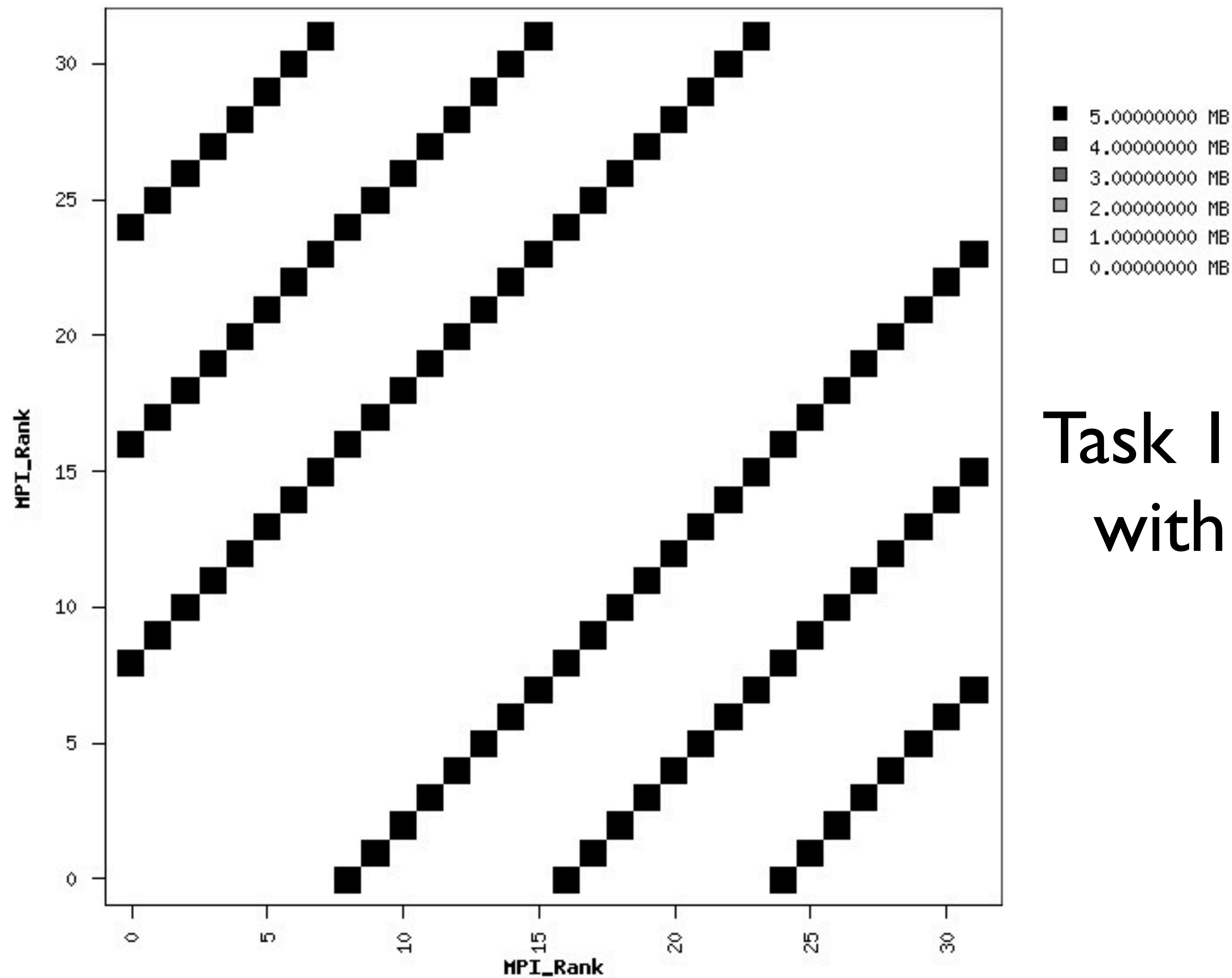
Event	Ntasks	Avg	Min(rank)	M
	*	0.00	0 (0)	
PAPI_DP_OPS	*	2680144.94	2657316 (17)	2
PAPI_FP_INS	*	1340161.06	1328743 (17)	1
PAPI_FP_OPS	*	1340679.03	1329251 (17)	1
PAPI_VEC_DP	*	1339983.88	1328573 (17)	1

Communication Event Statistics (100.00% detail, 3.2139e-04 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%W
MPI_Sendrecv	524288	480	70.484	3.681e-02	9.159e-01	58.77	
MPI_Allreduce	8	160	49.450	2.715e-02	8.710e-01	41.23	

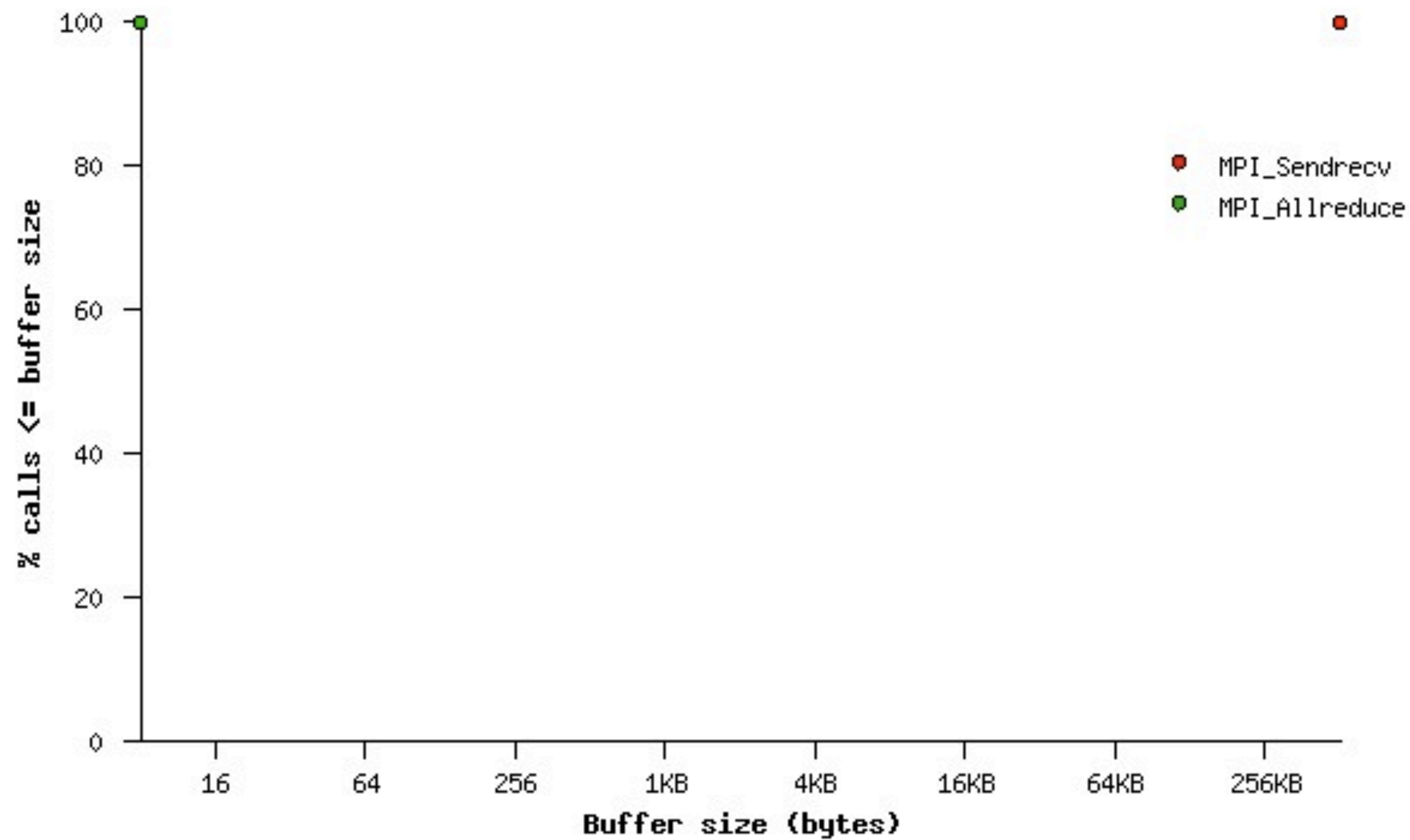
480 sendrecvs, 160 allreduces (across all procs)
sendrecv: 70s, 59% of MPI time.

SciNet compute • calcul CANADA



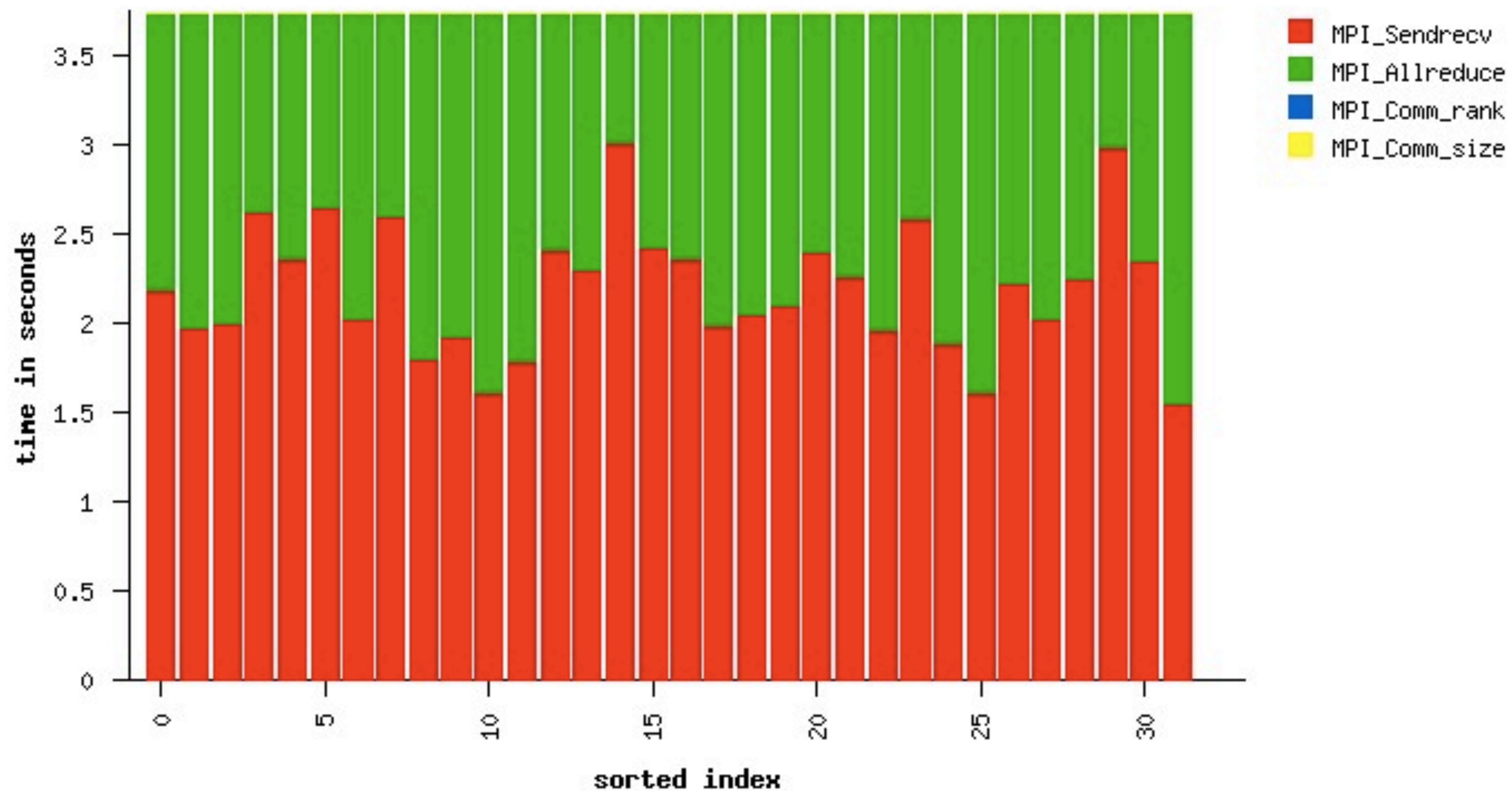
Task 10 communicates with tasks 18, 26, 2; etc.

SciNet compute • calcul CANADA



Only two message sizes:
Allreduce (single float)
Sendrecv (~32x32x64 floats)

SciNet compute • calcul CANADA



About 3.75 s/task in MPI; a lot of allreduce time is likely due to load imbalance (communications)

SciNet compute • calcul CANADA

mpitune

- IntelMPI utility
- Repeatedly (~couple dozen times, maybe more) runs your program while changing a handful of MPI parameters
- Have a *short* but realistically *sized* version of your probem for this!
- Can change the default bundle of parameters.

SciNet
compute • calcul
CANADA

mpitune

```
$ mpitune -of analysis.conf
   --application \"mpiexec -genv
   I_MPI_FABRICS shm:tcp -n 32
   analysis-noipm\"

$ mpirun -genv I_MPI_FABRICS shm:tcp
      -tune analysis.conf
      -np 32 ./analysis

$ ipm_parse -html ljdursi.*
```

SciNet compute • calcul CANADA

analysis.conf

```
-genv I_MPI_RDMA_SCALABLE_PROGRESS  0
-genv I_MPI_WAIT_MODE  1
-genv I_MPI_INTRANODE_EAGER_THRESHOLD  2097152
-genv I_MPI_RDMA_EAGER_THRESHOLD  3145728
-genv I_MPI_ADJUST_ALLREDUCE  '6:8-8'
```

Eager threshold increased
(2MB! But there's always a waiting receive)
RDMA not used here
Allreduce algorithm changed

SciNet
compute • calcul
CANADA

```
##IPMv0.983####################################################################
#
# command : ./analysis  (completed)
# host    : gpc-f104n043/x86_64_Linux      mpi_tasks : 32 on 4 nodes
# start   : 02/06/12/08:25:29              wallclock : 3.064447 sec
# stop    : 02/06/12/08:25:32              %comm     : 99.45
# gbytes  : 3.56665e+00 total              gflop/sec : 1.39936e-02 total
#
##############################################################################
# region  : *       [ntasks] =     32
#
#                          [total]        <avg>          min          max
# entries                       32            1            1            1
# wallclock                98.0203      3.06313      3.06145      3.06445
# user                     75.0186      2.34433      1.61775      2.68559
# system                   24.1903     0.755947     0.413937      1.48877
# mpi                      97.5254      3.04767      3.04669      3.04825
# %comm                                 99.4525      99.4511      99.5637
# gflop/sec              0.0139936  0.000437301  0.000433963  0.000440101
# gbytes                   3.56665     0.111458     0.111458     0.111458
#
# PAPI_FP_OPS          4.28828e+07  1.34009e+06  1.32986e+06  1.34867e+06
# PAPI_FP_INS          4.28661e+07  1.33957e+06  1.32935e+06  1.34815e+06
# PAPI_DP_OPS          8.57265e+07  2.67895e+06  2.65852e+06  2.69614e+06
# PAPI_VEC_DP          4.28604e+07  1.33939e+06  1.32917e+06  1.34798e+06
#
#                           [time]      [calls]       <%mpi>      <%wall>
# MPI_Sendrecv             59.0132          480        60.51        60.21
# MPI_Allreduce            38.5122          160        39.49        39.29
# MPI_Comm_rank        8.79297e-06           32         0.00         0.00
# MPI_Comm_size         7.6741e-06           32         0.00         0.00
###############################################################################
```

SendRecv: 59 (was 71); Allreduce 39 (was 50)

SciNet compute • calcul CANADA

Computation		
Event	**Count**	**Pop**
	0	*
PAPI_DP_OPS	85726533	*
PAPI_FP_INS	42866118	*
PAPI_FP_OPS	42882785	*
PAPI_VEC_DP	42860415	*

Communication

% of MPI Time

MPI_Sendrecv
MPI_Allreduce
MPI_Comm_rank
MPI_Comm_size

HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Ma
	*	0.00	0 (0)	
PAPI_DP_OPS	*	2678954.16	2658521 (21)	2696
PAPI_FP_INS	*	1339566.19	1329347 (21)	1348
PAPI_FP_OPS	*	1340087.03	1329856 (21)	1348
PAPI_VEC_DP	*	1339387.97	1329174 (21)	1347

Communication Event Statistics (100.00% detail, -2.4647e-04 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wal
MPI_Sendrecv	524288	480	59.013	2.298e-02	6.650e-01	60.51	
MPI_Allreduce	8	160	38.512	2.292e-04	6.296e-01	39.49	

SciNet compute • calcul CANADA

mpitune

- 20% improvement in runtime! (Extreme case)
- Can work very well for a code dominated by one (or very small number of) communications patters
- Need to find shortest-time case that exercises all of the communications patterns on real-sized problems.
- Works only with IntelMPI

SciNet
compute • calcul
CANADA

otpo

- Part of OpenMPI suite of tools
- Mainly used for tuning OpenMPI as a whole for given cluster
- Runs well-established benchmarks (NAS, netpipe, Skapi)
- If your code looks like one of those, can be useful.

SciNet compute • calcul CANADA

Locality

- Can use 'hostname' to find out what hosts are being used
- And with intel mpi, "-l" labels the output by each rank

```
$ mpirun -l -np 32 hostname | sort -n
0: gpc-f109n001
1: gpc-f109n001
2: gpc-f109n001
3: gpc-f109n001
4: gpc-f109n001
5: gpc-f109n001
6: gpc-f109n001
7: gpc-f109n001

8: gpc-f109n002
9: gpc-f109n002
10: gpc-f109n002
11: gpc-f109n002
12: gpc-f109n002
```

SciNet compute • calcul CANADA

Locality

- OpenMPI: "--tag-output"
- [exe,rank]

```
$ mpirun --tag-output -np 32 hostname

[1,0]<stdout>:gpc-f109n001
[1,1]<stdout>:gpc-f109n001
[1,2]<stdout>:gpc-f109n001
[1,3]<stdout>:gpc-f109n001
[1,4]<stdout>:gpc-f109n001
[1,5]<stdout>:gpc-f109n001
[1,6]<stdout>:gpc-f109n001
[1,7]<stdout>:gpc-f109n001
[1,8]<stdout>:gpc-f109n002
[1,9]<stdout>:gpc-f109n002
[1,10]<stdout>:gpc-f109n002
[1,11]<stdout>:gpc-f109n002
[1,12]<stdout>:gpc-f109n002
[1,13]<stdout>:gpc-f109n002
[1,14]<stdout>:gpc-f109n002
....
```

SciNet
compute • calcul
CANADA

Locality

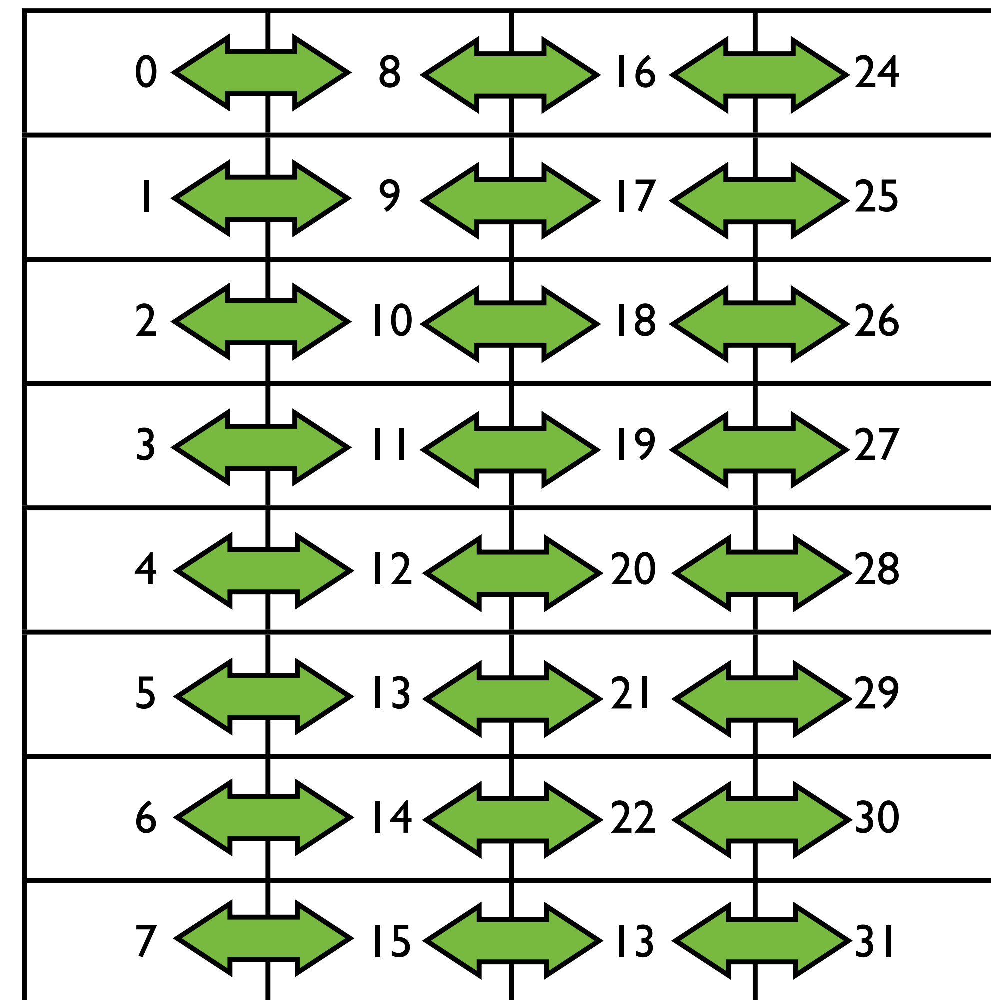
- OpenMPI: "--display-map"
- At start of job, lays out the ranks on each host

```
mpirun -display-map -np 32 hostname

========================   JOB MAP   ===========

Data for node: Name: gpc-f109n001   Num procs: 8
   Process OMPI jobid: [932,1] Process rank: 0
   Process OMPI jobid: [932,1] Process rank: 1
   Process OMPI jobid: [932,1] Process rank: 2
   Process OMPI jobid: [932,1] Process rank: 3
   Process OMPI jobid: [932,1] Process rank: 4
   Process OMPI jobid: [932,1] Process rank: 5
   Process OMPI jobid: [932,1] Process rank: 6
   Process OMPI jobid: [932,1] Process rank: 7

Data for node: Name: gpc-f109n002   Num procs: 8
   Process OMPI jobid: [932,1] Process rank: 8
   Process OMPI jobid: [932,1] Process rank: 9
   Process OMPI jobid: [932,1] Process rank: 10
   Process OMPI jobid: [932,1] Process rank: 11
   Process OMPI jobid: [932,1] Process rank: 12
   Process OMPI jobid: [932,1] Process rank: 13
   Process OMPI jobid: [932,1] Process rank: 14
   Process OMPI jobid: [932,1] Process rank: 15

....
==================================================
```

SciNet
compute • calcul
CANADA

Locality

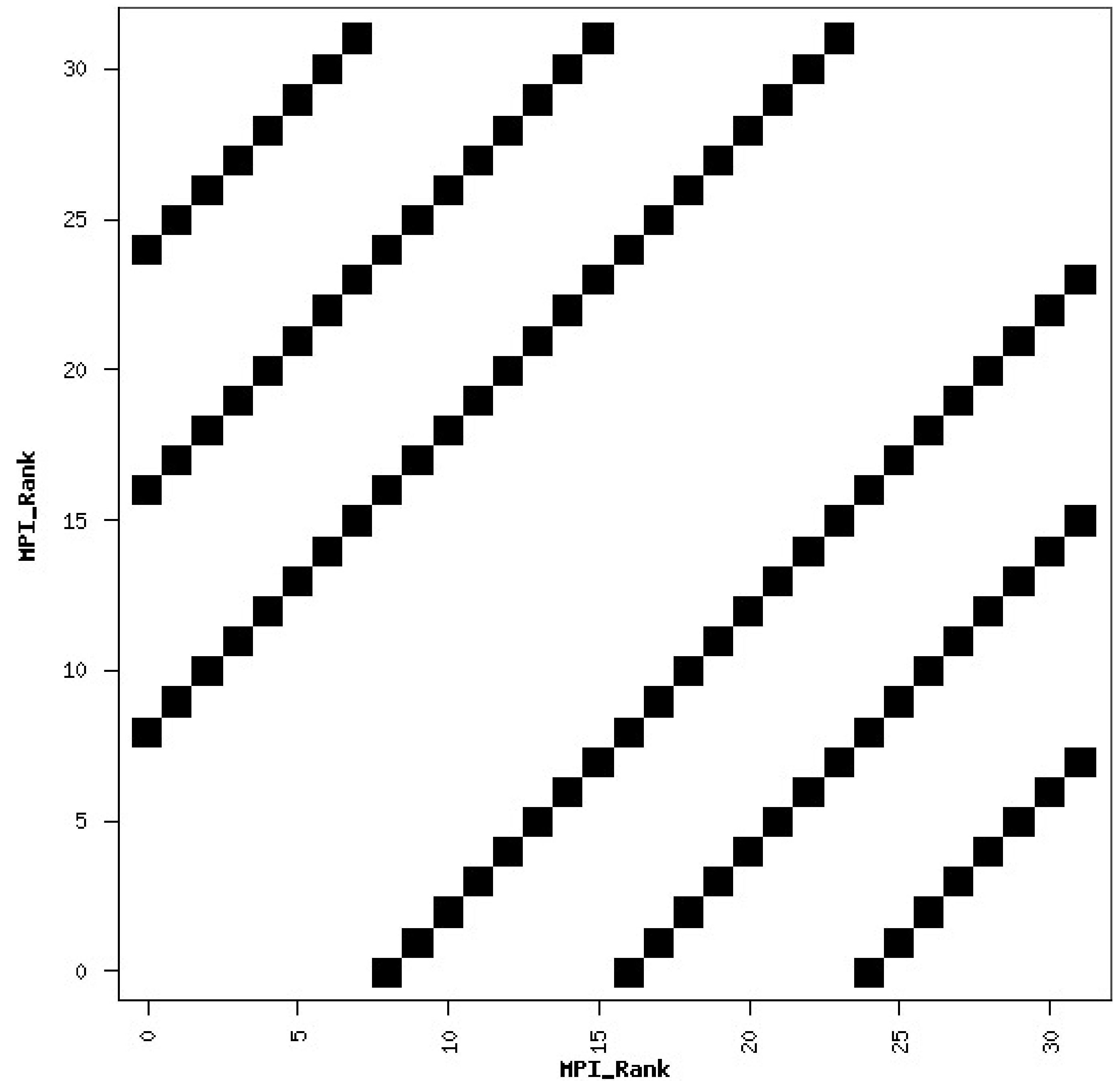
- Note the setup for our analysis routine
- Almost all the communications going off-node
- Off-node always slower than on.



SciNet
compute • calcul CANADA

Locality

- Note the setup for our analysis routine
- Almost all the communications going off-node
- Off-node always slower than on.



Round-Robin allocation

- Instead of filling up a node before next,
- Puts one rank on node, 2nd rank on 2nd node, etc.

```
$ mpirun -l -rr -np 32 hostname | sort -n
0: gpc-f109n001
1: gpc-f109n006
2: gpc-f109n005
3: gpc-f109n002
4: gpc-f109n001
5: gpc-f109n006
6: gpc-f109n005
7: gpc-f109n002
8: gpc-f109n001
9: gpc-f109n006
10: gpc-f109n005
11: gpc-f109n002
12: gpc-f109n001
```

SciNet compute • calcul CANADA

Round-Robin allocation

- IntelMPI: -rr
- OpenMPI --bynode
- Other OpenMPI options: --bysocket, --bycore..

SciNet
compute • calcul
CANADA

Run with -rr

Computation

Event	Count	Pop
	0	*
PAPI_DP_OPS	86631470	*
PAPI_FP_INS	43318596	*
PAPI_FP_OPS	43387516	*
PAPI_VEC_DP	43312874	*

Communication

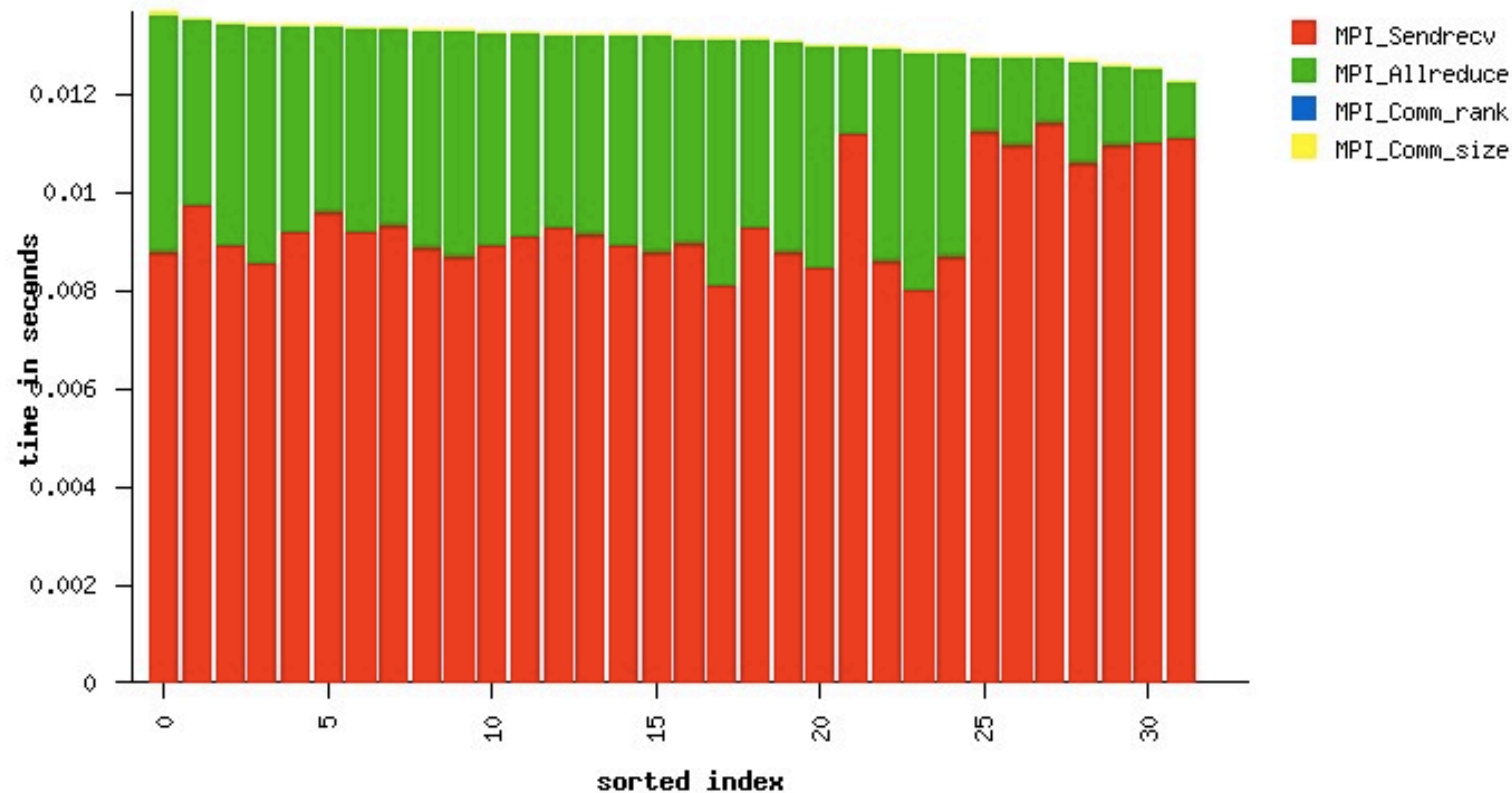
% of MPI Time

IPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Ma
	*	0.00	0 (0)	
PAPI_DP_OPS	*	2707233.44	2692219 (2)	27320
PAPI_FP_INS	*	1353706.12	1346200 (2)	13660
PAPI_FP_OPS	*	1355859.88	1348309 (2)	13682
PAPI_VEC_DP	*	1353527.31	1346019 (2)	13659

Communication Event Statistics (100.00% detail, 4.0830e-08 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Sendrecv	524288	480	0.303	3.782e-04	2.797e-03	72.37	
MPI_Allreduce	8	160	0.116	3.161e-04	2.945e-03	27.63	



Huge difference! By keeping most communications on-node, enormously reduce runtime.

SciNet
compute • calcul
CANADA

Locality: -rr

- An admittedly exteme case, but an important point
- Layout of nodes for locality is extremely important.
- Could also fix this in the code by reordering (MPI_CART_CREATE)
- Even this case can be tuned, for improvements in allreduce

SciNet
compute • calcul
CANADA

Hybrid MPI/OpenMP

- Locality is extremely important in the case of hybrid codes.
- Typically you want one MPI task per node (or per socket), and multiple threads per task.
- Want them to stay put; threads shouldn't move around within the node.

SciNet
compute • calcul
CANADA

OpenMPI

- hwloc library implements binding
- Make sure you specify how many cores per rank:
- Default will just

SciNet
compute • calcul
CANADA

Good:

```
gpc-f109n002-$ mpirun --display-map -cpus-per-rank 4 -np 8 hostname

 ========================   JOB MAP   ========================

 Data for node: Name: gpc-f109n002 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 0
   Process OMPI jobid: [61447,1] Process rank: 1

 Data for node: Name: gpc-f109n003 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 2
   Process OMPI jobid: [61447,1] Process rank: 3

 Data for node: Name: gpc-f109n004 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 4
   Process OMPI jobid: [61447,1] Process rank: 5

 Data for node: Name: gpc-f109n005 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 6
   Process OMPI jobid: [61447,1] Process rank: 7

 =============================================================
```

SciNet compute • calcul CANADA

Bad:

```
$ mpirun --display-map -np 8 hostname

 ========================   JOB MAP   ========================

 Data for node: Name: gpc-f109n002 Num procs: 8
   Process OMPI jobid: [61470,1] Process rank: 0
   Process OMPI jobid: [61470,1] Process rank: 1
   Process OMPI jobid: [61470,1] Process rank: 2
   Process OMPI jobid: [61470,1] Process rank: 3
   Process OMPI jobid: [61470,1] Process rank: 4
   Process OMPI jobid: [61470,1] Process rank: 5
   Process OMPI jobid: [61470,1] Process rank: 6
   Process OMPI jobid: [61470,1] Process rank: 7

 =============================================================
```

SciNet
compute • calcul
CANADA

IntelMPI

- Same deal; if you only want (say) 2 tasks per node, use -perhost 2
- -rr if you want them to be round-robined between nodes.

SciNet compute • calcul CANADA

Specifying process maps

- If you have a specific process layout in mind, either MPI library will allow you to do that.
- With hybrid codes, in IntelMPI, best to export OMP_NUM_THREADS to the appropriate number, and then use I_MPI_PIN_DOMAIN=omp to keep threasds in right place

SciNet
compute • calcul
CANADA

Conclusions

- Be aware of where your processes are communicating
- IPM is an invaluable tool for this!
- mpitune is worth using IntelMPI for

SciNet
compute • calcul
CANADA