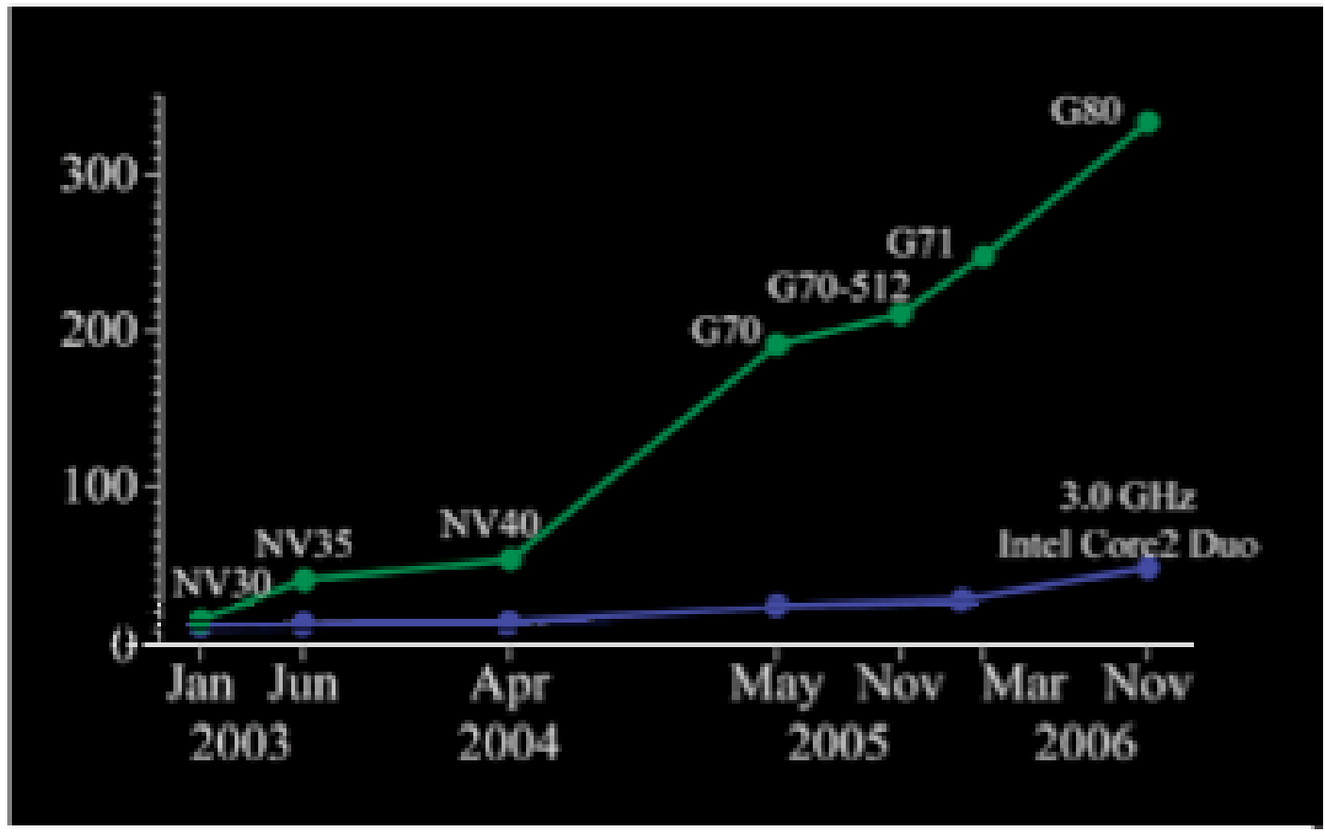
Advanced Topics:
Additional Resources
Performance tools and exotic architectures

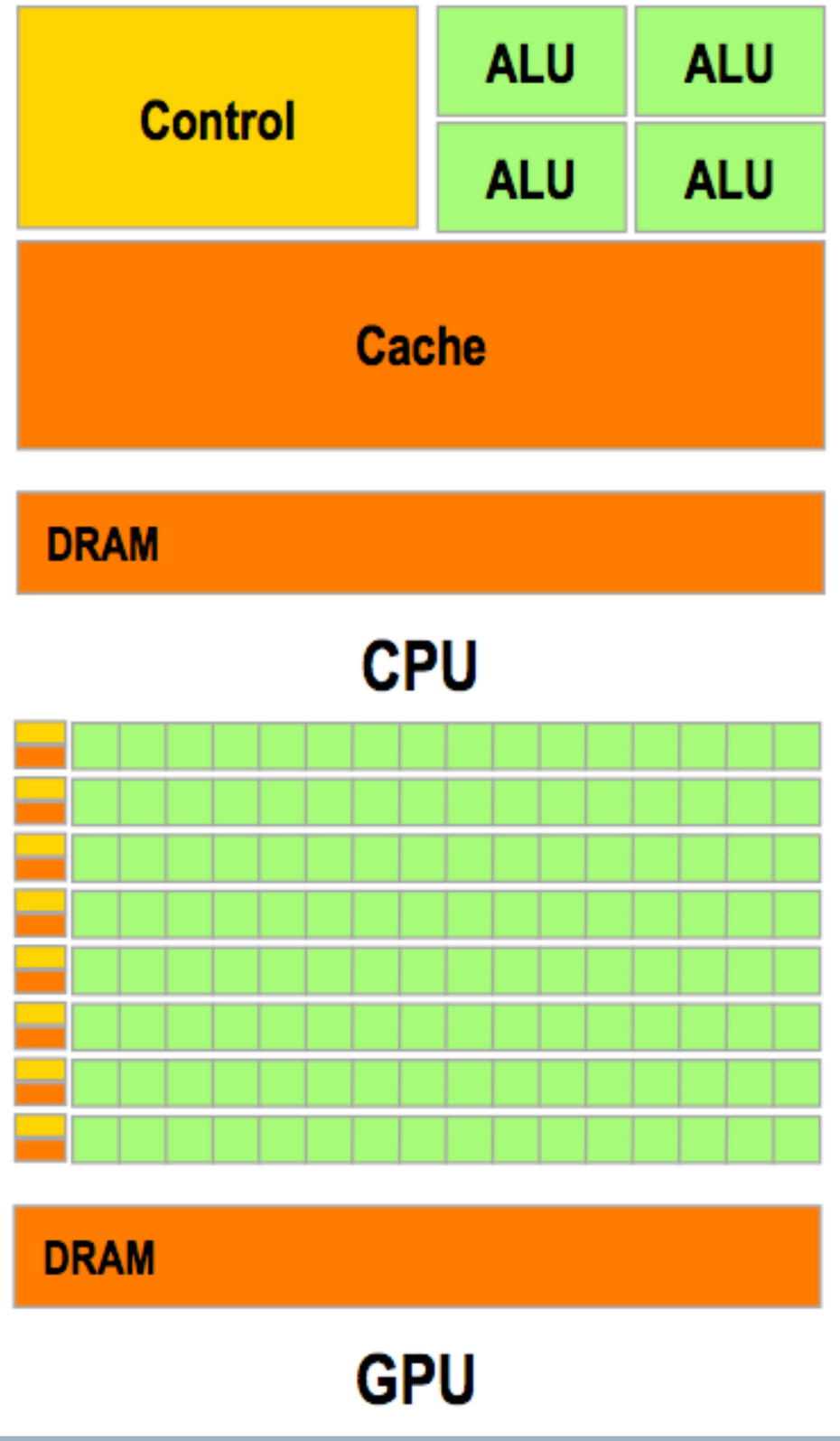
GPUs

- Don't look now, but your graphics card is possibly more powerful than your CPU.



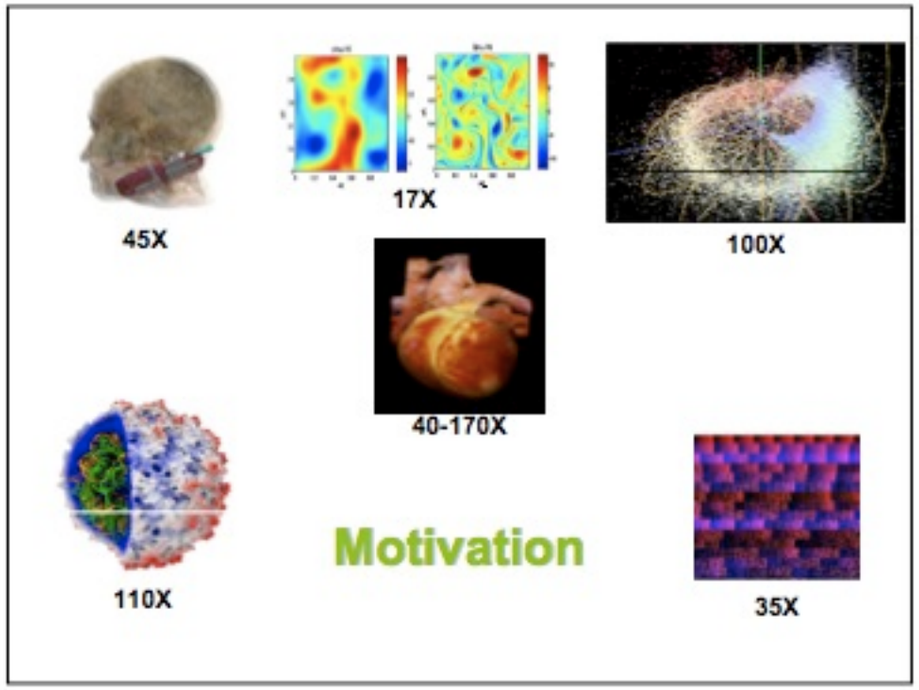
GPU

- The triumphant return of the massively data-parallel type machine (Connection Machine, MasPar..)



Programming GPUs

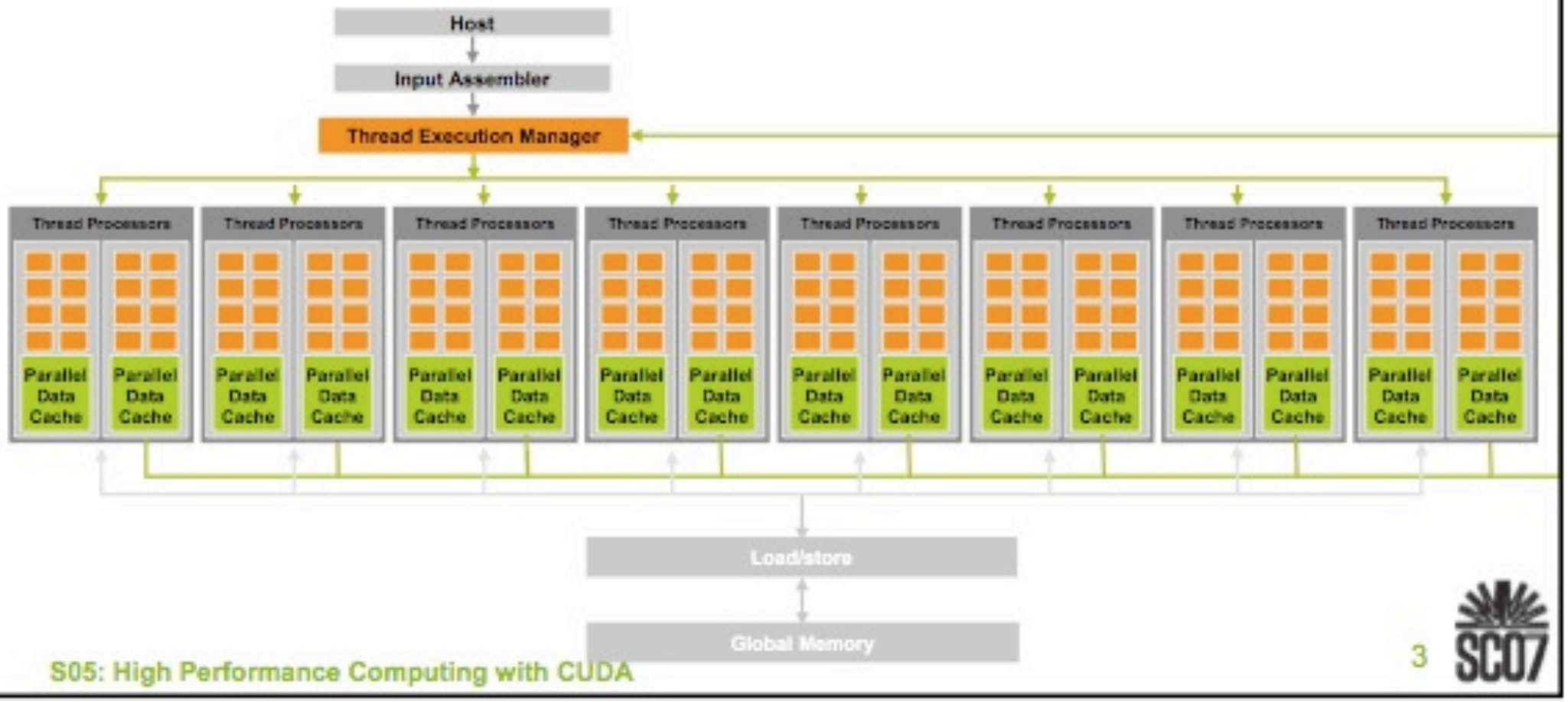
- Used to be pretty bad; put array in as 'textures', have each point in your grid be a vertex that maps the texture...
- Much better now: CUDA (NVidia), OpenCL (NVidia, Apple,AMD; coming soon). BLAS, FFT libraries...



NVidia SC2007 tutorial slides

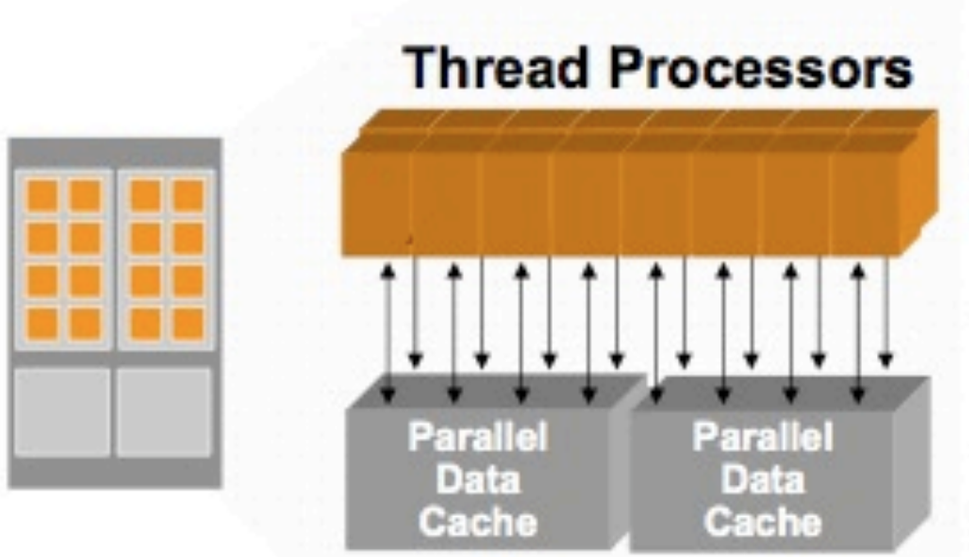
GPU Computing

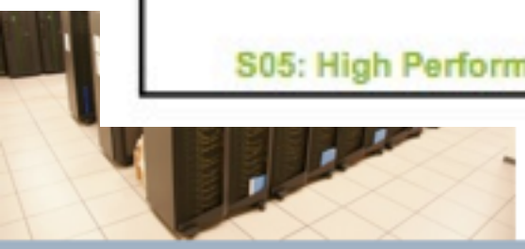
- Processors execute computing threads
- Thread Execution Manager issues threads
- 128 Thread Processors
- Parallel Data Cache accelerates processing



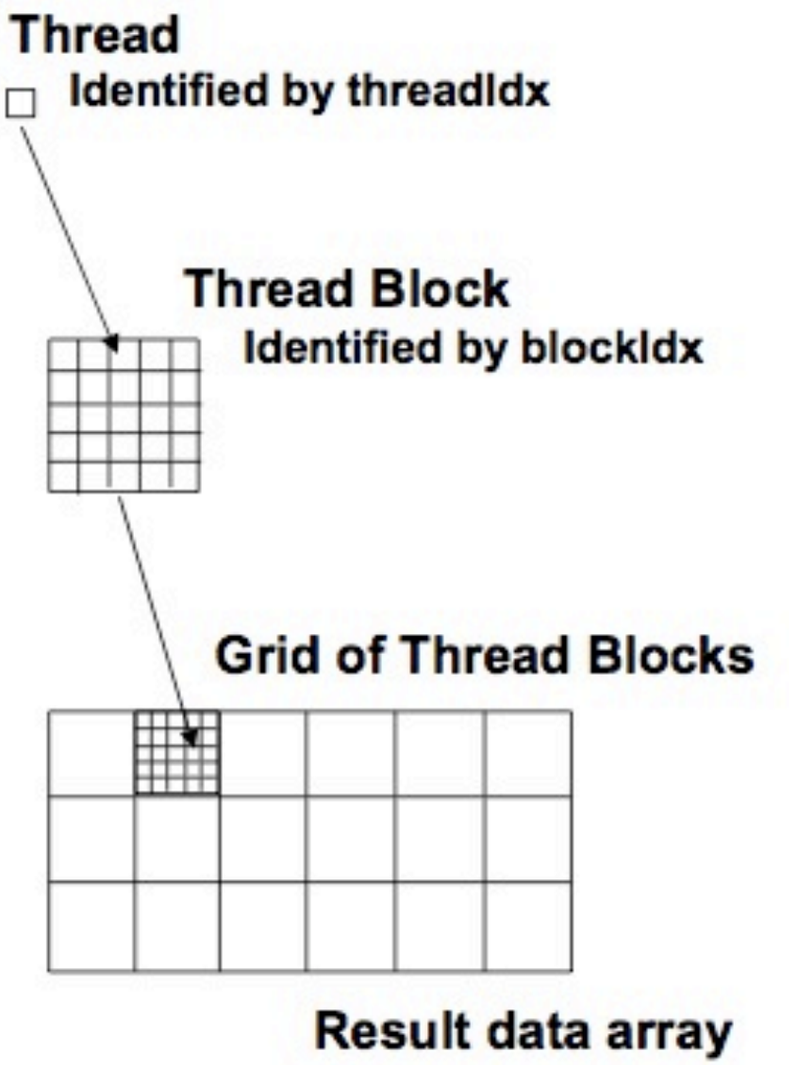
Thread Processor Group

- 128, 1.35 GHz processors
- 16KB Parallel Data Cache per group
- Scalar architecture
- IEEE 754 Precision





Execution Model



Multiple levels of parallelism

- Thread block
  - Up to 512 threads per block
  - Communicate through shared memory
  - Threads guaranteed to be resident
  - `threadIdx, blockIdx`
  - `__syncthreads()`
- Grid of thread blocks
  - `f<<<nblocks, nthreads>>>(a,b,c)`

C-Code Example to Add Arrays

CPU C program

```
void add_matrix_cpu
            (float *a, float *b, float *c, int N)
{
    int i, j, index;
    for (i=0;i<N;i++) {
     for (j=0;j<N;j++) {
        index =i+j*N;
        c[index]=a[index]+b[index];
        }
    }
}
void main()
{
   .....
        add_matrix(a,b,c,N);
}
```

CUDA C program

```
__global__ void add_matrix_gpu
            (float *a, float *b, float *c, int N)
{
    int i=blockIdx.x*blockDim.x+threadIdx.x;
    int j=blockIdx.y*blockDim.y+threadIdx.y;
    int index =i+j*N;
    if( i <N && j <N) c[index]=a[index]+b[index];
}
```

```
void main()
{
    dim3 dimBlock (blocksize,blocksize);
    dim3 dimGrid (N/dimBlock.x,N/dimBlock.y);
    add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
}
```

Books We Like

- Using MPI, 2e; Gropp, Lusk, Skjellum
- Using OpenMP; Kuck, Chapman, Jost, and van der Pas

Parallel Frameworks

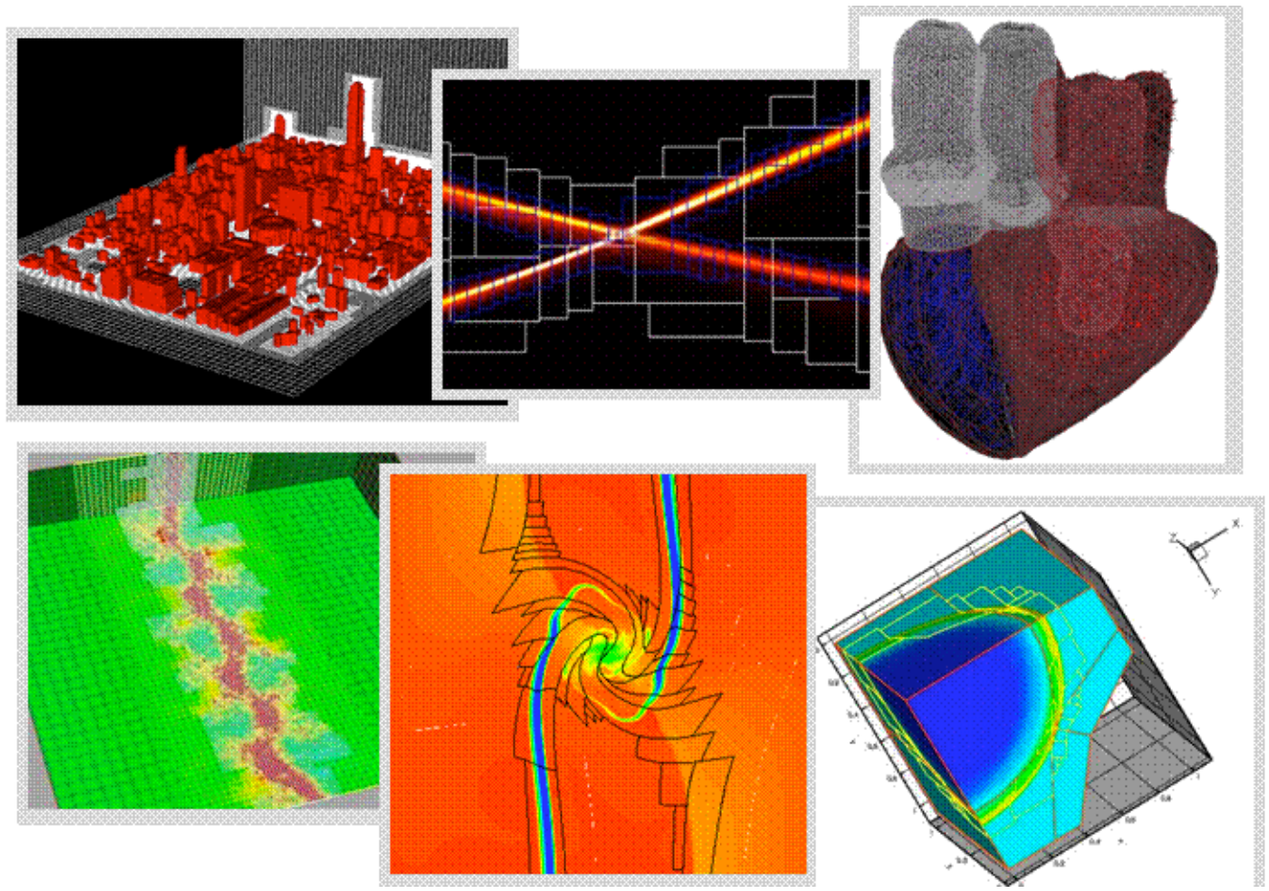
- Parallel infrastructure for codes for scientific computing
- Meshing, AMR, IO, etc.

Chombo

- C++ environment for AMR solution of finite difference equations
- Many bits built in (elliptic solvers, ...)

http://seesar.lbl.gov/ANAG/chombo/

Samrai

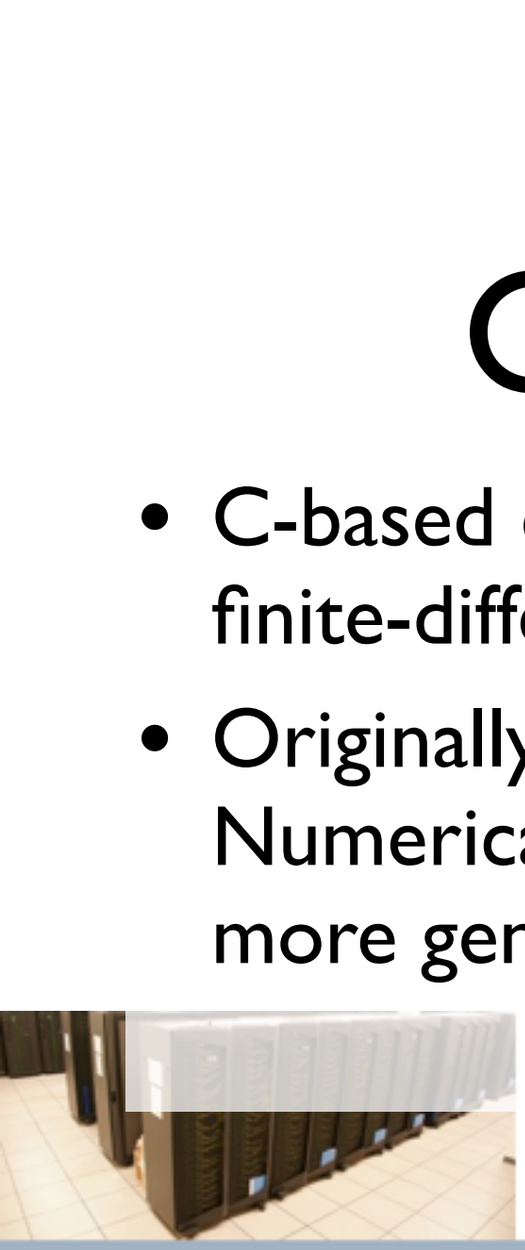


- C++ AMR environment
- Used for many different applications

https://computation.llnl.gov/casc/SAMRAI/

Cactus

- C-based environment for finite-difference equations
- Originally developed for Numerical GR, but somewhat more general purpose



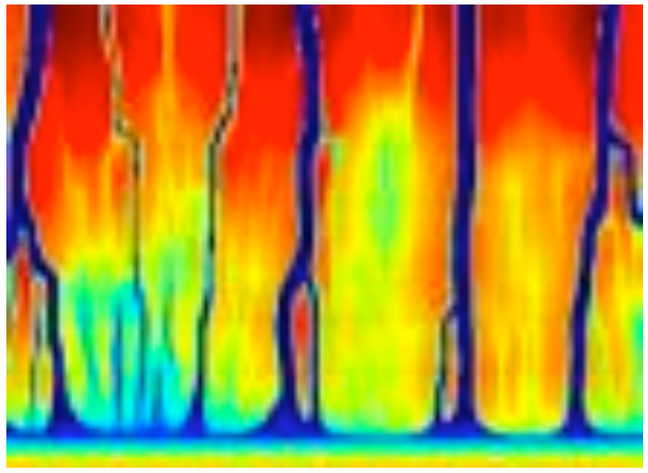
http://www.cactuscode.org

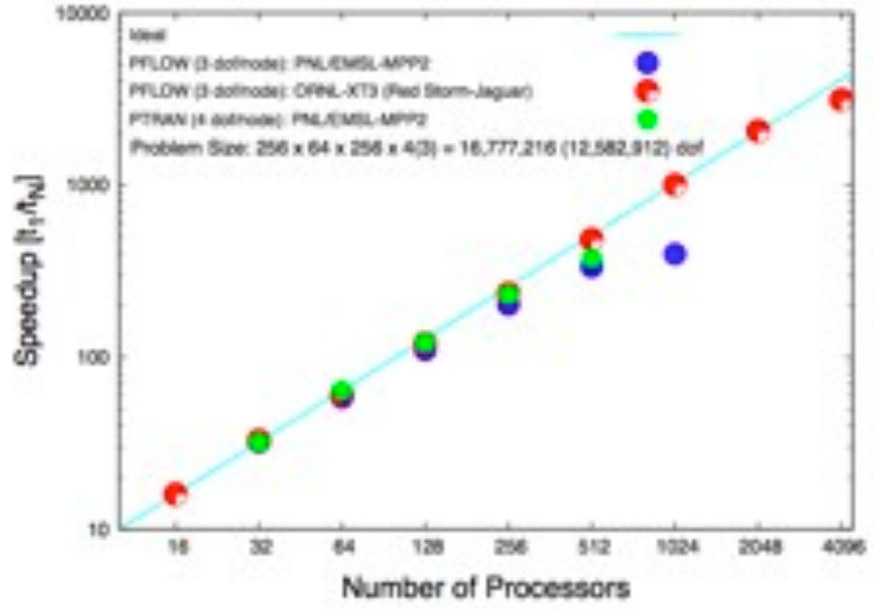
Parallel Libraries

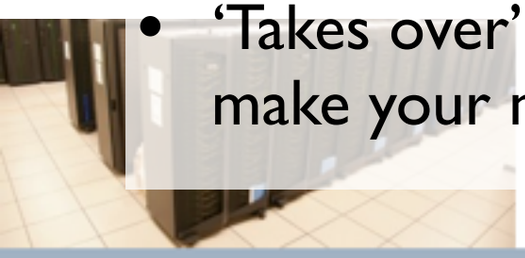
- Libraries for doing common computations in parallel

PETSc

- Parallel vectors, matrices on top of MPI
- Parallel preconditioners, Krylov subspace methods, Newton-type nonlinear solvers, ODE solvers
- Easy to try lots of different solvers with a given code
- Developed/used by US DOE labs; won't go away anytime soon, actively developed
- 'Takes over' your application -- have to make your matrices PETSc matricies







http://www-unix.mcs.anl.gov/petsc/petsc-2/

See also http://www.netlib.org/scalapack/, etc

Other Linear-Algebraish things

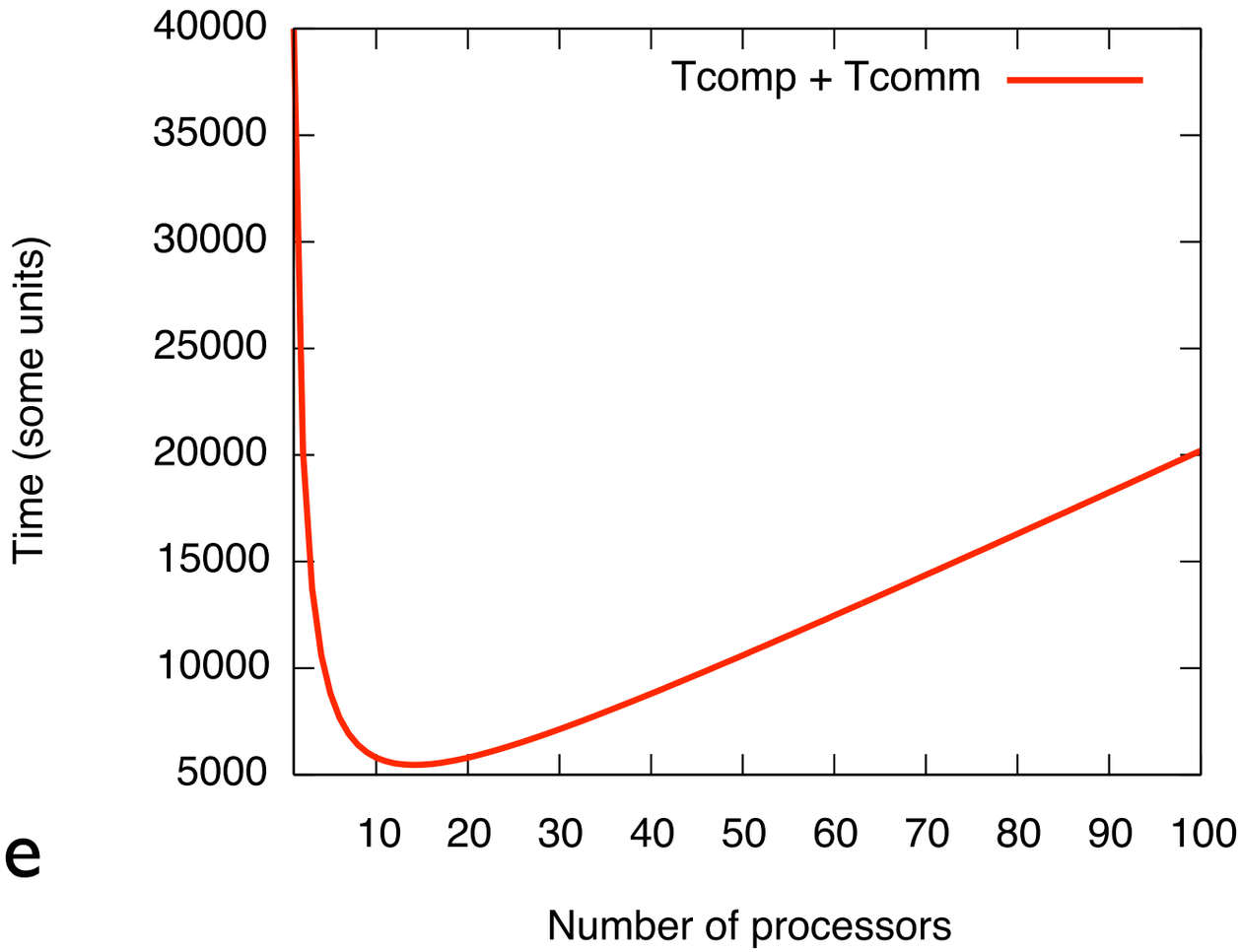
- Trillinos/Epetra; subset of blas/lapack http://trilinos.sandia.gov/packages/
- Aztec; iterative solver for sparse systems http://www.cs.sandia.gov/CRF/aztec1.html

FFTW

- Fastest Fourier Transfor in the West
- MPI **or** OpenMP
- http://www.fftw.org

Performance Tools

- Parallel Programs are *complicated*
- When something gives worse than expected performance, how do you know what to fix?



Measuring Performance

- You can't improve what you can't measure
- You can't even tell there's a problem if you're not measuring.

http://commons.wikimedia.org/wiki/File:Plastic_tape_measure.jpg

First line of defence:

- `/bin/time` (or `/usr/bin/time`)
- by-section timing (eg, like `tick(), tock()`)

```
$ /usr/bin/time -p ls
Makefile          pca_mpi_utils.c
pca_mpi_utils.h pca_utils.c
pca_utils.h
real              0.00
user              0.00
sys               0.00
```

```
$ ./nbody --nsteps=10 --nooutput
mydata has 1500 particles in a 3-
dimensional space.
Simulation is 1
Tock registers      6.5011e-02s
Tock registers      7.2196e-01s
```

gprof

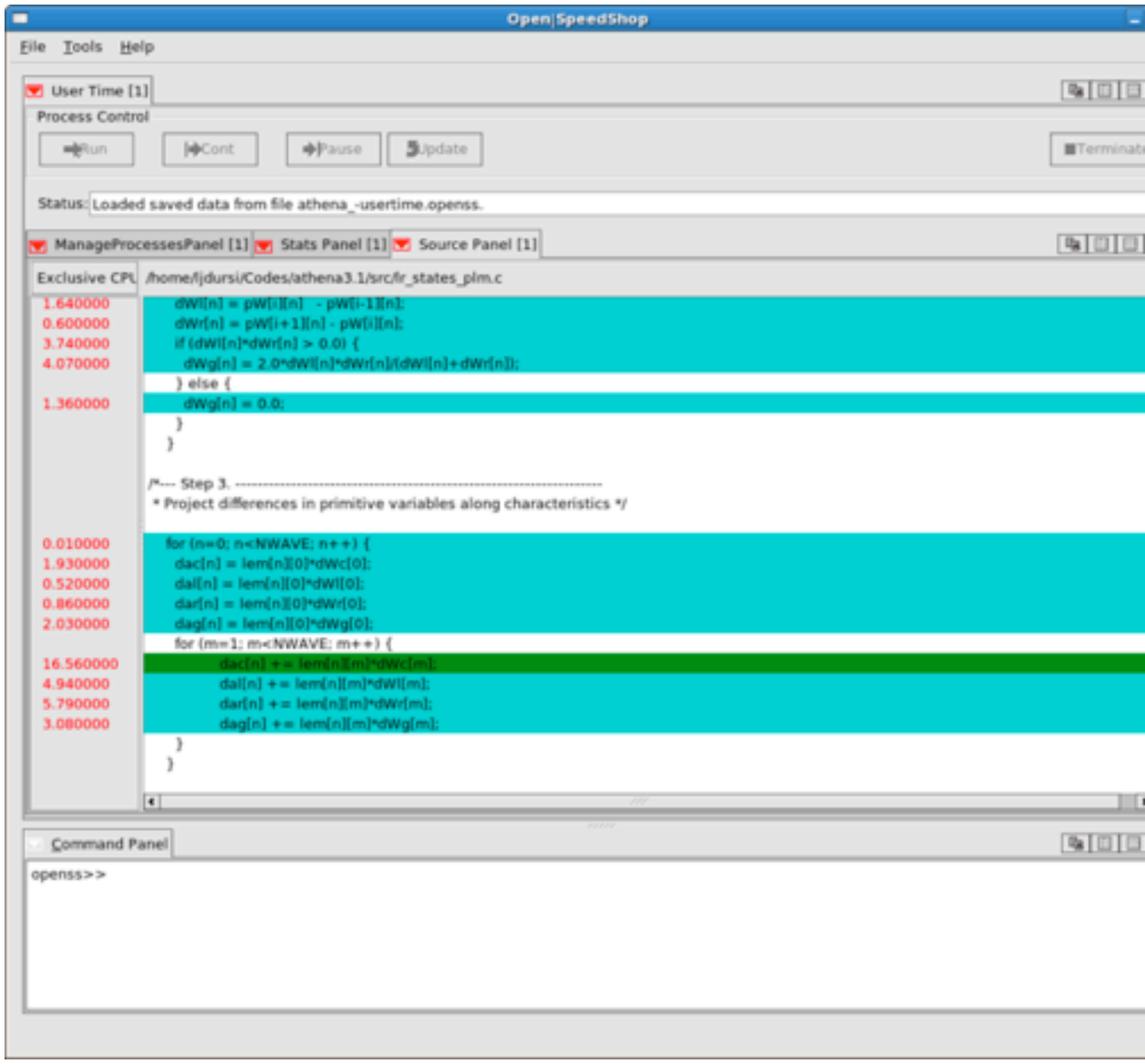
- Talked about by J. Sievers
- Works almost anywhere
- Sampling (as vs. tracing)
- With long enough run, can give meaningful stats line-by-line
- installed on TCS, GPC

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double calc_pi(long n) {
    long in = 0;
    long out = 0;
    long i;
    double x,y;

    for (i=0; i<n; i++) {
        x = drand48();
        y = drand48();
        if (x*x+y*y < 1) {
            in++;
        } else {
            out++;
        }
    }

    return 4.*(double)in/(double)(in+out);
}

int main(int argc, char **argv) {
    long n, defaultn=100000;
    double pi;
    time_t t;

    /* seed random number generator */
    srand48(time(&t));

    /* get number of tries */
    if (argc < 2 || (n=atoi(argv[1]))<1) {
        n = defaultn;
        printf("Using default n = %ld\n", n);
    }

    pi = calc_pi(n);
    prin
    retu
}
```

```
$ gcc -g -pg -o pi pi.c
$ ./pi
[....]
$ gprof --line pi gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 70.31      0.70     0.70                             calc_pi (pi.c:14 @ 40078b)
 14.27      0.84     0.14                             calc_pi (pi.c:17 @ 4007bc)
  5.10      0.89     0.05                             calc_pi (pi.c:11 @ 4007c1)
  4.08      0.93     0.04                             calc_pi (pi.c:15 @ 4007b5)
  3.06      0.96     0.03                             calc_pi (pi.c:13 @ 400781)
  2.55      0.98     0.03                             calc_pi (pi.c:12 @ 400777)
  1.53      1.00     0.02                             calc_pi (pi.c:11 @ 40076d)
  0.00      1.00     0.00        1     0.00     0.00  calc_pi (pi.c:5 @ 40074c)
```

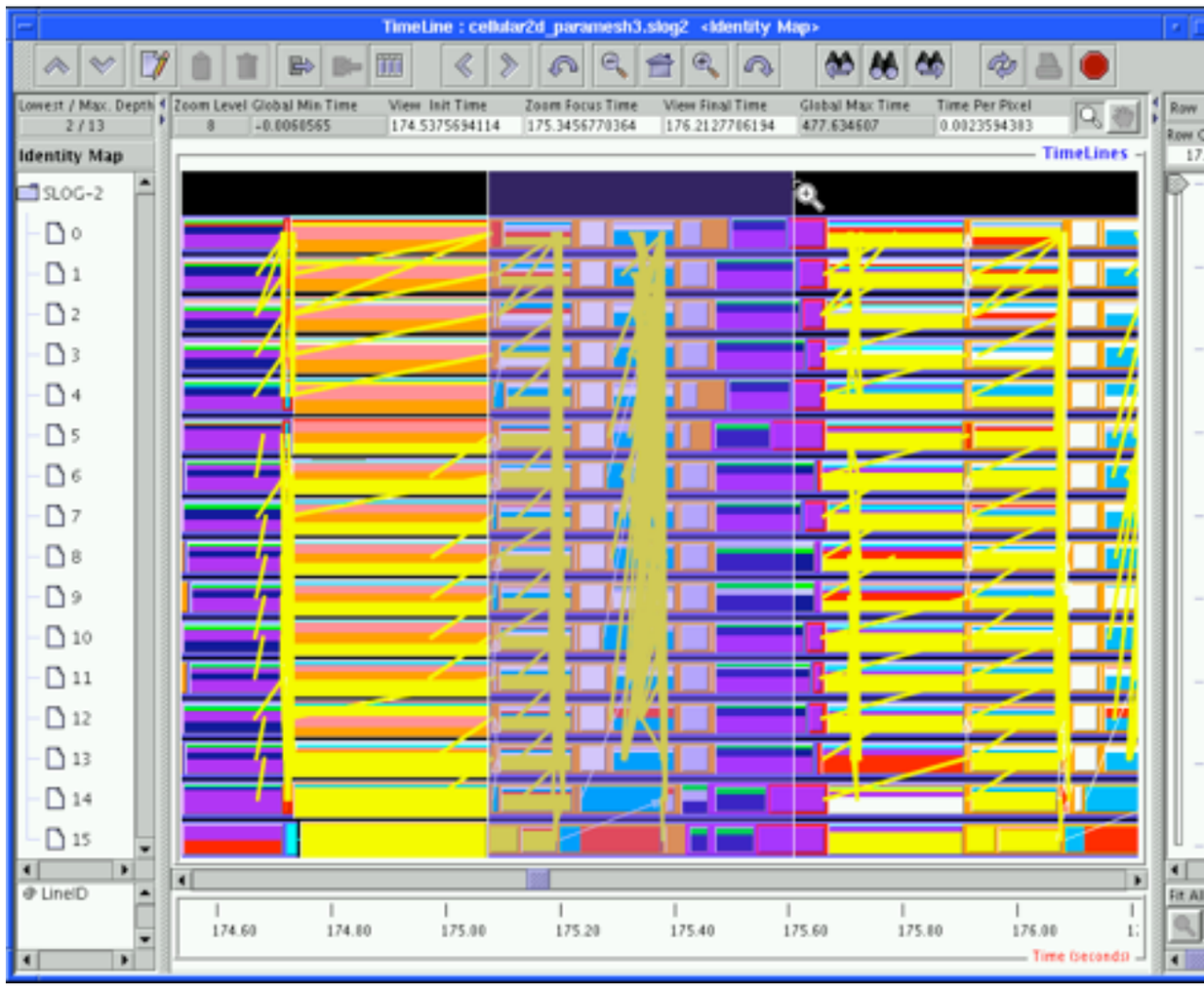
OpenSpeedShop

- http://www.openspeedshop.org
- Linux
- 'experiments'
- Saves data, makes it very easy to compare results as you're optimizing.
- Installed on GPC (module load openspeedshop)
  - `$ openss –f ./icos pcsamp`

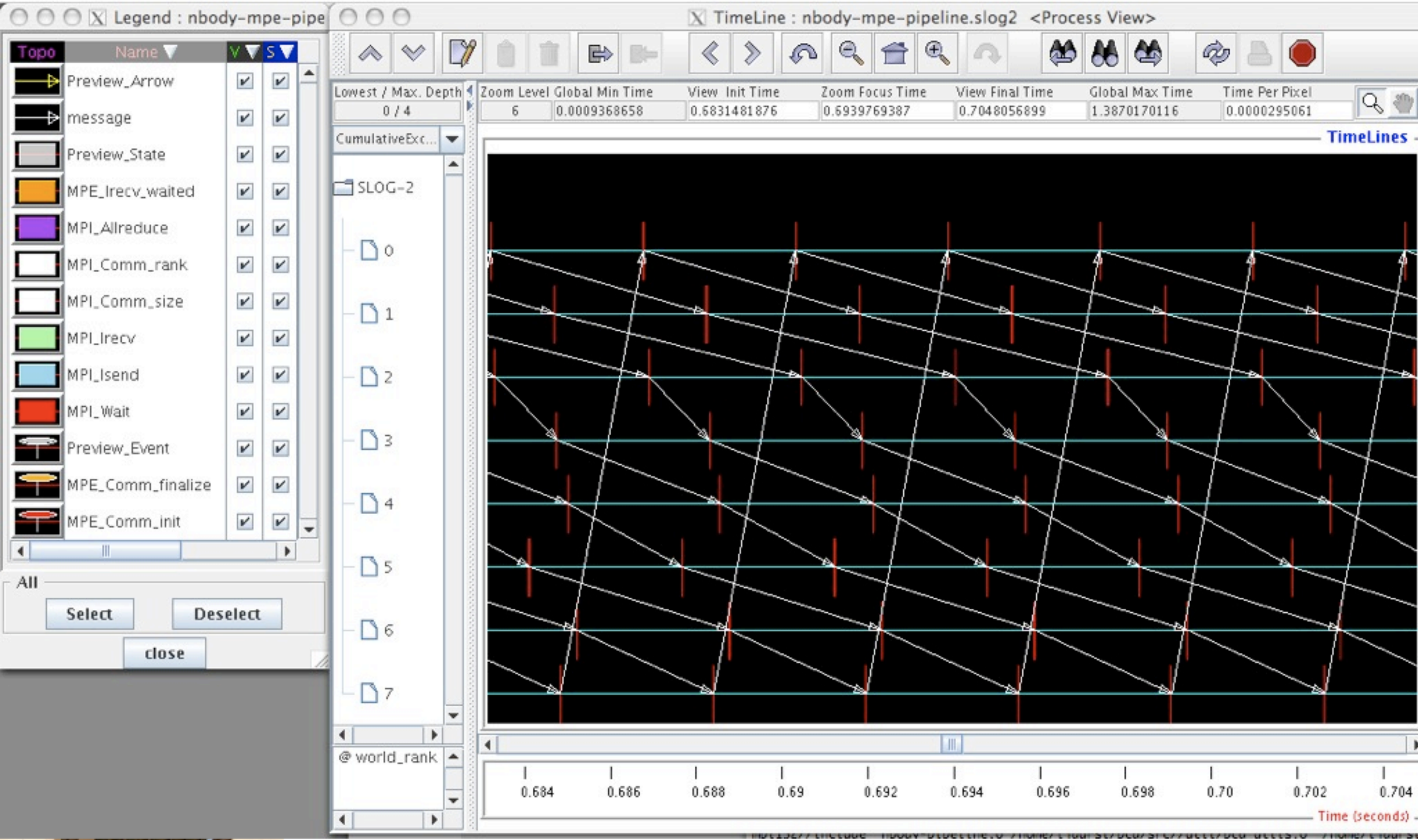


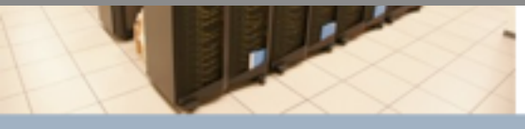
Jumpshot/MPE

- MPI has built in support for logging, so there are lots of MPI tools
- Logging big programs not easy!
- Comes with some installs of mpich.
- Installed on TCS, soon to be on GPC



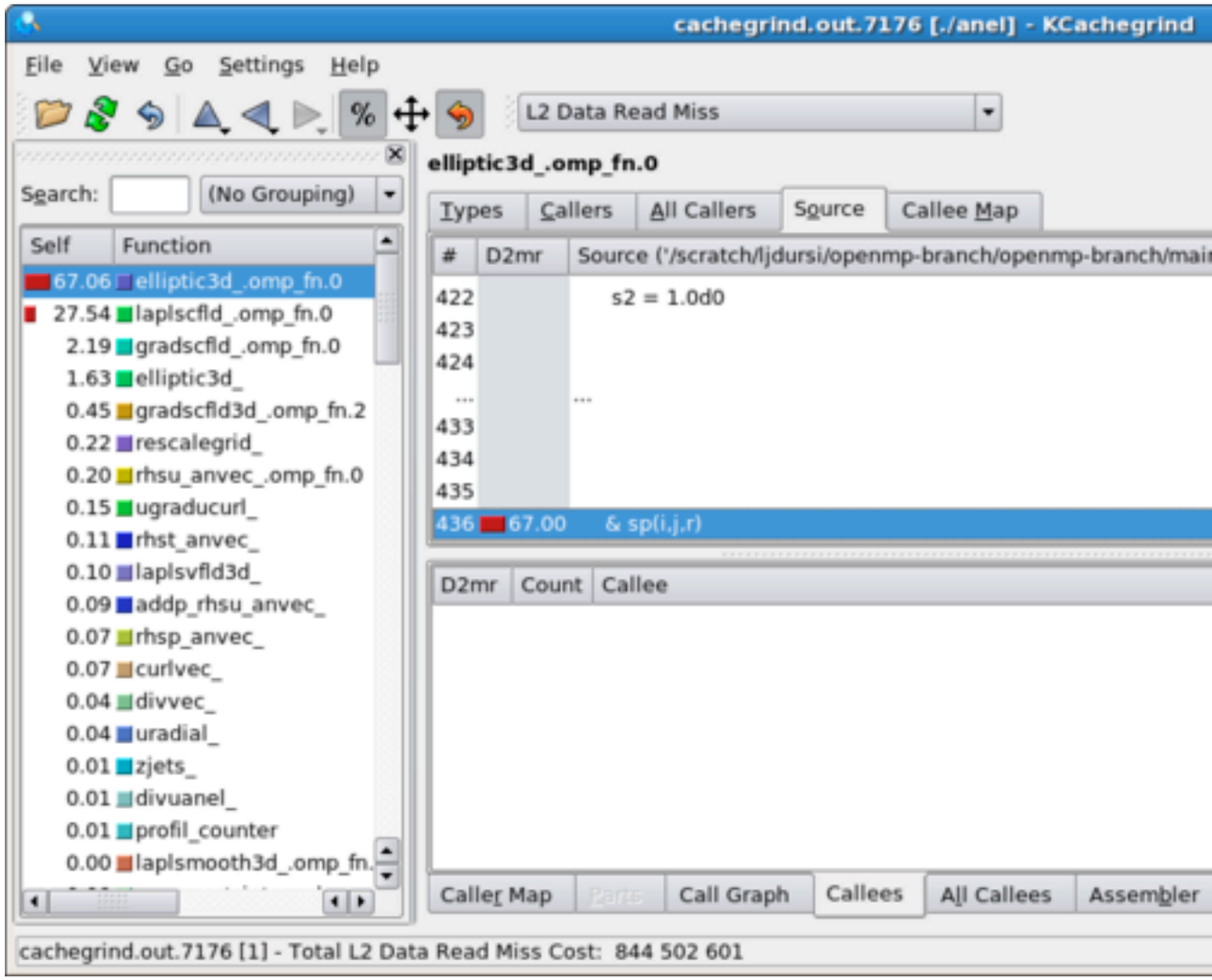
http://www-unix.mcs.anl.gov/perfvis/


Legend : nbody-mpe-pipe
Topo
Name
V
S
Preview_Arrow
message
Preview_State
MPE_Irecv_waited
MPI_Allreduce
MPI_Comm_rank
MPI_Comm_size
MPI_Irecv
MPI_Isend
MPI_Wait
Preview_Event
MPE_Comm_finalize
MPE_Comm_init
All
Select
Deselect
close
TimeLine : nbody-mpe-pipeline.slog2 <Process View>
Lowest / Max. Depth
0 / 4
Zoom Level
6
Global Min Time
0.0009368658
View Init Time
0.6831481876
Zoom Focus Time
0.6939769387
View Final Time
0.7048056899
Global Max Time
1.3870170116
Time Per Pixel
0.0000295061
CumulativeExc...
SLOG-2
0
1
2
3
4
5
6
7
@ world_rank
TimeLines
0.684
0.686
0.688
0.69
0.692
0.694
0.696
0.698
0.70
0.702
0.704
Time (seconds)



cachegrind (valgrind)

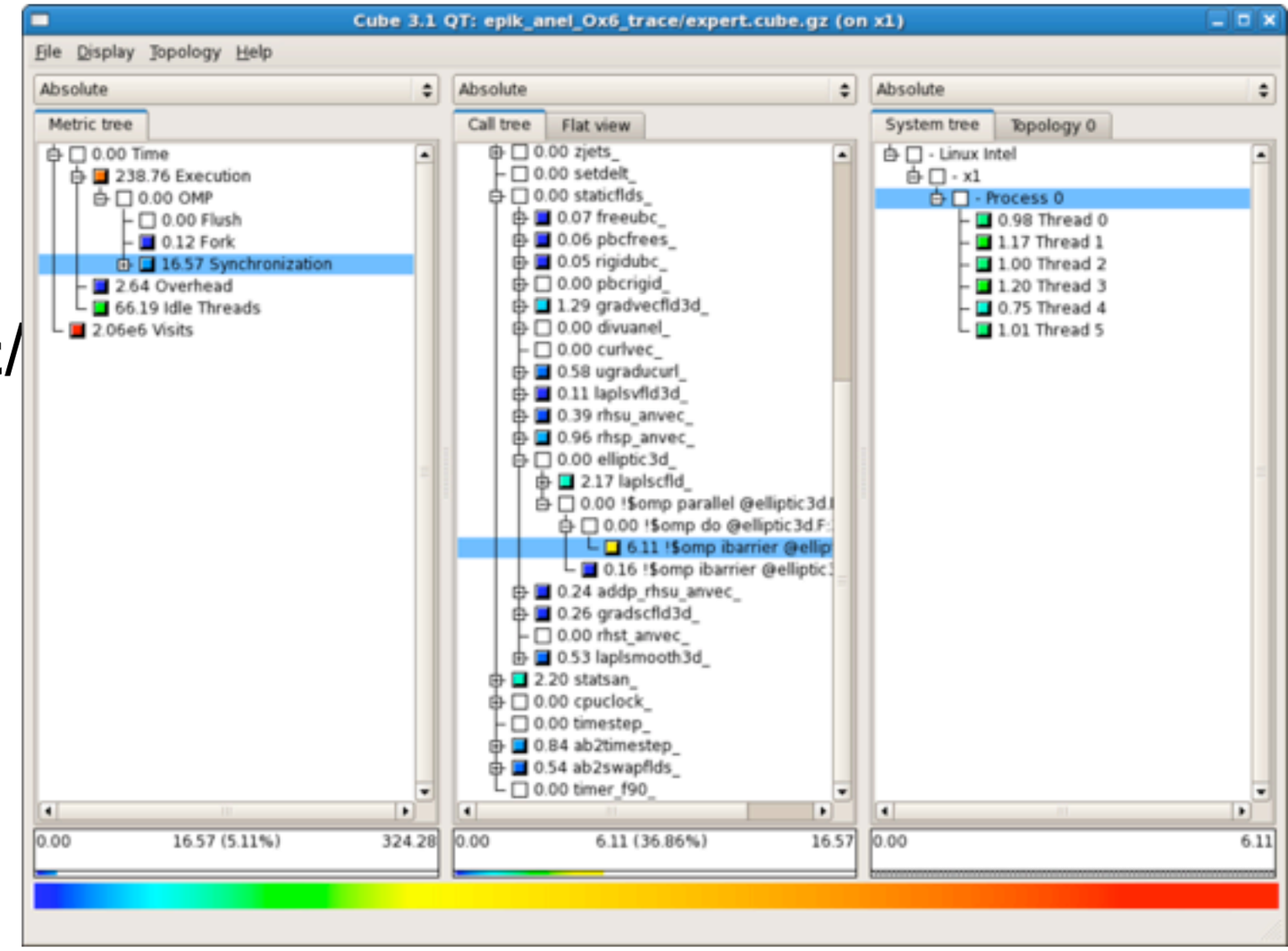
- Part of valgrind package
- Emulator
- Exquisitely detailed description of cache performance
- Linux tool; installed on GPC




```
$ valgrind --tool=cachegrind
myprogram myprogram_arguments

$ kcachegrind ./cachegrind.out
```

Scalasca

- http://www.fz-juelich.de/jsc/scalasca/
- Auto-instrument code
- Then run
- Sampling + tracing
- OpenMP + MPI
- Knows parallel issues
- Installed on TCS, GPC (module load scalasca)



Us!

- With specific questions, email support@scinet

- For longer discussions, make an appointment and come talk with us