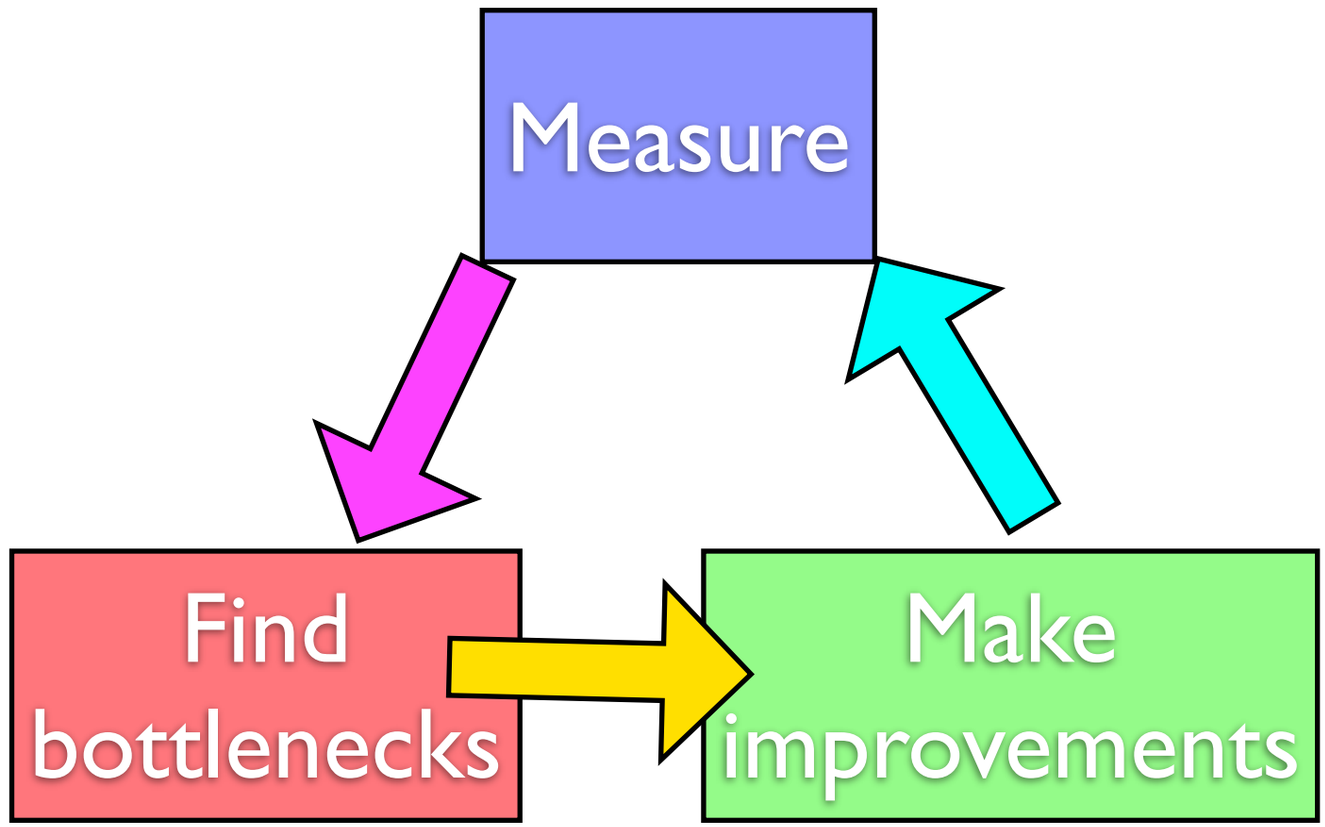
Profiling and Tuning

SciNet TechTalk, December SciNet Users Group Meeting

SciNet

How to improve Performance?

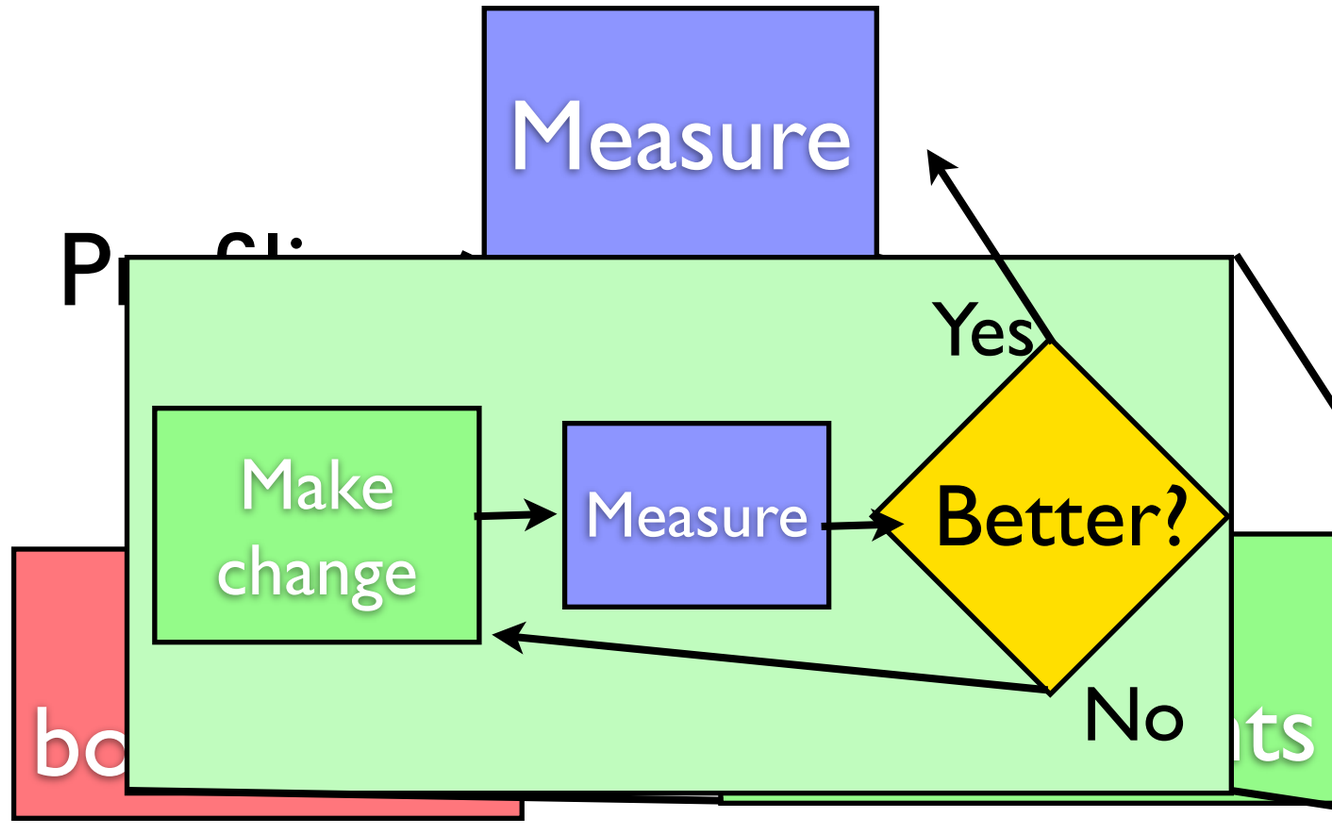
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

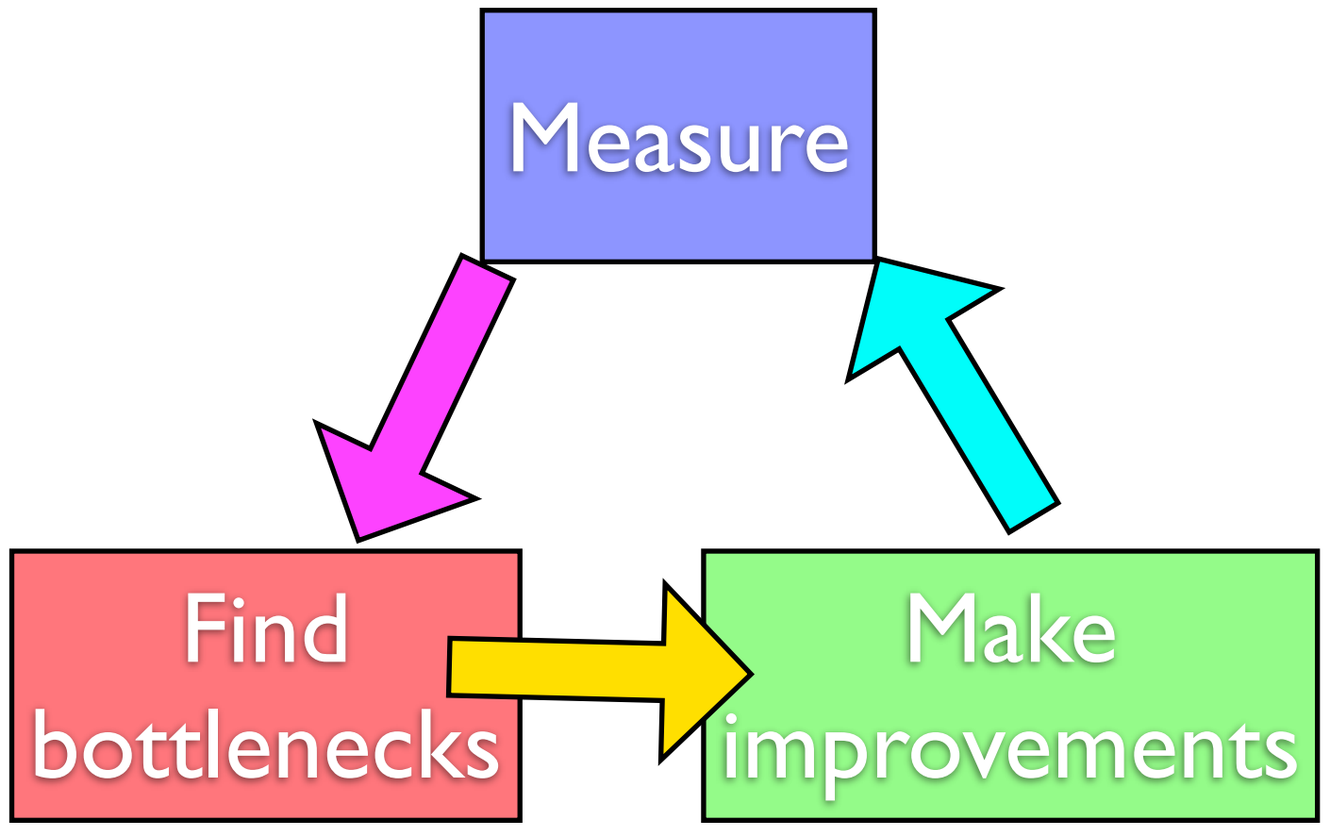
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

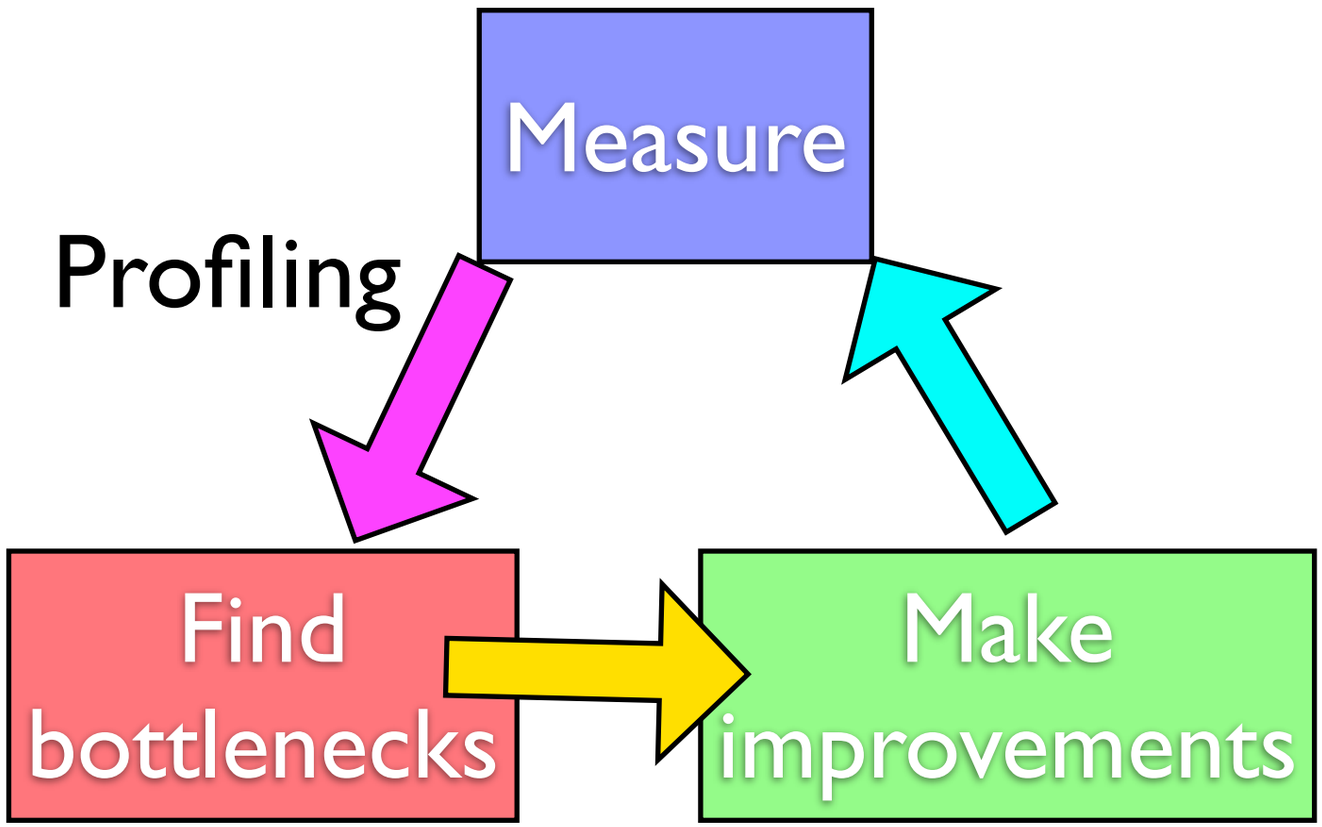
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

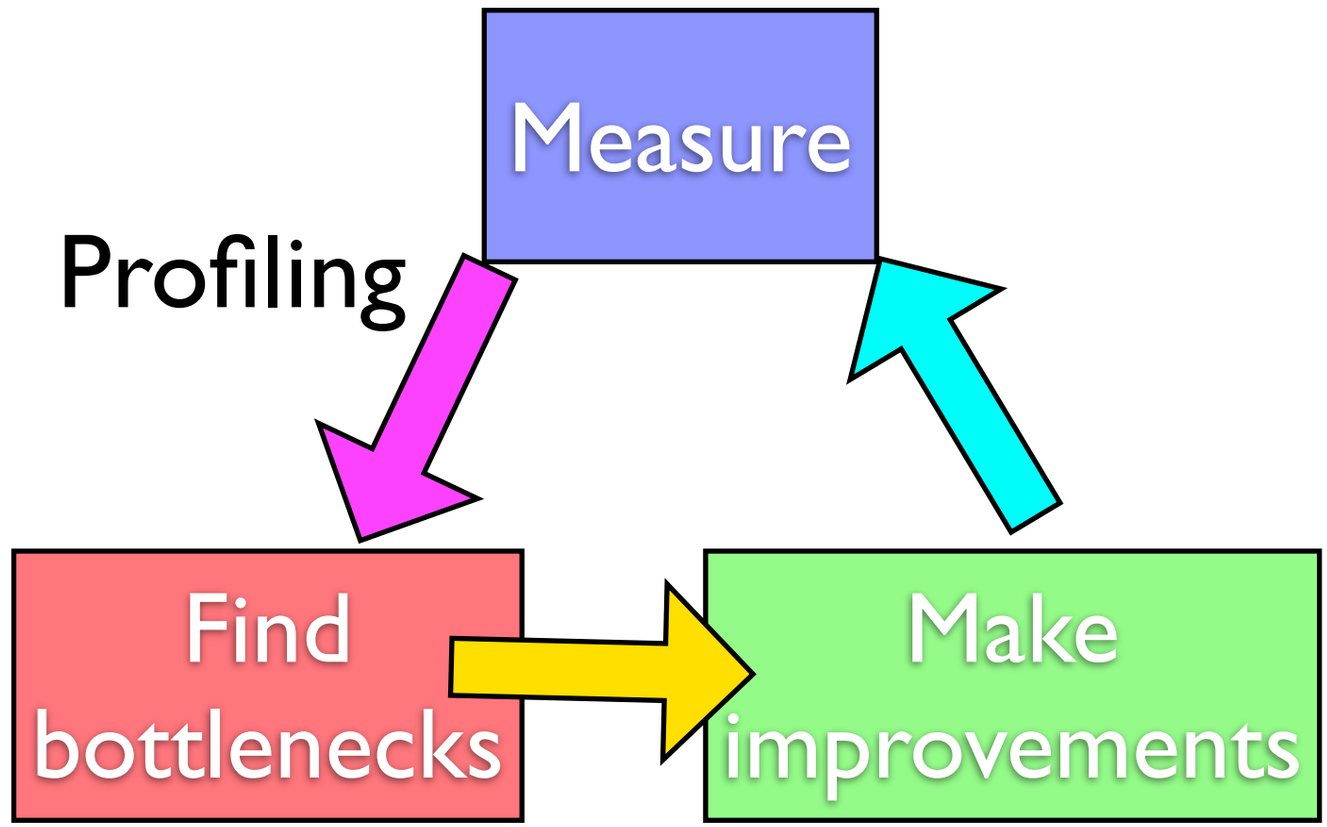
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet

How to improve Performance?

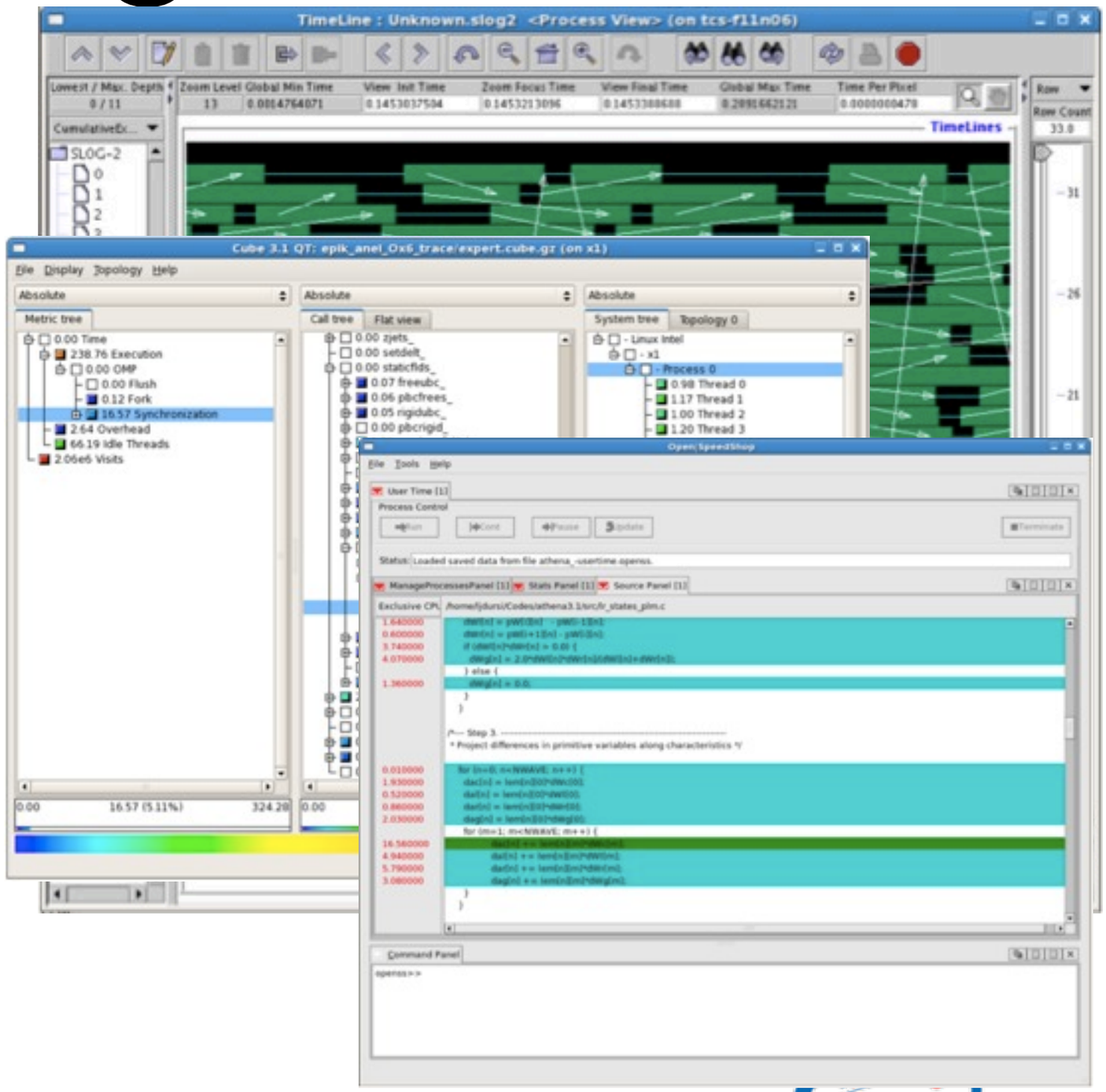
- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



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.

```
      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]
```

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
```

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%)

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

```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.

```c
/* 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

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

SciNet

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

SciNet

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.

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.

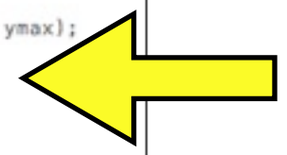
```
        case SIM_PROJECTILE:
            ymin = xmin = 0.;
            ymax = xmax = 1.;
            dx = (xmax-xmin)/npts;
            dy = (ymax-ymin)/npts;
            init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
            projectile_initvalues(&d, psize, pdens, pvel);
  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

```
        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)

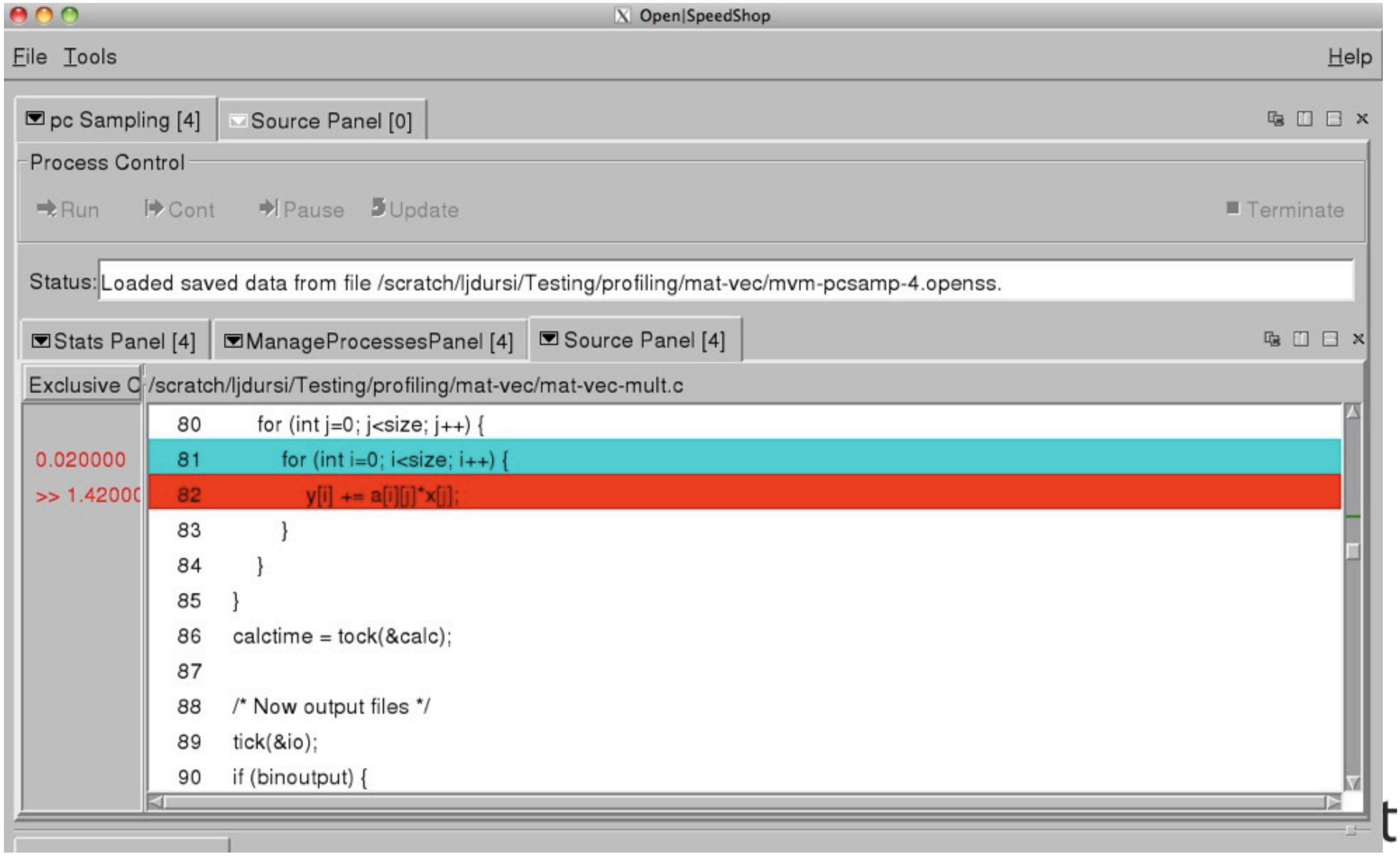
SciNet

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

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'

COMPARE SAVED PERFORMANCE DATA: I have two saved performance experiment data files that I w

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.
- FPE: I would like to know how many times my program is causing Floating Point Exceptions and where in

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

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!

Command Panel

openss>>

Compare Experiments [5]

tus: Experiment 5 is being compared with experiment 7

Stats Panel [5] | Source Panel [5]

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

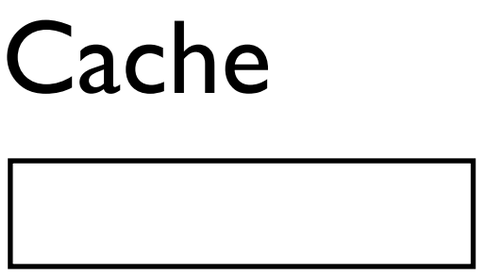
```
70    tick(&calc);
71    if (transpose) {
72        #pragma omp parallel for default(none) shared(x,y,a,size)
73        for (int i=0; i<size; i++) {
74            for (int j=0; j<size; j++) {
75                y[i] += a[i][j]*x[j];
76            }
77        }
78    } else {
79        #pragma omp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

0.01000

SciNet

Cache Thrashing

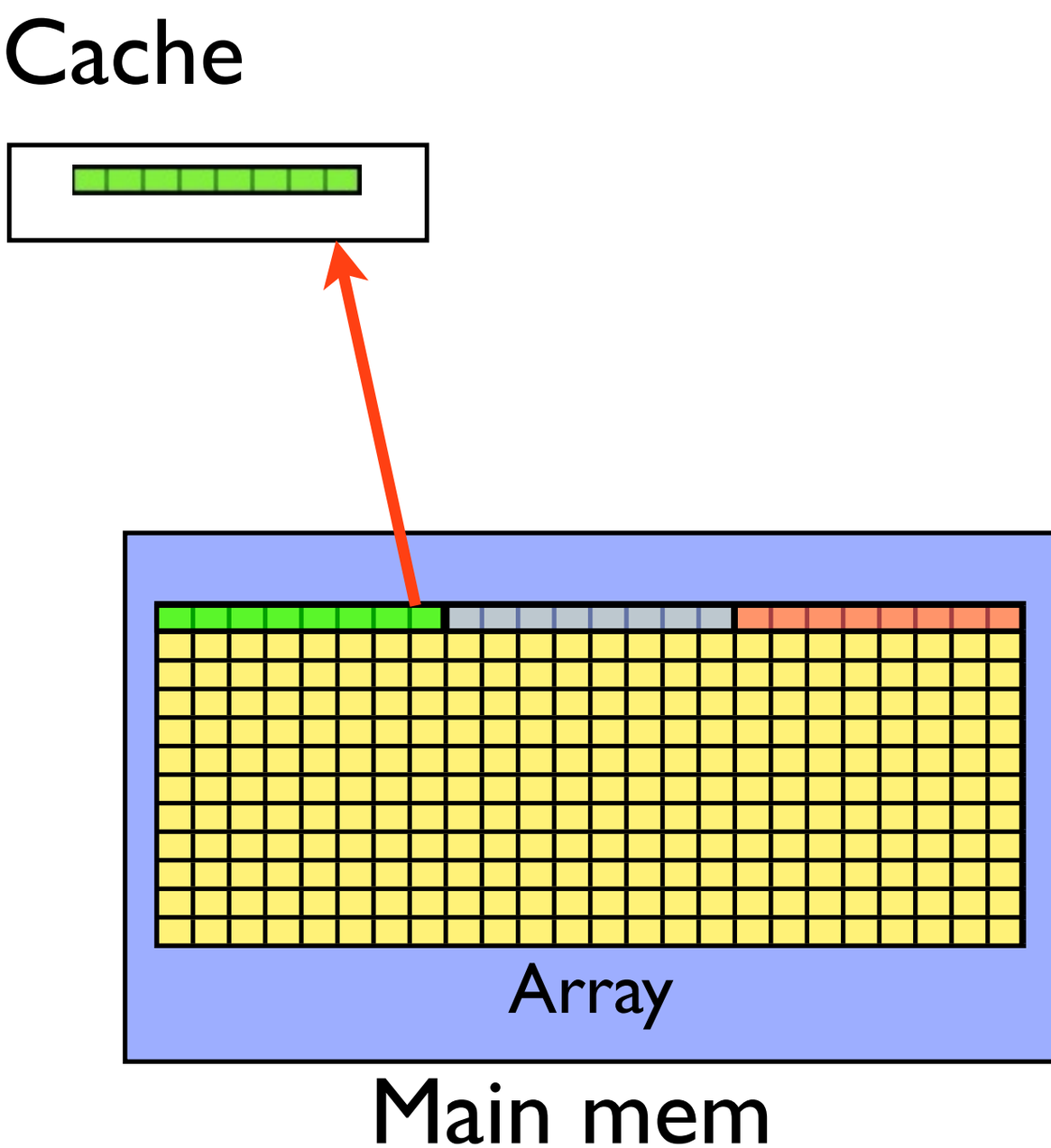
- Memory bandwidth is key to getting good performance on modern systems
- Main Mem - big, slow
- Cache - small, fast
  - Saves recent accesses, a line of data at a time



SciNet

Cache Thrashing

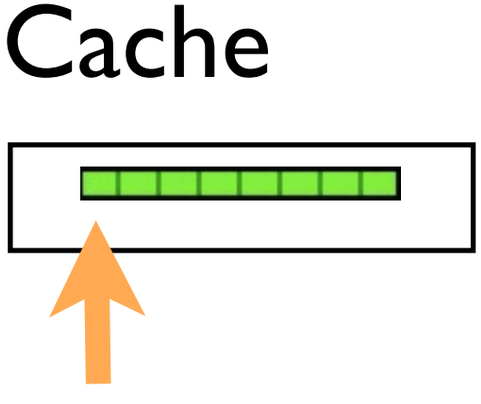
- When accessing memory in order, only one access to slow main mem for many data points
- Much faster

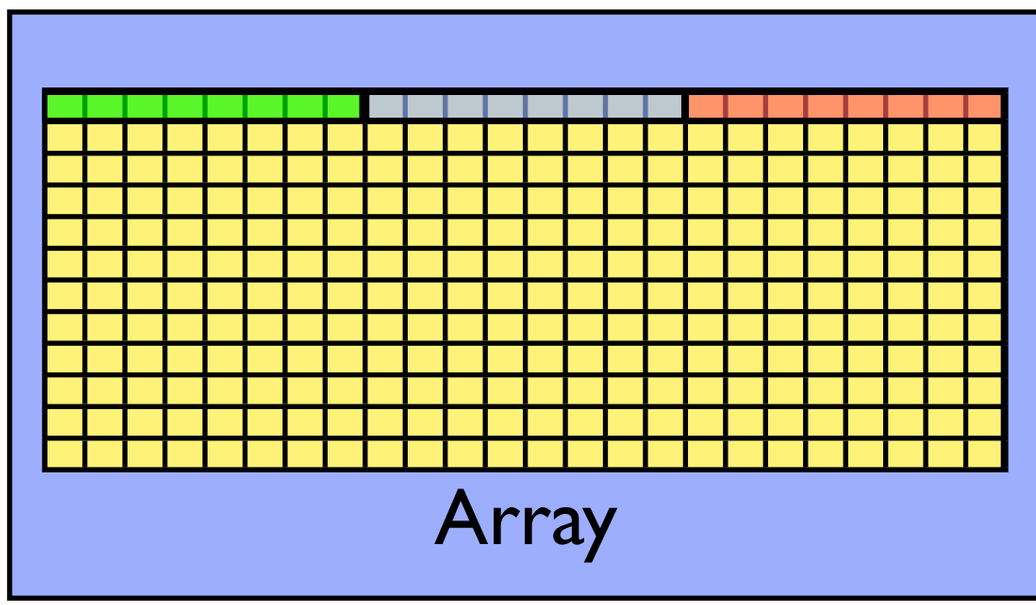


SciNet

Cache Thrashing

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

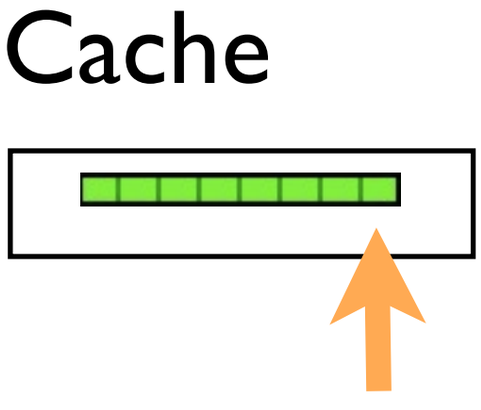




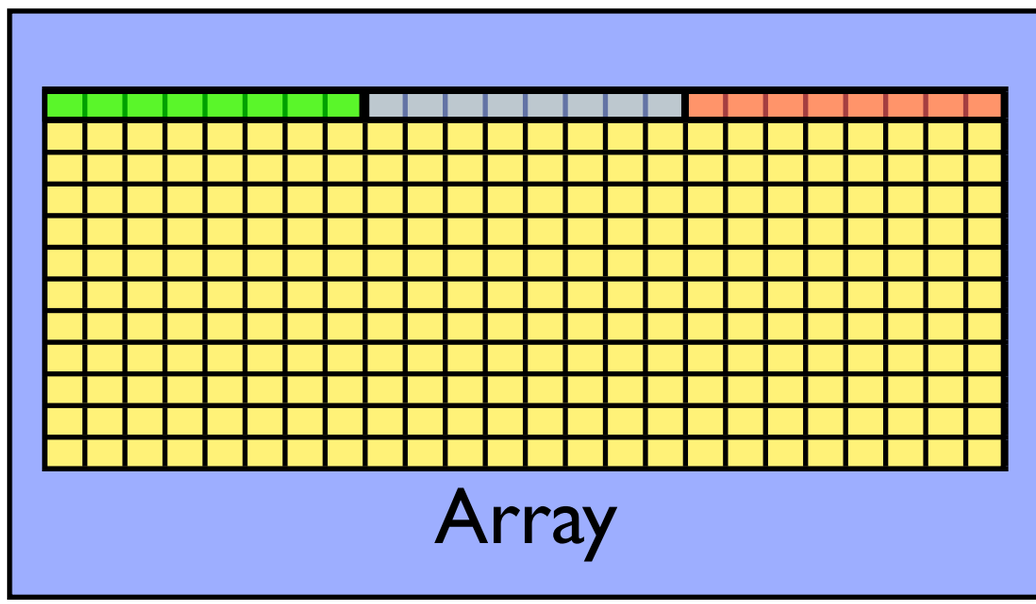
Main mem

SciNet

Cache Thrashing



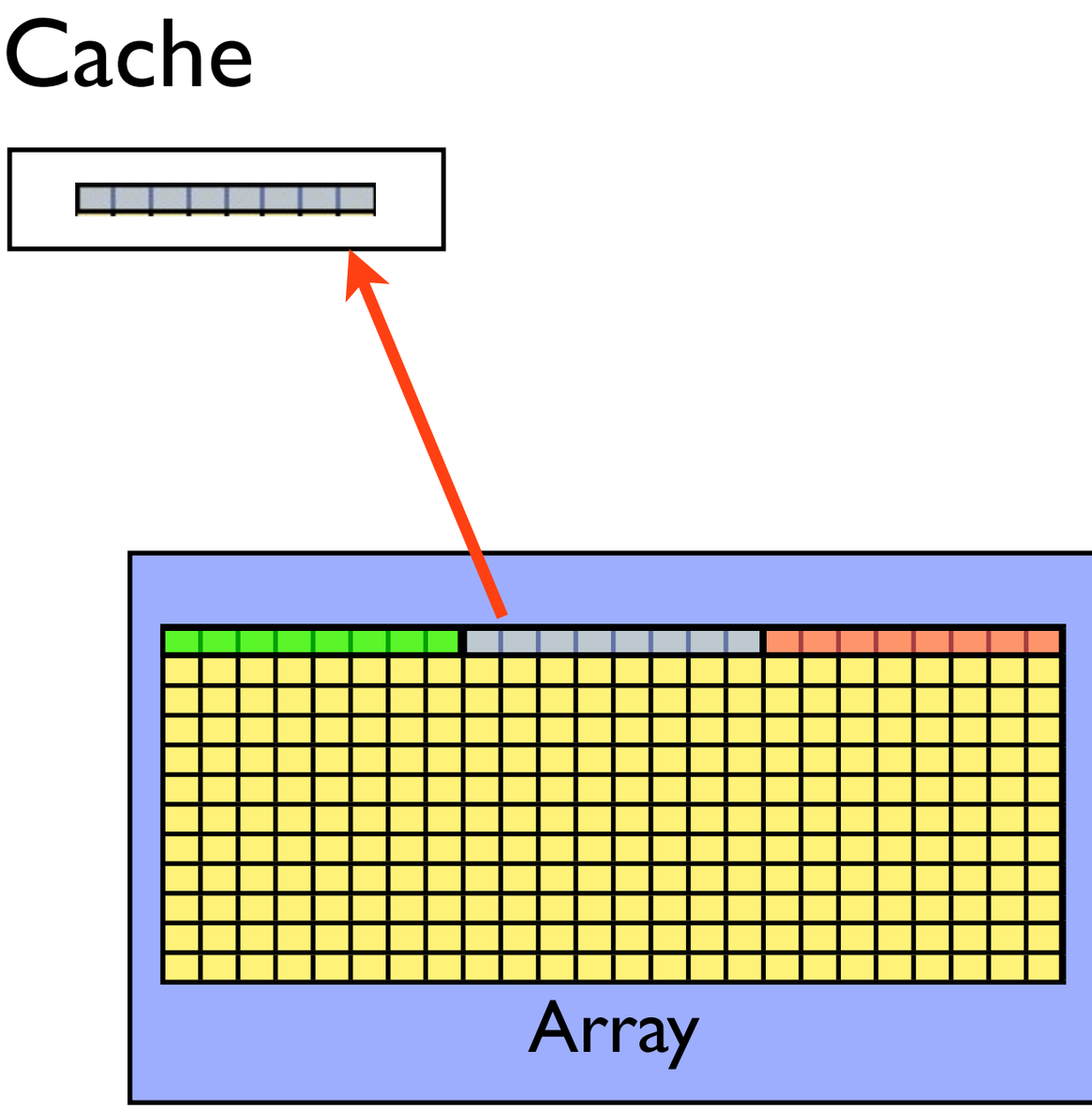
- When accessing memory in order, only one access to slow main mem for many data points
- Much faster



SciNet

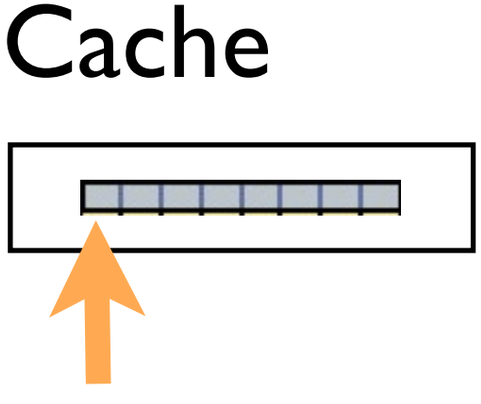
Cache Thrashing

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

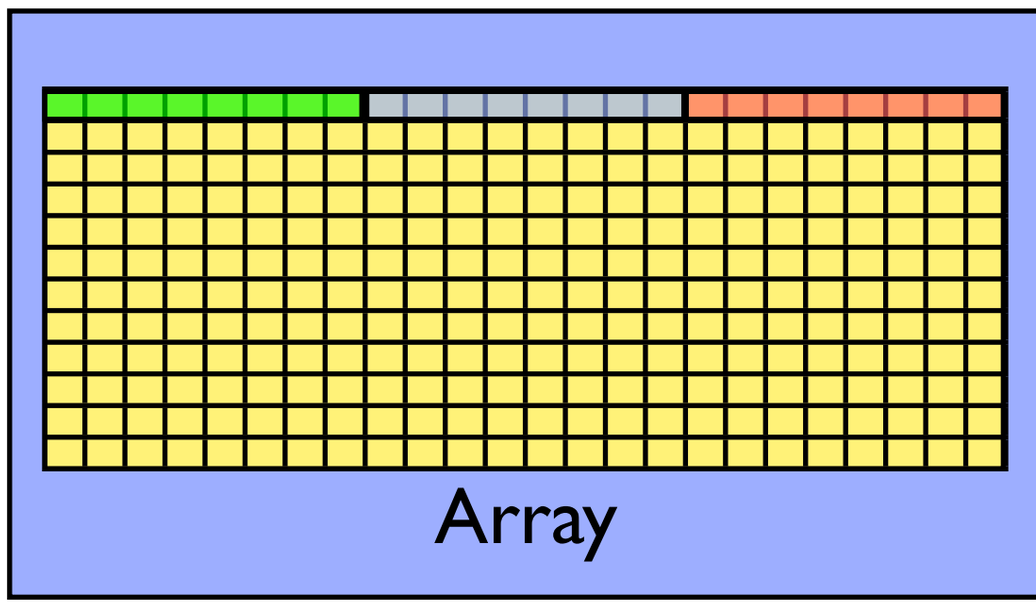


SciNet

Cache Thrashing

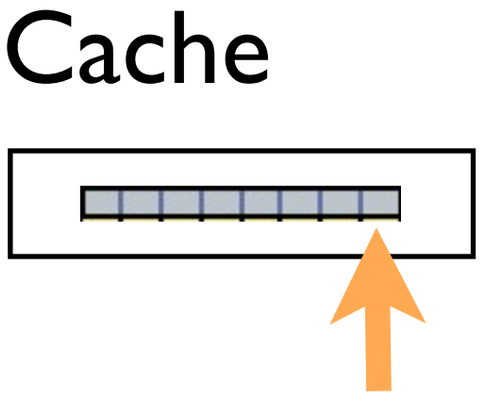


- When accessing memory in order, only one access to slow main mem for many data points
- Much faster

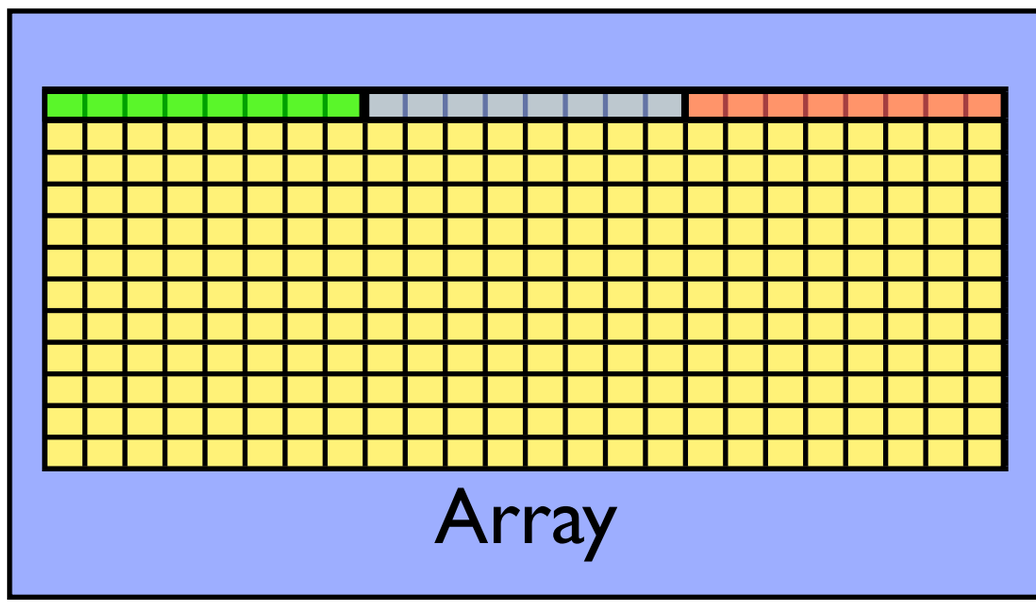


SciNet

Cache Thrashing



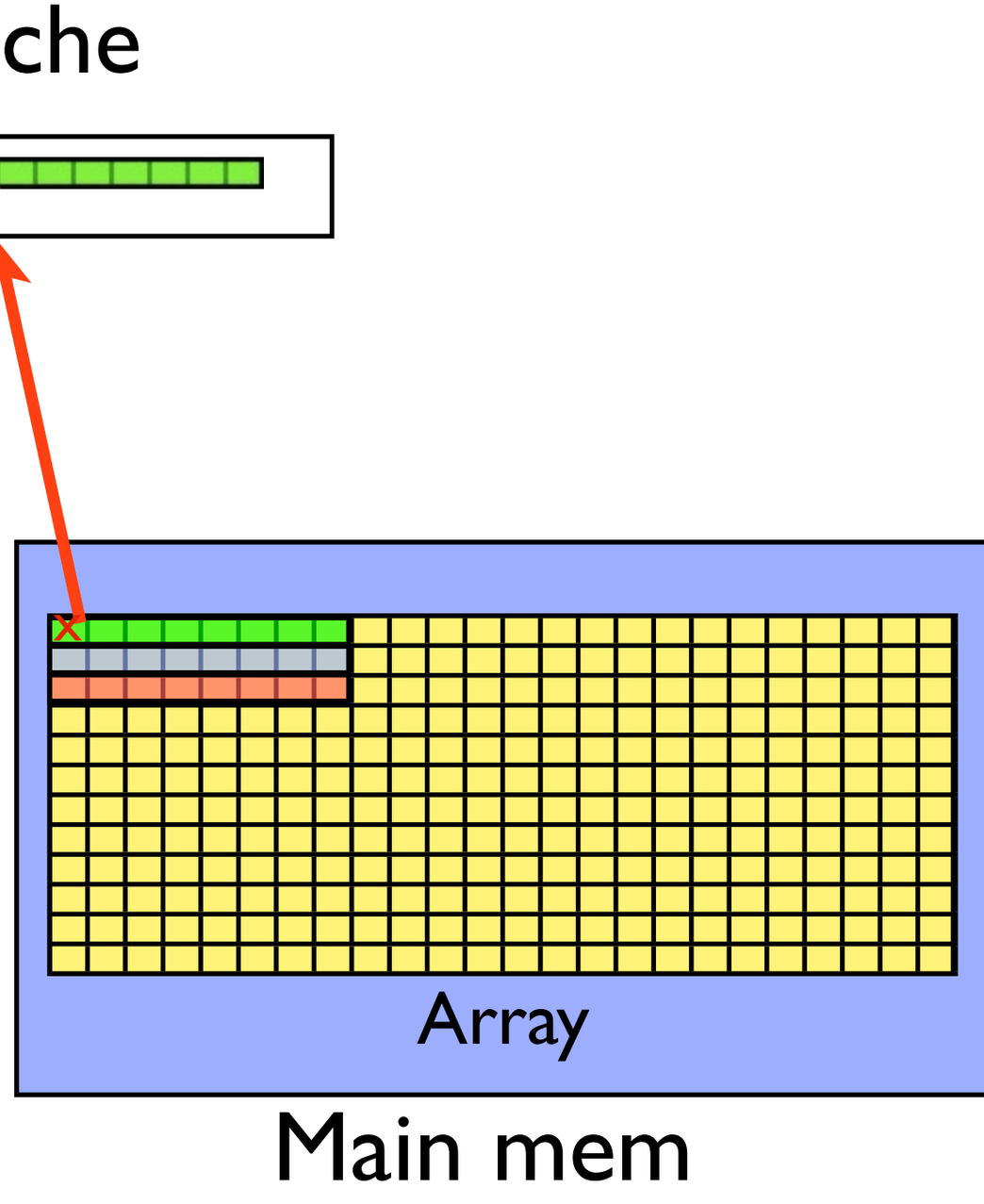
- When accessing memory in order, only one access to slow main mem for many data points
- Much faster



SciNet

Cache Thrashing

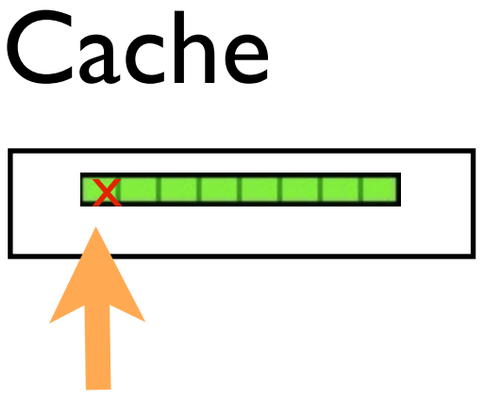
- When accessing memory out of order, much worse
- Each access is new cache line (cache miss)- slow access to main memory

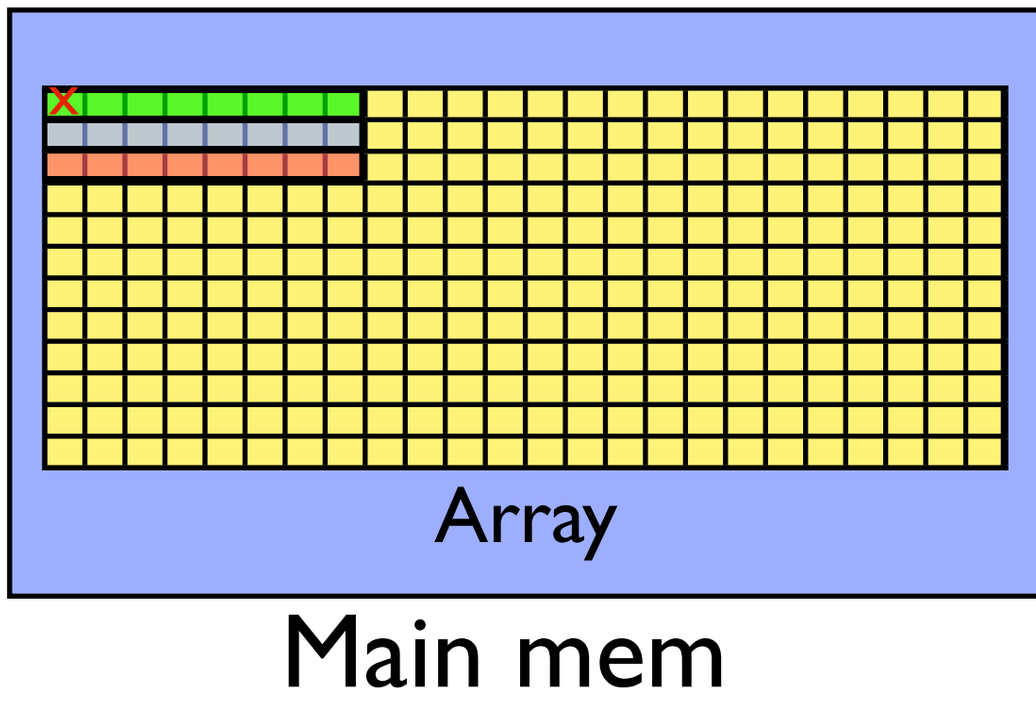


SciNet

Cache Thrashing

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

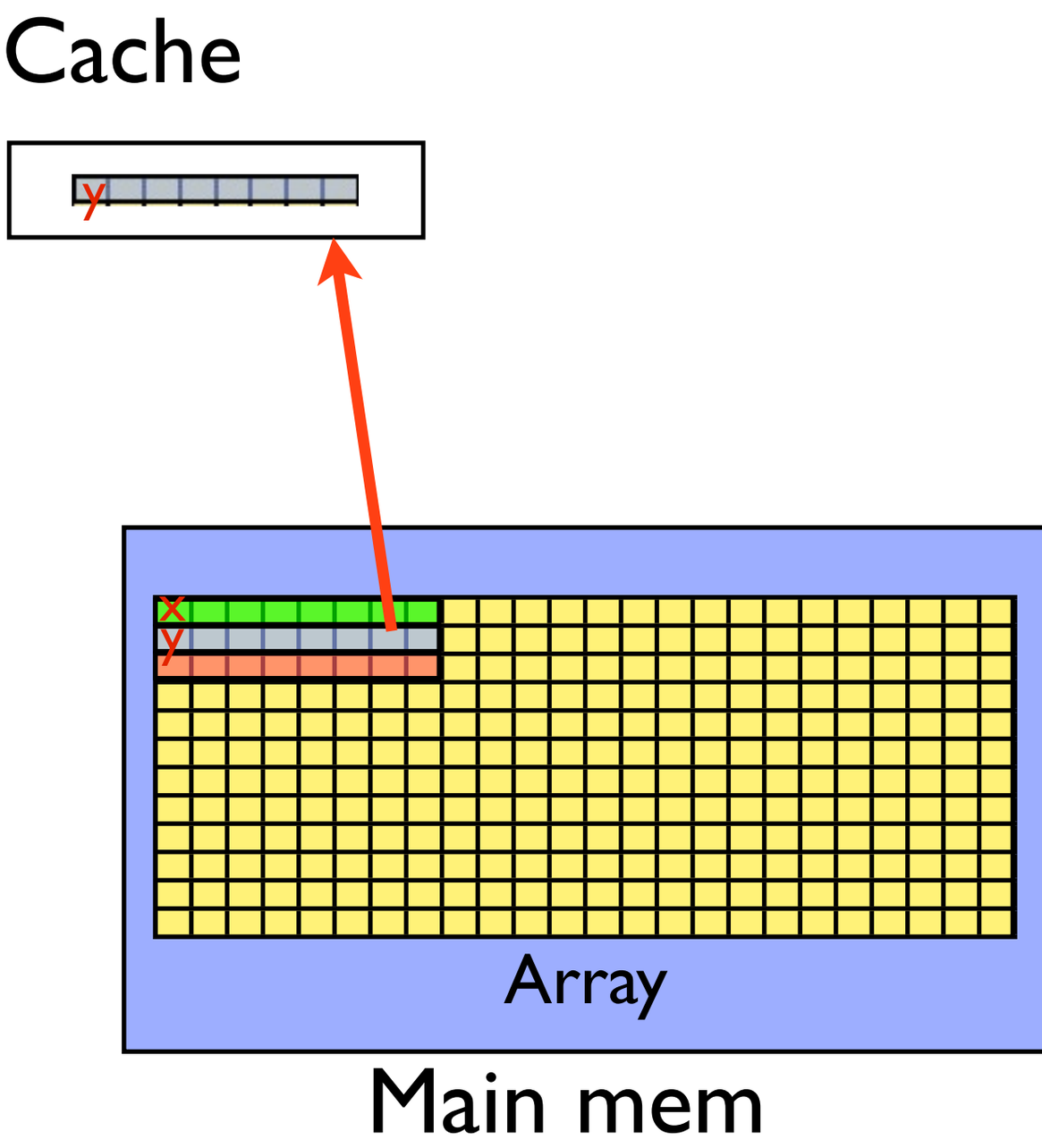




SciNet

Cache Thrashing

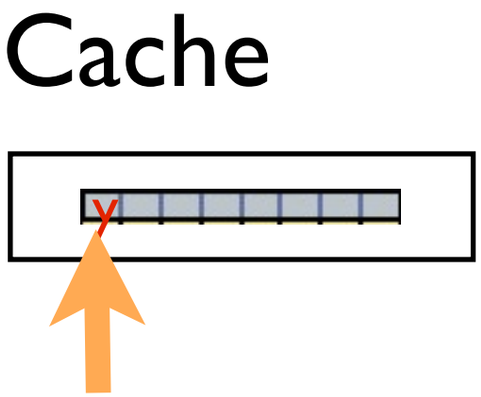
- When accessing memory out of order, much worse
- Each access is new cache line (cache miss)- slow access to main memory

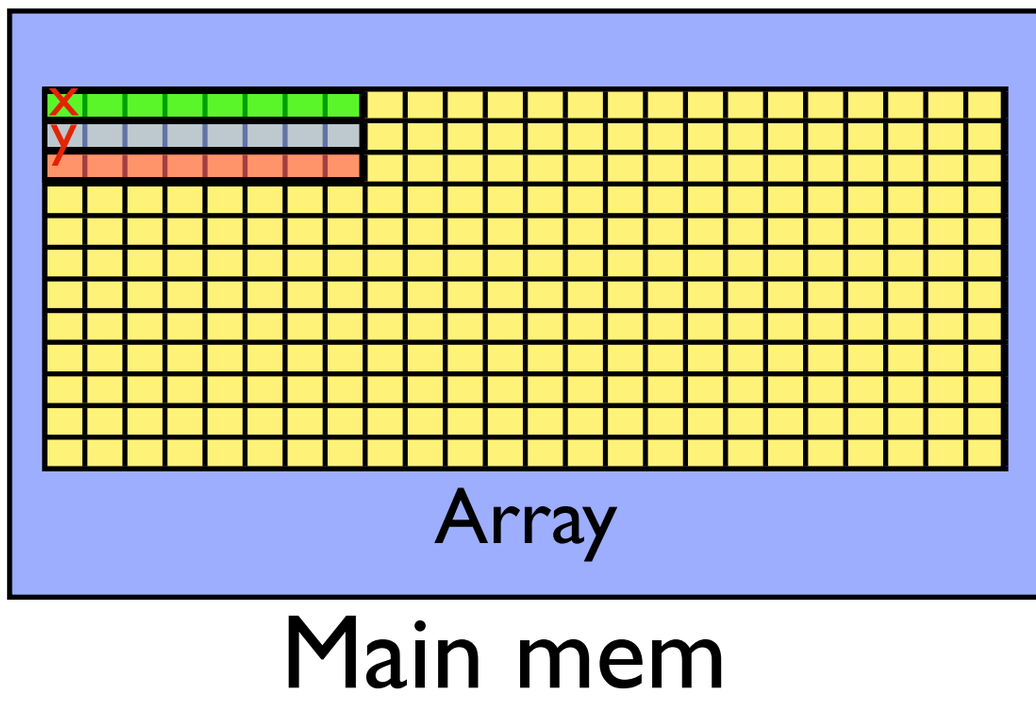


SciNet

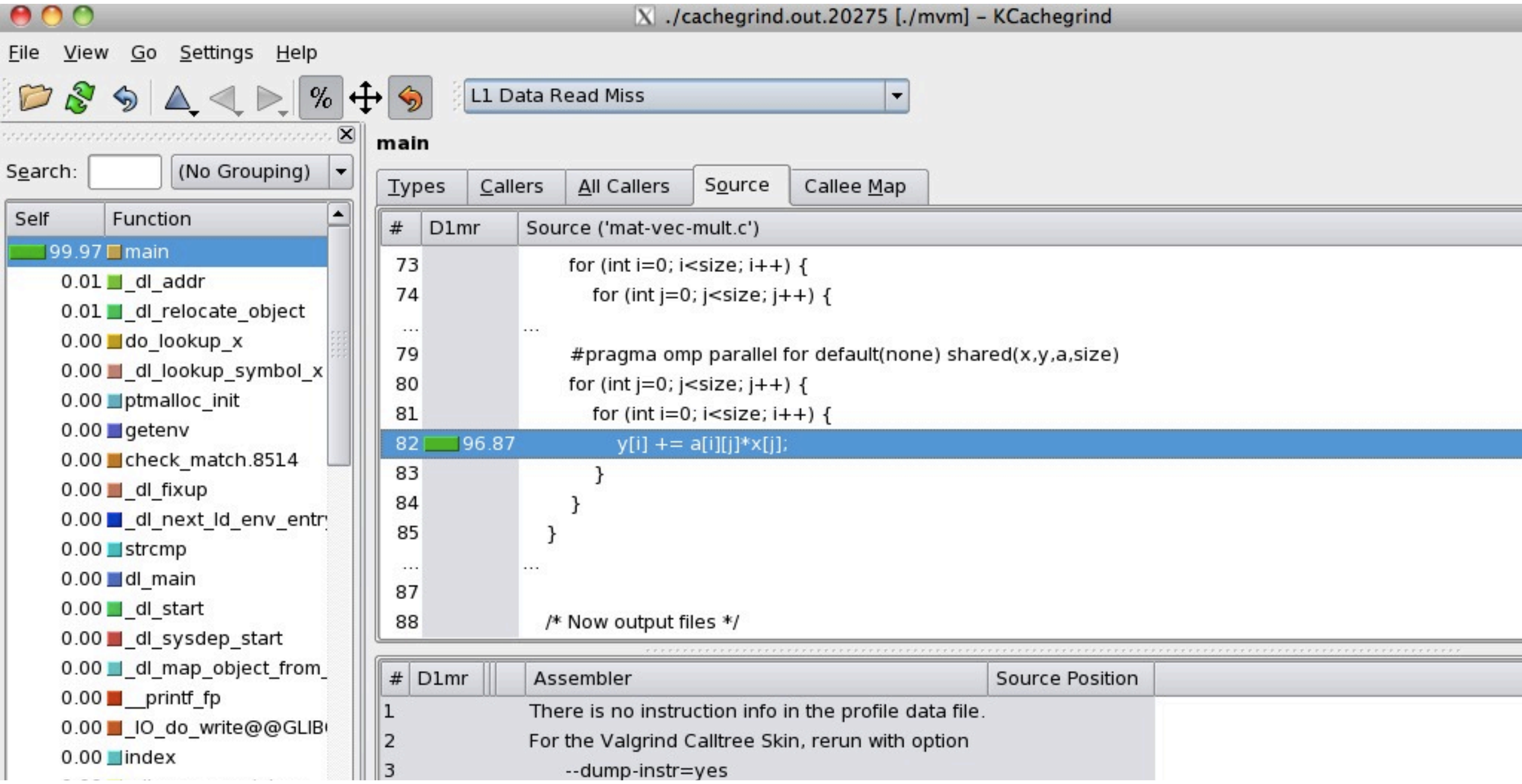
Cache Thrashing

- 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

SciNet

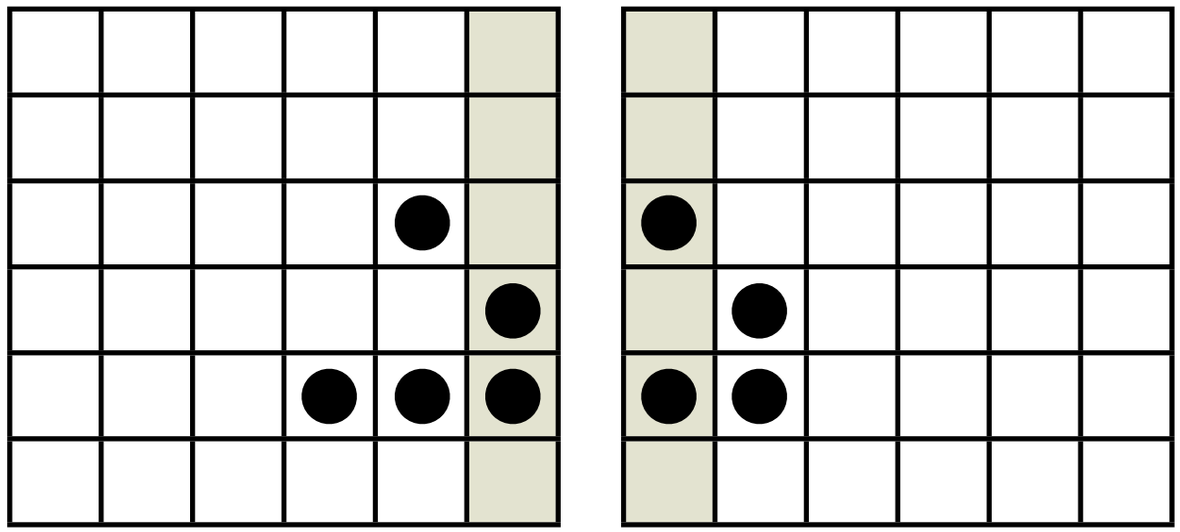
Open|Speedshop

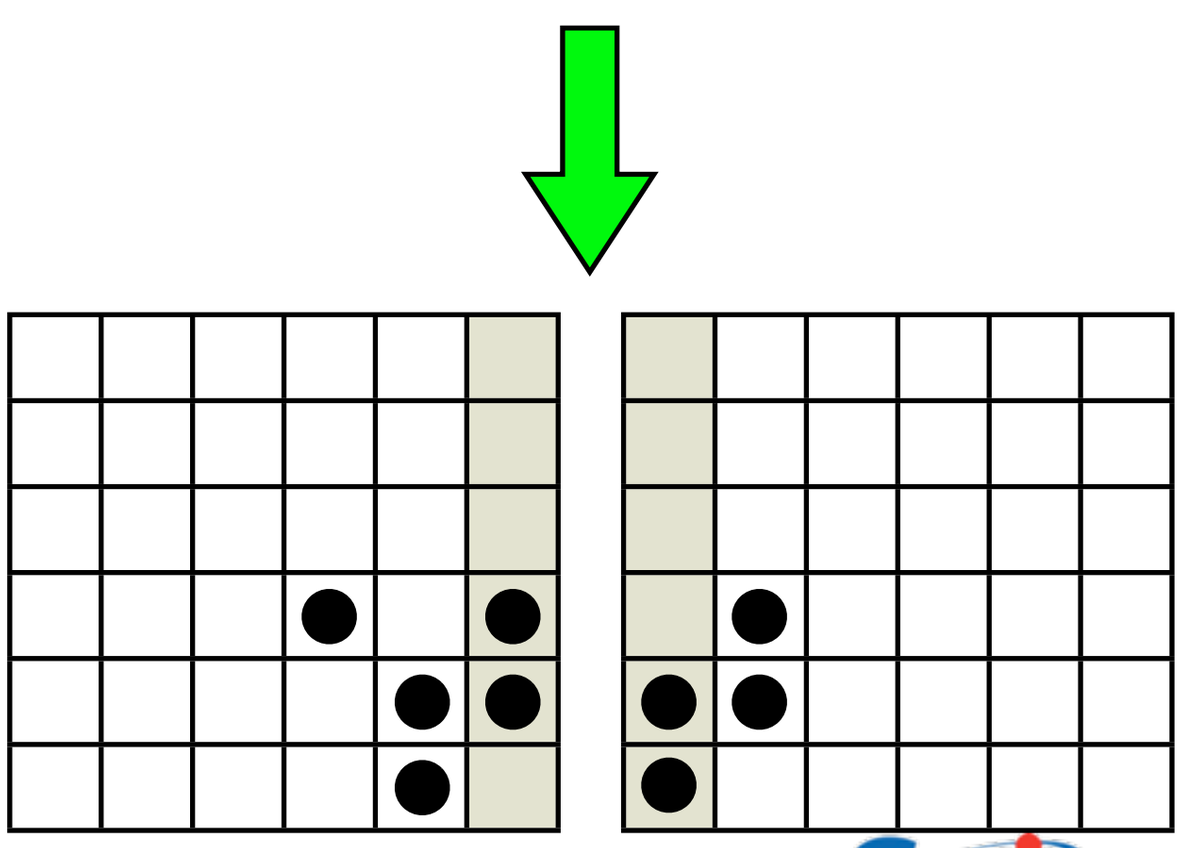
- Also has very powerful UNIX command line tools "openss -f `./mvm --transpose' pcsamp" and python scripting interface.
- Experiments: pcsamp (gprof), usertime (includes call graph), iot (I/O tracing - find out where I/O time is being spent), mpit (MPI tracing)

SciNet

Game of Life

- Simple MPI implementation of Conway game of life
- Live cell with 2,3 neighbours lives; 0-1 starves; 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

IPM profile for 3756669.gpc-sched

file:///Users/ljdursi/Desktop/ipm/gol_newswitch/index.html

Google

FindStuff

3756669.gpc-sched

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memory Usage
- Executable Info
- Host List
- Environment
- Developer Info

Powered by IPM

command: ./gameoflife --infilename=bigin.txt --overlap			
codename:	unknown	state:	running
username:	ljdursi	group:	scinet
host:	gpc-f109n007 (x86_64_Linux)	mpi_tasks:	16 on 2 hosts
start:	12/06/10/15:27:53	wallclock:	3.93598e+00 sec
stop:	12/06/10/15:27:57	%comm:	0.230736134965117
total memory:	1.803652 gbytes	total gflop/sec:	-4.06506130620582
switch(send):	0.0341888386756182 gbytes	switch(recv):	0.0331104658544064 gbytes

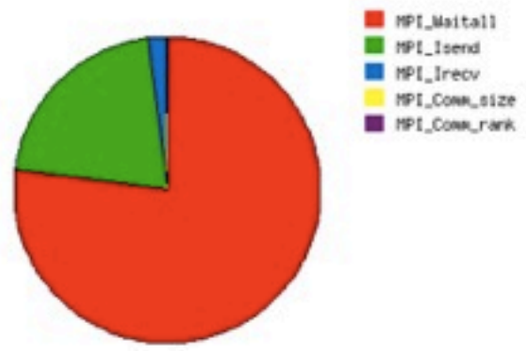
Computation

Event	Count	Pop
NULL	0	*

(Hardware counters coming soonish)

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)
NULL	*	0.00	0 (0)	0 (0)

Communication Event Statistics (100.00% detail, -2.4880e-08 error)

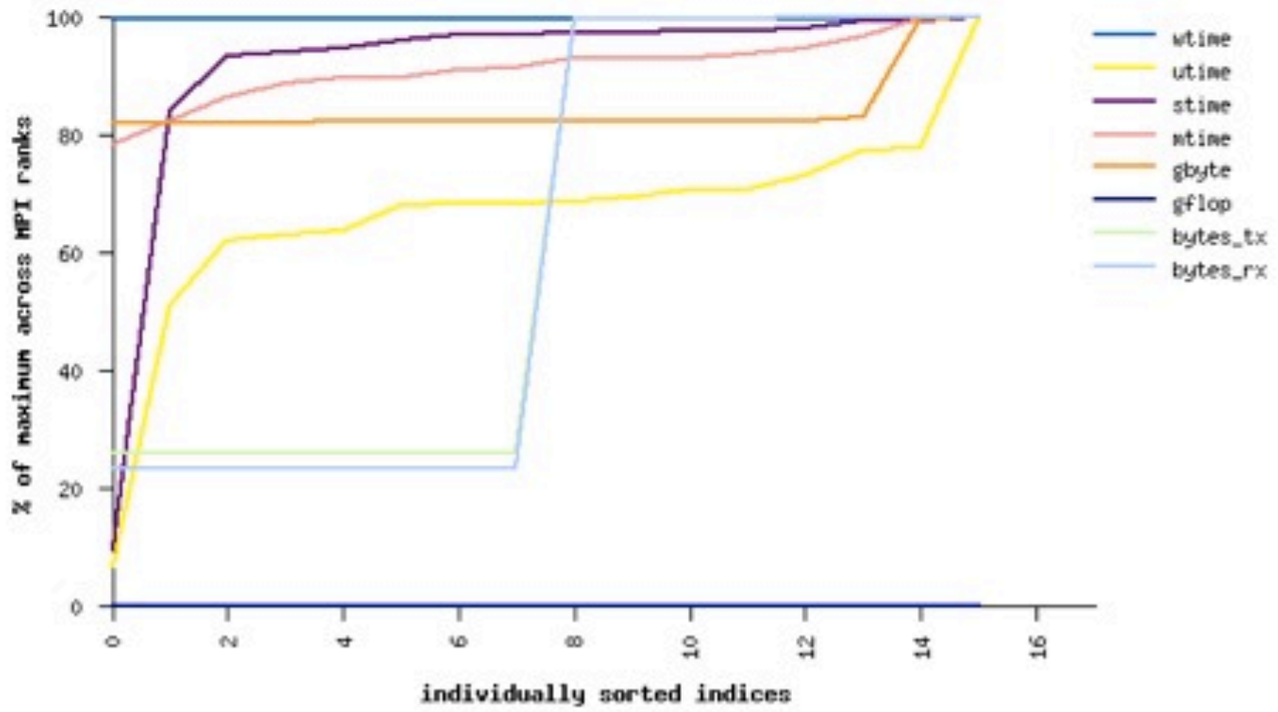
	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Waitall	164	1200	0.085	5.125e-06	6.507e-04	58.26	0.13
MPI_Waitall	156	400	0.027	9.381e-06	4.927e-04	18.89	0.04
MPI_Isend	1	6400	0.017	3.192e-06	4.934e-05	12.01	0.03
MPI_Isend	25	3200	0.010	3.867e-06	2.632e-05	7.14	0.02

Load balance by task: HPM counters

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

SciNet

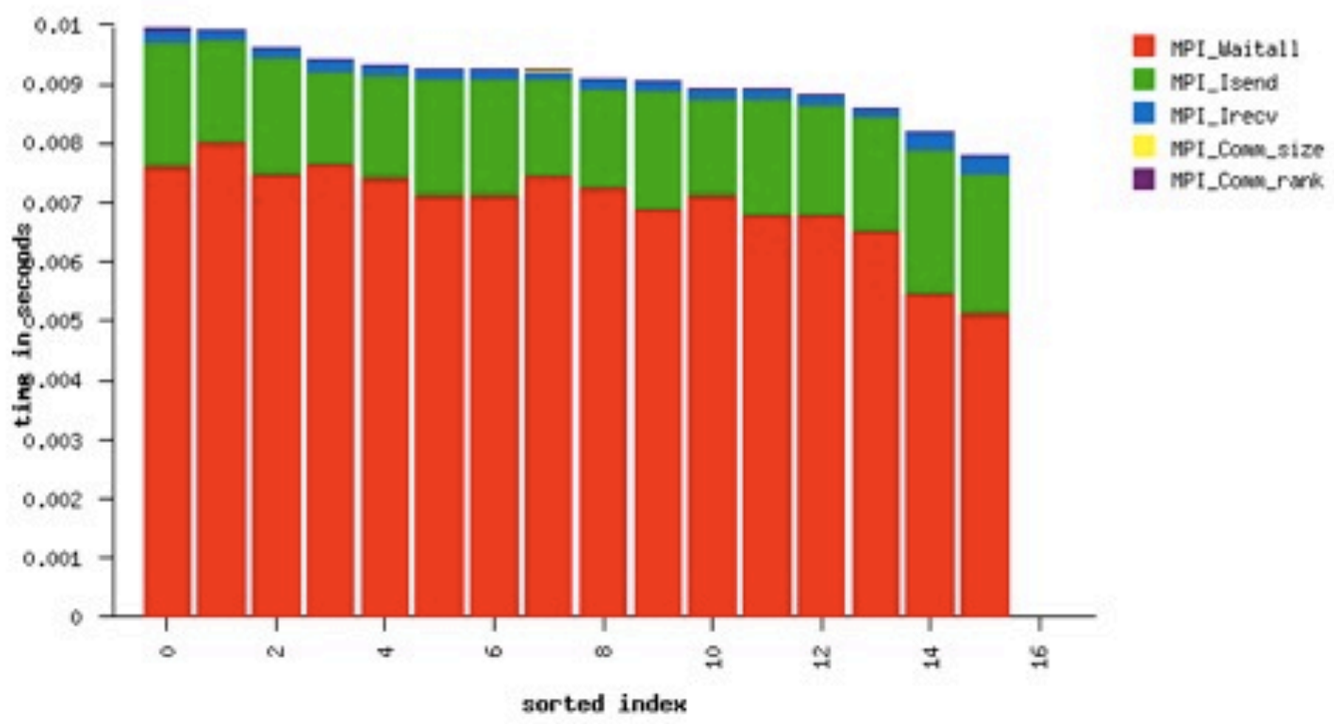
Load balance by task: memory, flops, timings



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

by MPI rank, by MPI time

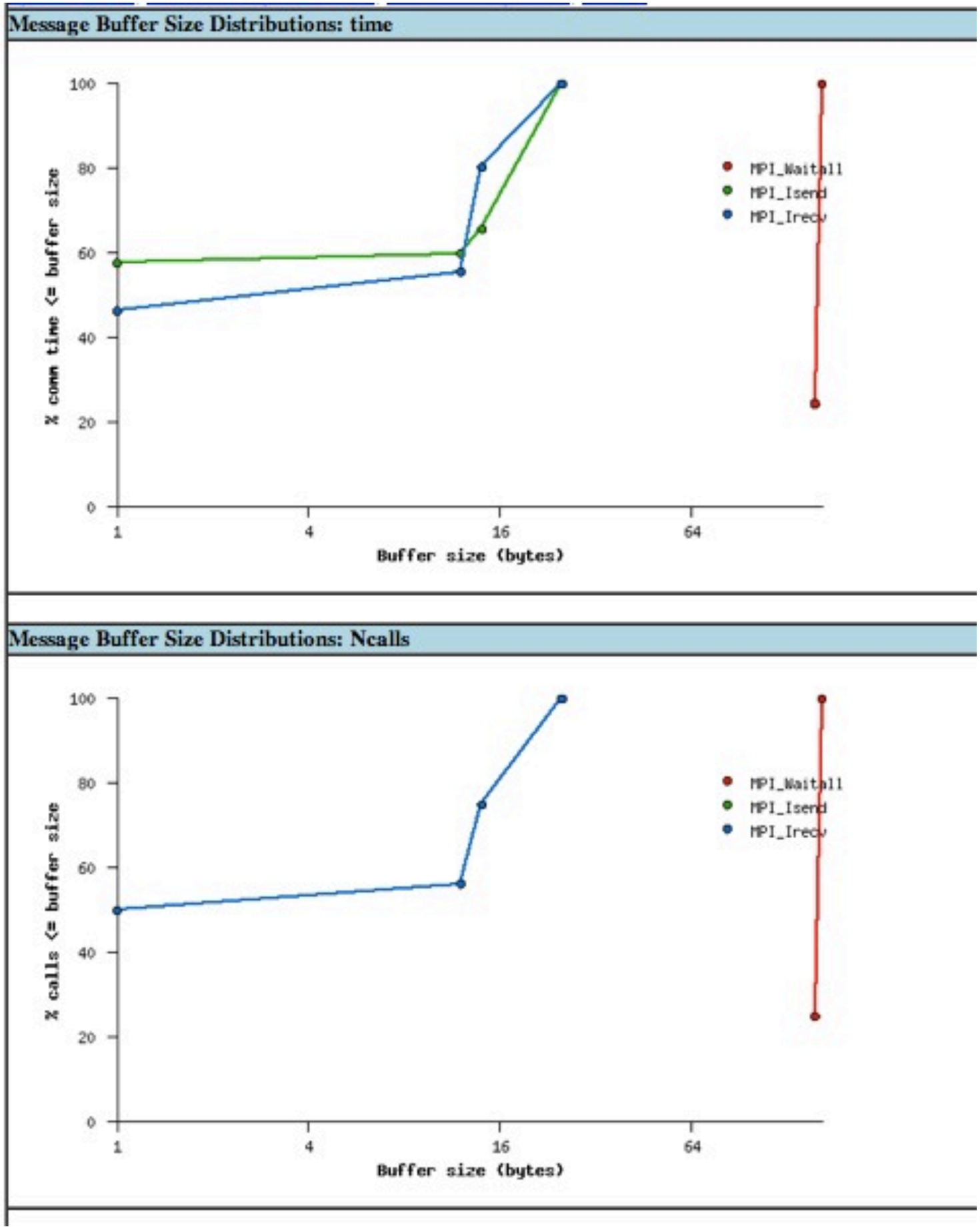
Communication balance by task (sorted by MPI time)



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

Message Buffer Size Distributions: time

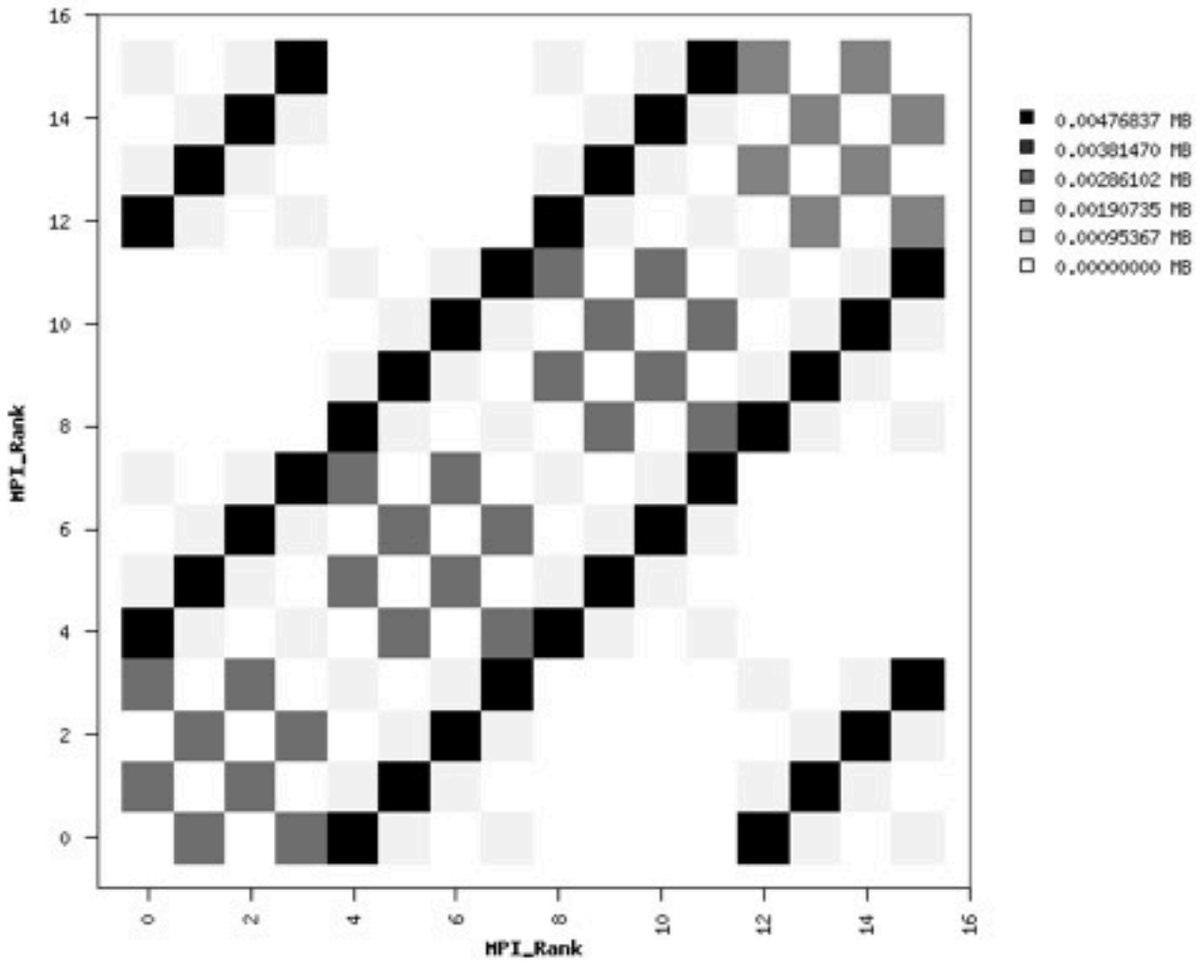
SciNet



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

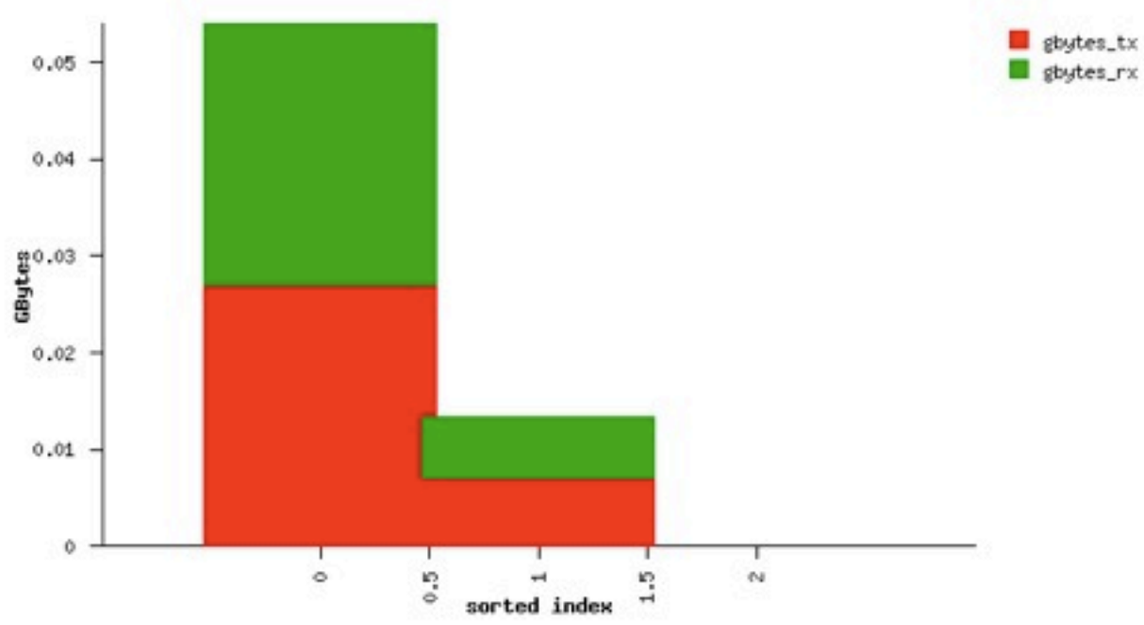
SciNet

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)

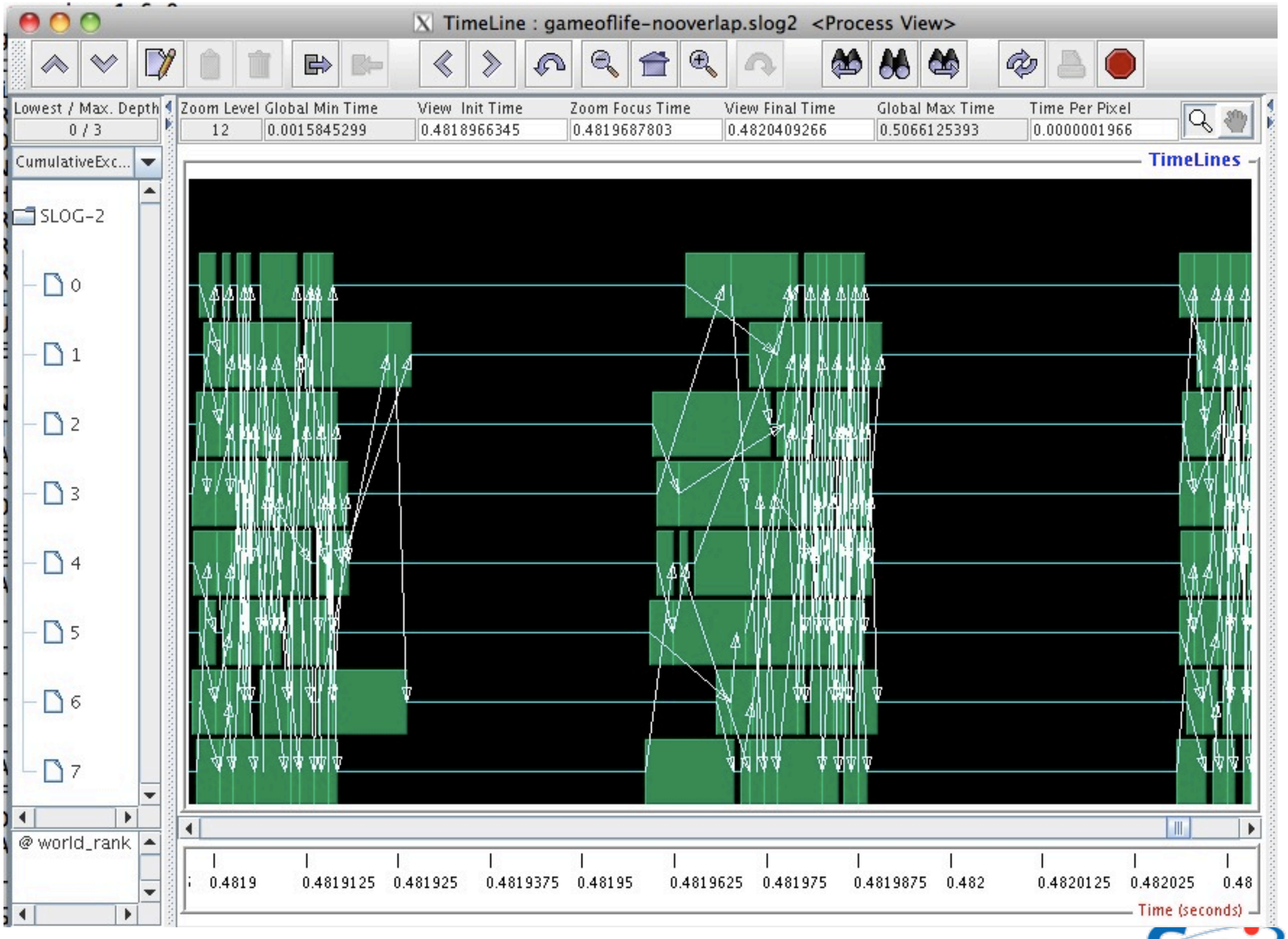
SciNet

MPE/Jumpshot

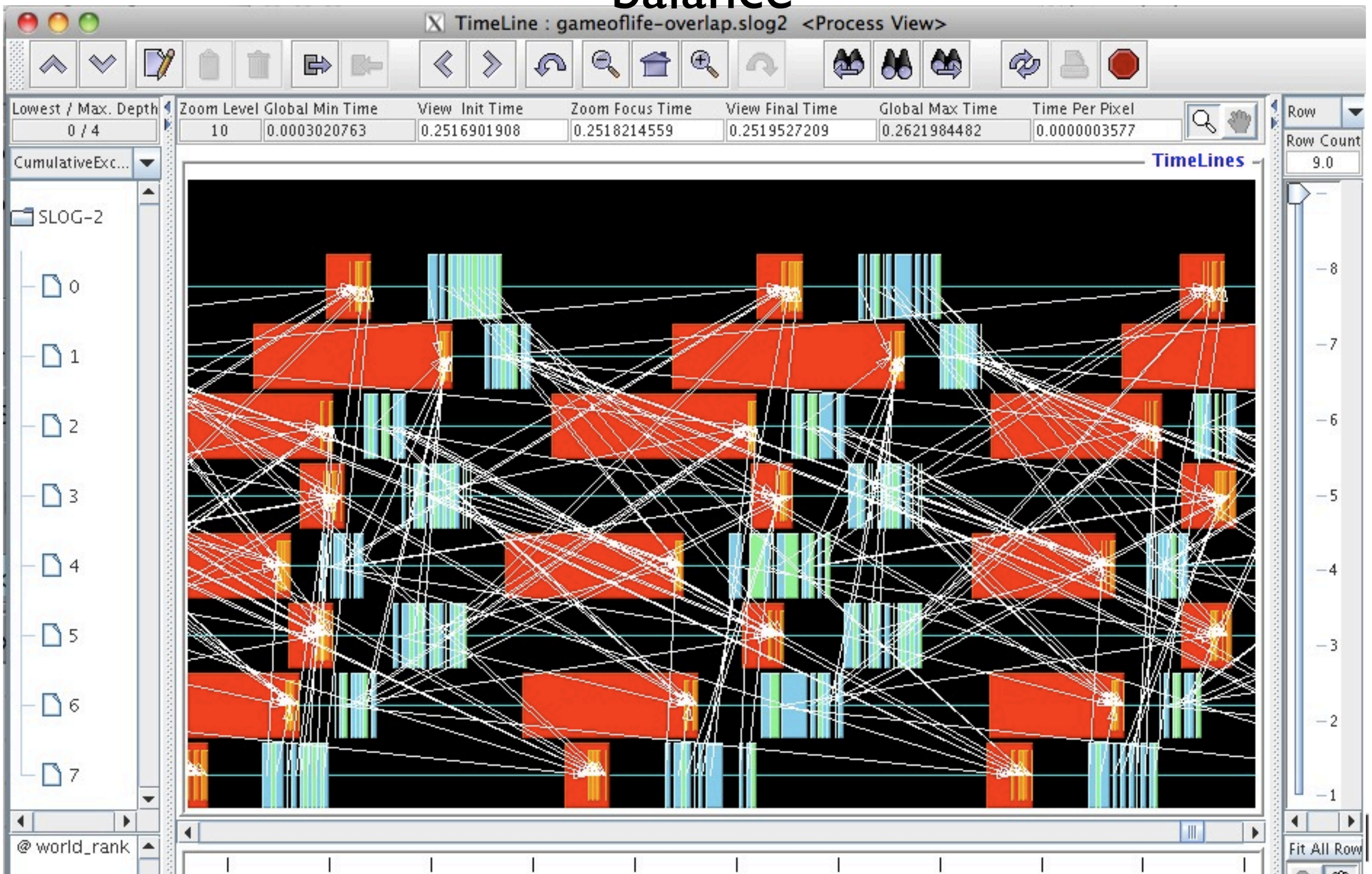
- More detailed view of MPI calls
- Rather than just counting, actually logs every MPI call, can then be visualized.
- Higher overhead - more detailed data.

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

SciNet


TimeLine : gameoflife-nooverlap.slog2 <Process View>
Lowest / Max. Depth
0 / 3
Zoom Level
12
Global Min Time
0.0015845299
View Init Time
0.4818966345
Zoom Focus Time
0.4819687803
View Final Time
0.4820409266
Global Max Time
0.5066125393
Time Per Pixel
0.0000001966
CumulativeExc...
SLOG-2
0
1
2
3
4
5
6
7
@ world_rank
TimeLines
0.4819
0.4819125
0.481925
0.4819375
0.48195
0.4819625
0.481975
0.4819875
0.482
0.4820125
0.482025
0.48
Time (seconds)

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

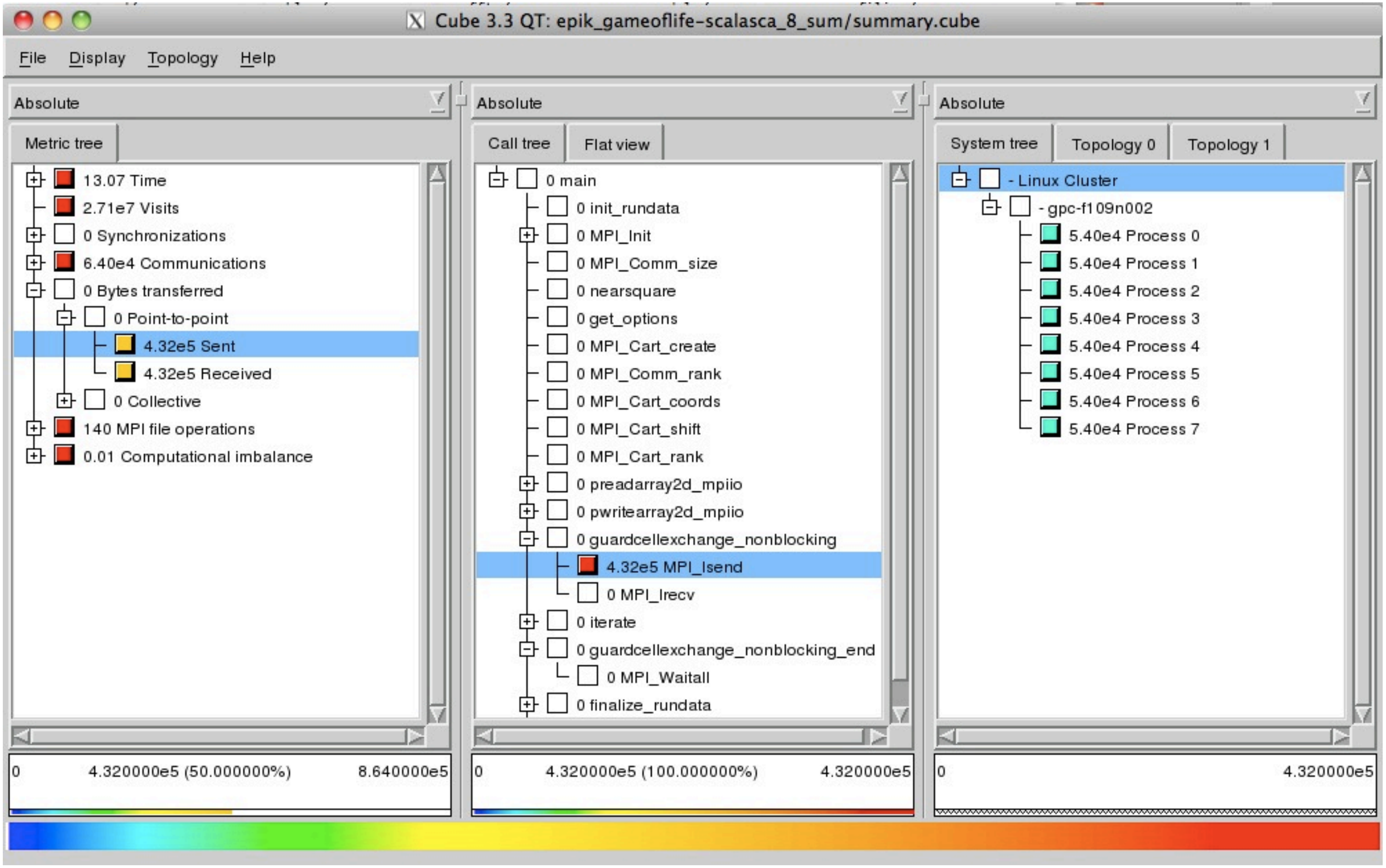


Scalasca - Analysis

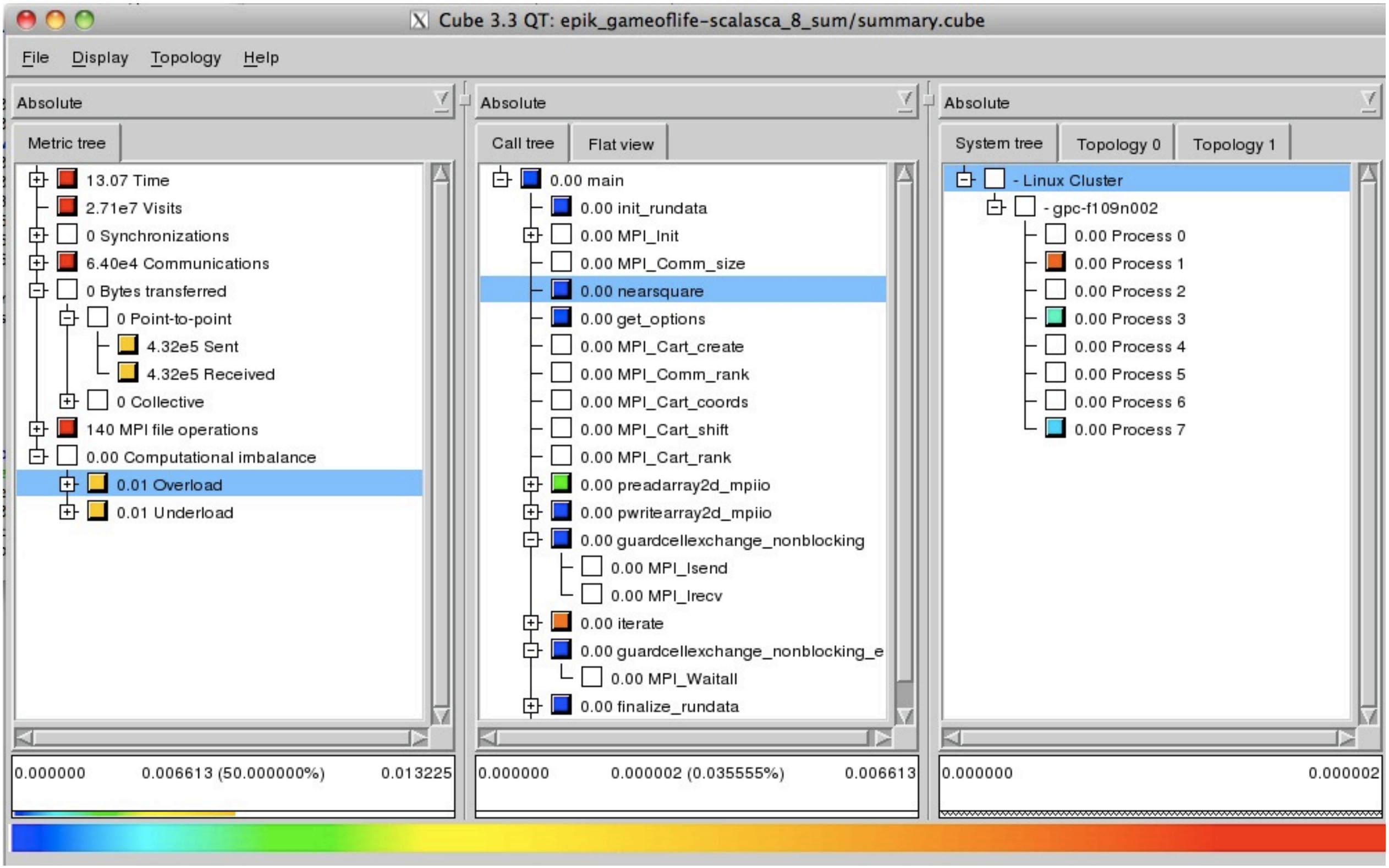
- Low-level automated instrumentation of code.
- High-level analysis of that data.
- Compile, run as normal, but prefix with:
  - compile: scalasca -instrument
  - run: scalasca -analyze
- Then scalasca -examine the resulting directory.

SciNet

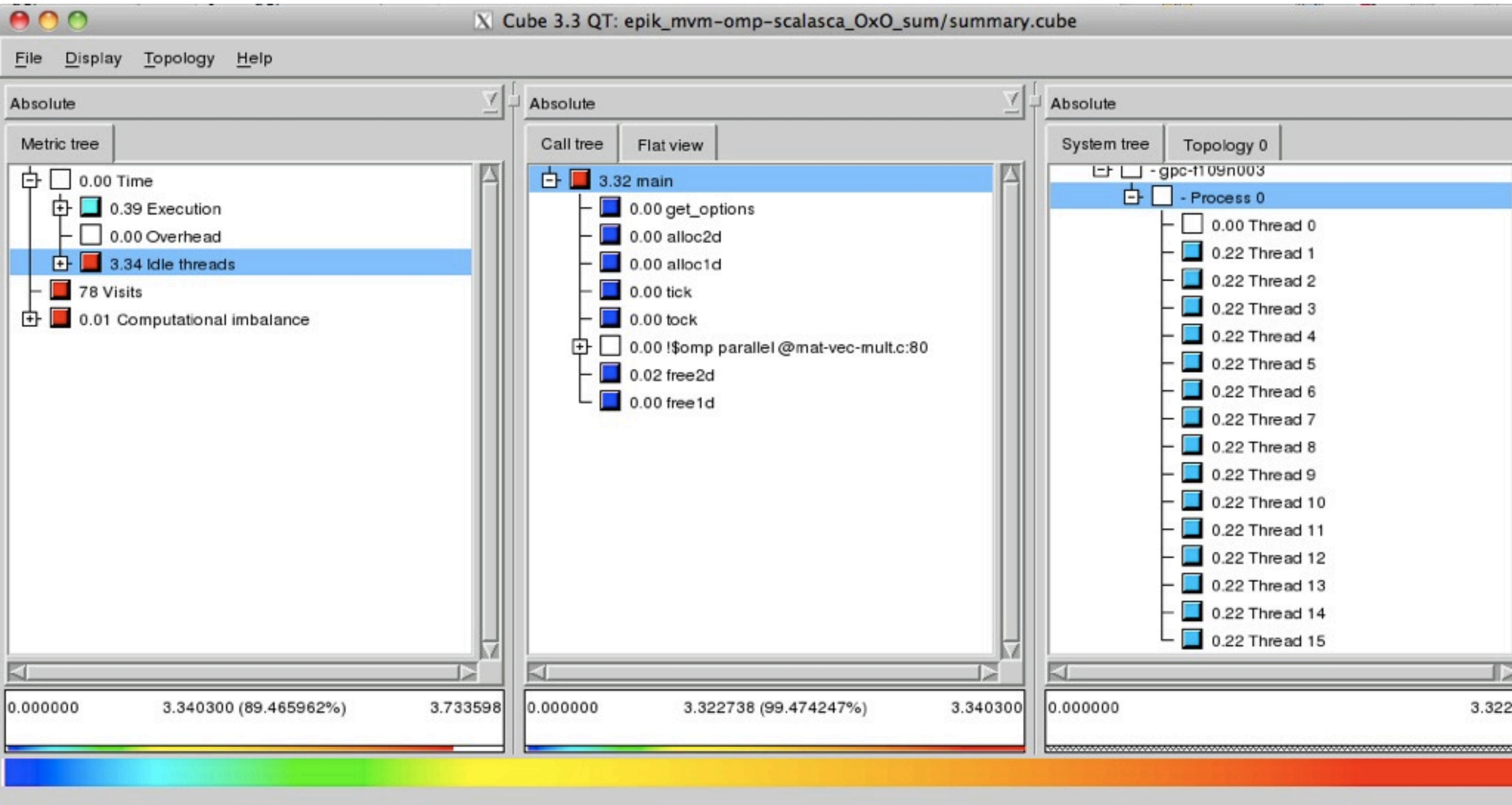
Game of life: can take a look at data sent, received



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