From Peak to Peak: Maximizing Performance on PSC's Cray XT3

Deborah Weisser

Nick Nystrom, Shawn Brown, Roberto Gomez, Junwoo Lim, David O'Neal, John Urbanic, R. Reddy, Yang Wang, Chad Vizino, Joel Welling

Survey of performance optimizations at PSC

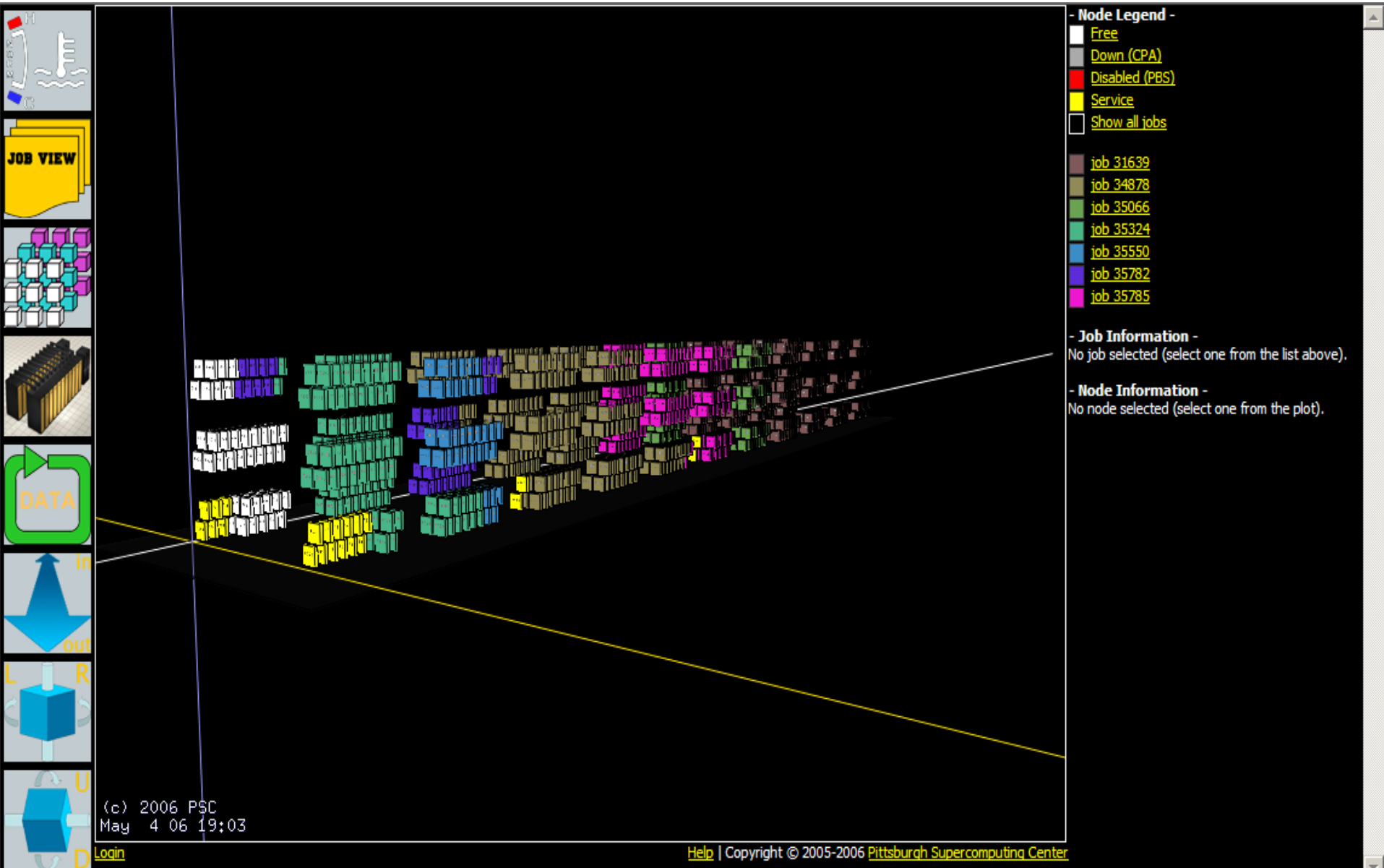
- Optimizing job placement
- Generating efficient C code: a PGCC example
- Craypat analysis of LEO
- PSCC
- NAMD

Optimizing Job Placement

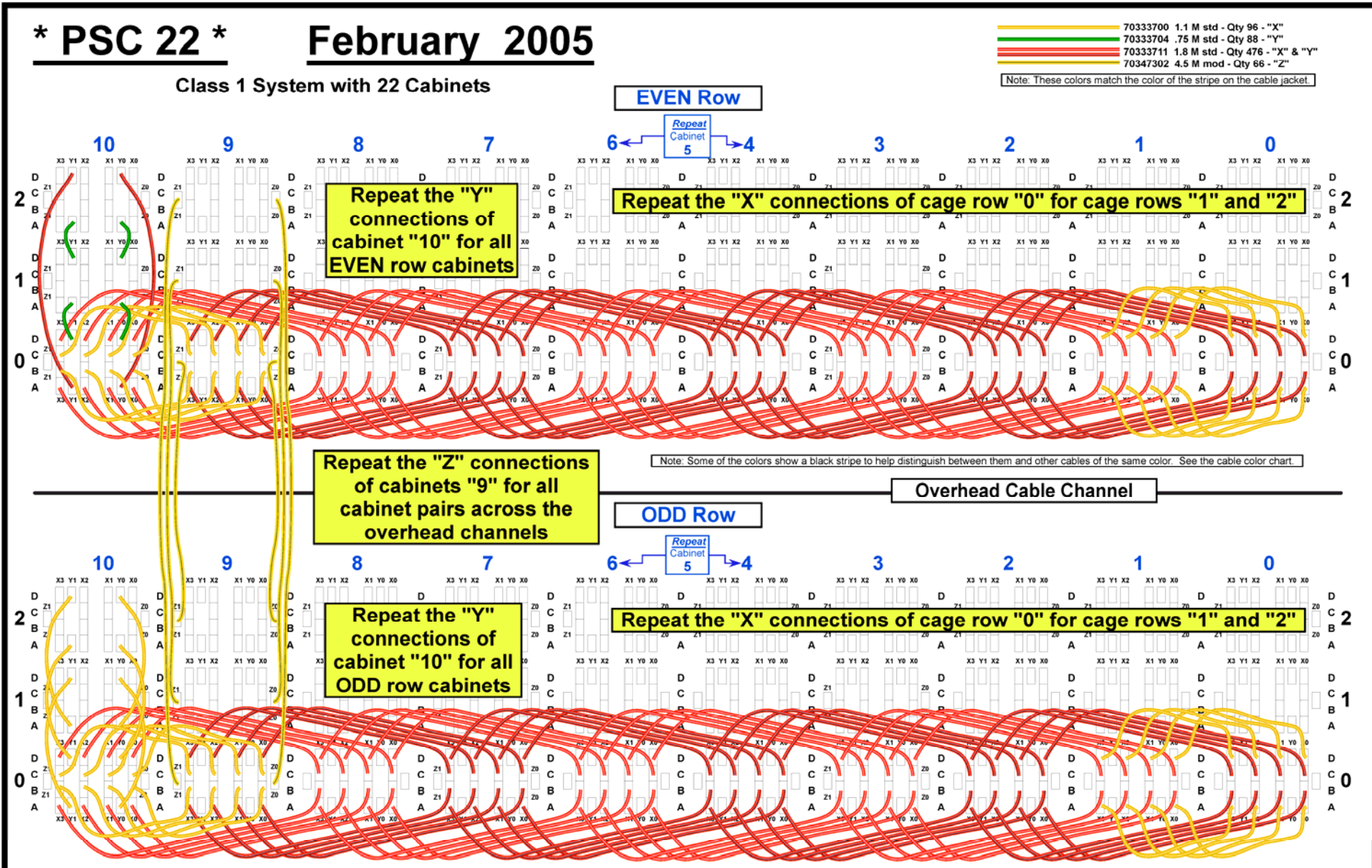
- Motivation: the SeaStar's 3D mesh interconnect (configured at PSC as a 3D torus) offers exceptional bandwidth and low latency
  - *however, improperly scheduled jobs produce contention that undermines otherwise scalable applications*
  - empirical observations indicated that production jobs were being badly fragmented, seriously degrading their performance
  - recently, we have quantified the impact of noncontiguous partitions on a variety of communication-intensive applications
- Joint work with Chad Vizino to build support for job layout into XT3 scheduling

PITTSBURGH SUPERCOMPUTING CENTER

Job Placement by Cabinet



Bigben Wiring Diagram



Default Scheduling Algorithm

- Nodes allocated numerically from low to high from among free nodes:

cabinet 0 – row 0
cabinet 0 – row 1
cabinet 1 – row 0
cabinet 1 – row 1
…
cabinet 10 – row 0
cabinet 10 – row 1

Job placement

Nodes allocated numerically from low to high from among free nodes, which results the following cabinet order:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 0

…but cabinets are connected differently:

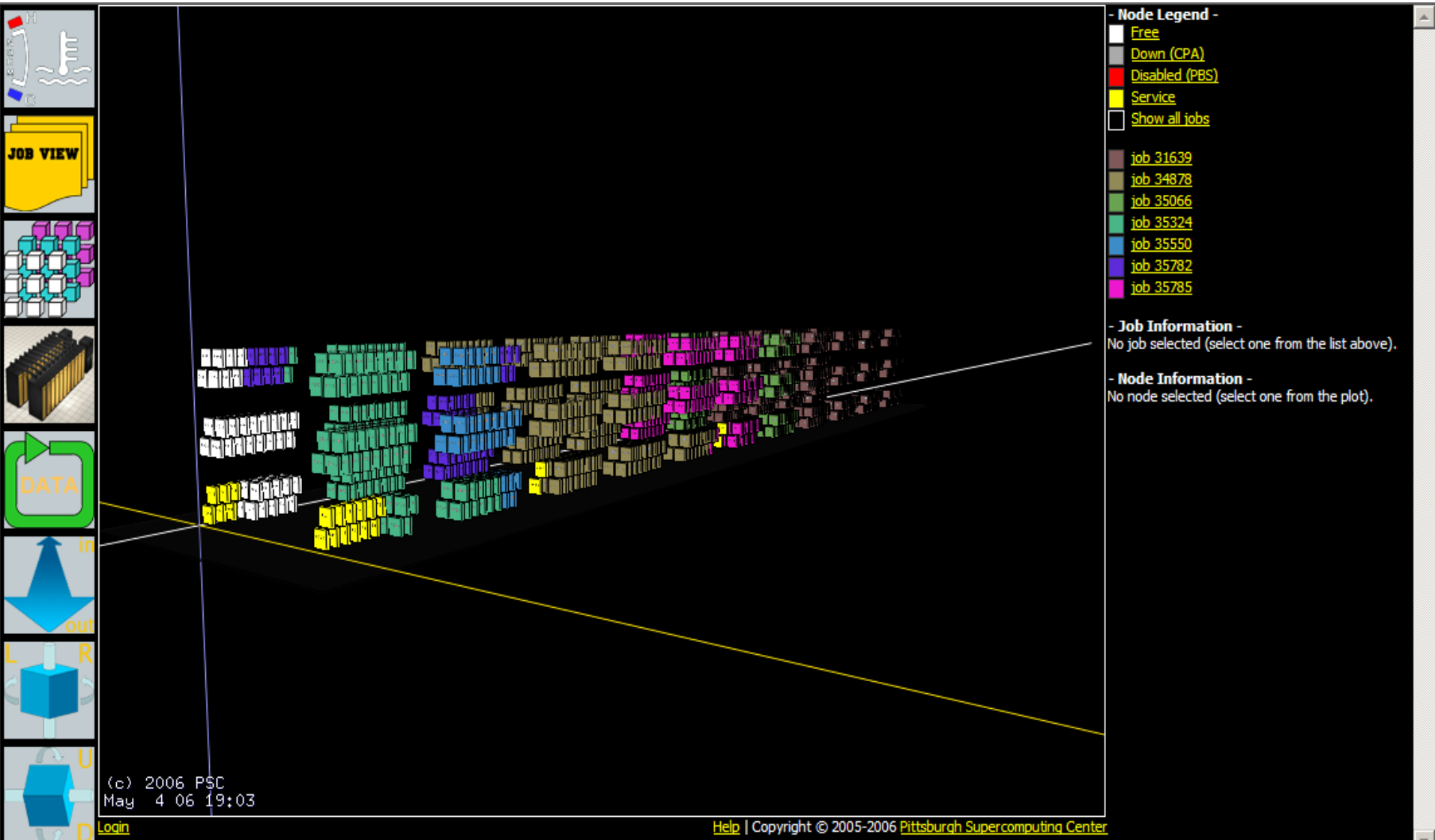
0 – 2 – 4 – 6 – 8 – 10 – 9 – 7 – 5 – 3 – 1 – 0

The number of hops between numerically adjacent cabinets varies from 1 to 5

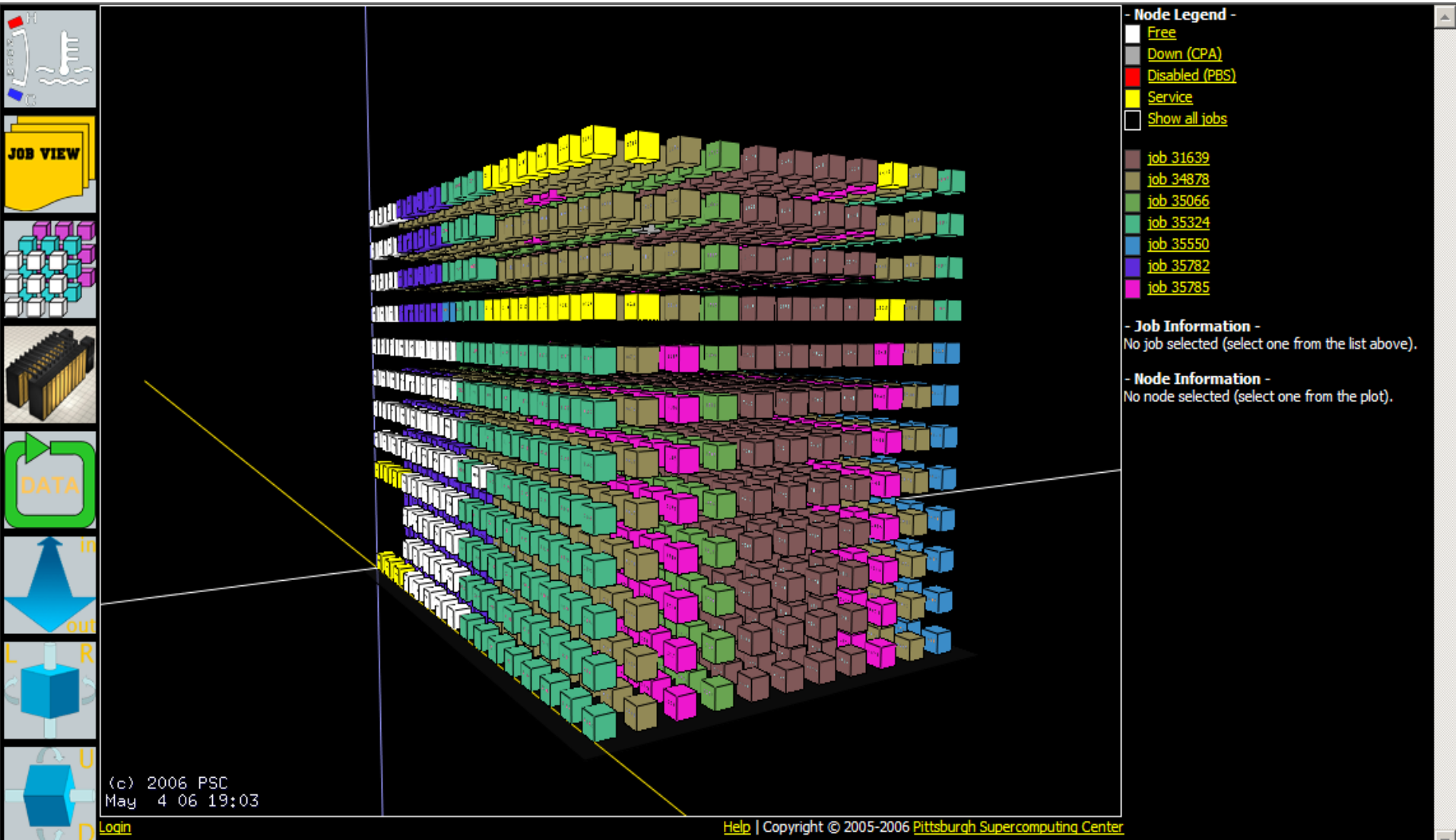
Hops between cabinets

	0	1	2	3	4	5	6	7	8	9	10
0											
1	1										
2	1	2									
3	2	1	3								
4	2	3	1	4							
5	3	2	4	1	5						
6	3	4	2	5	1	5					
7	4	3	5	2	5	1	4				
8	4	5	3	5	2	4	1	3			
9	5	4	5	3	4	2	3	1	2		
10	5	5	4	4	3	3	2	2	1	1	

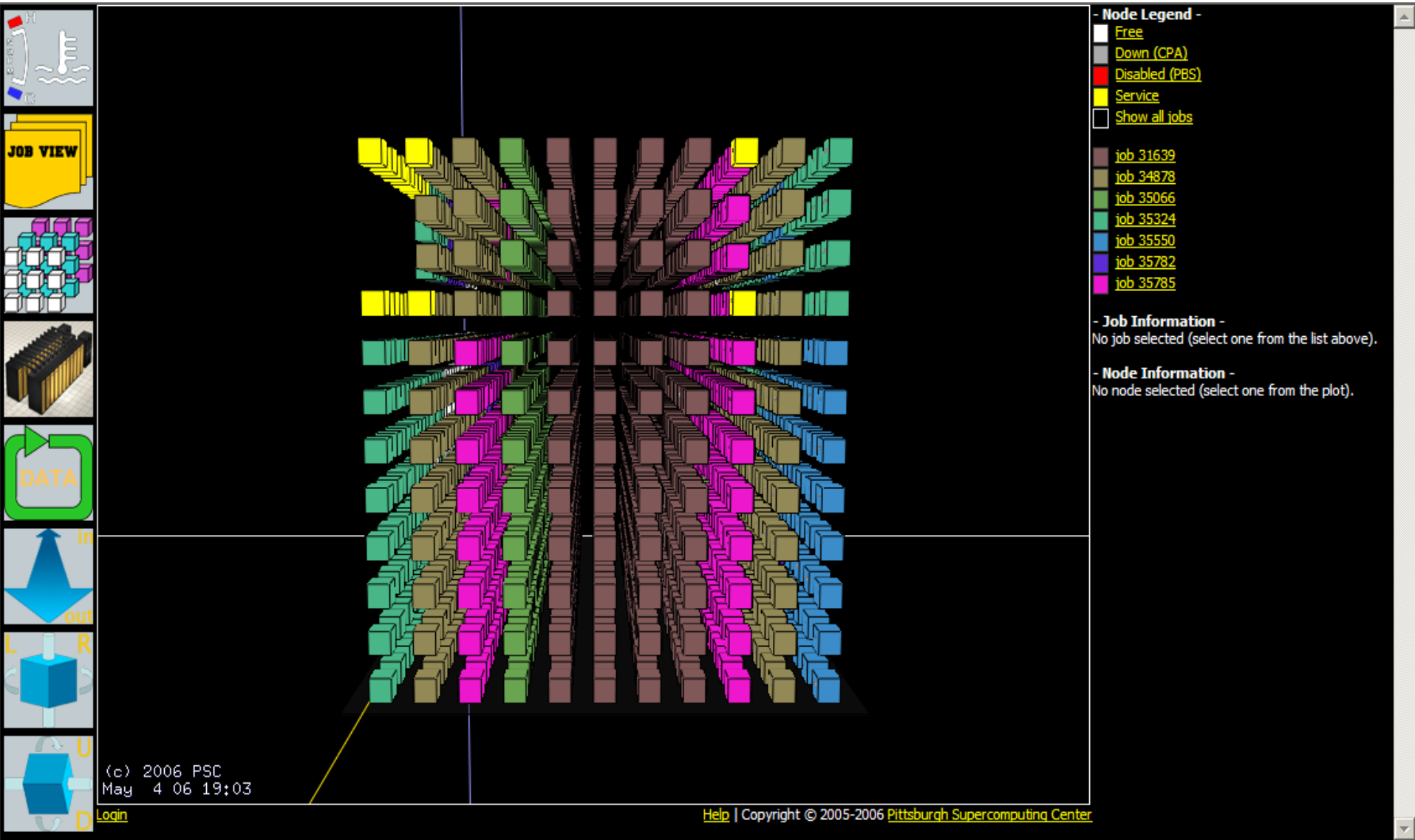
Job placement by cabinet



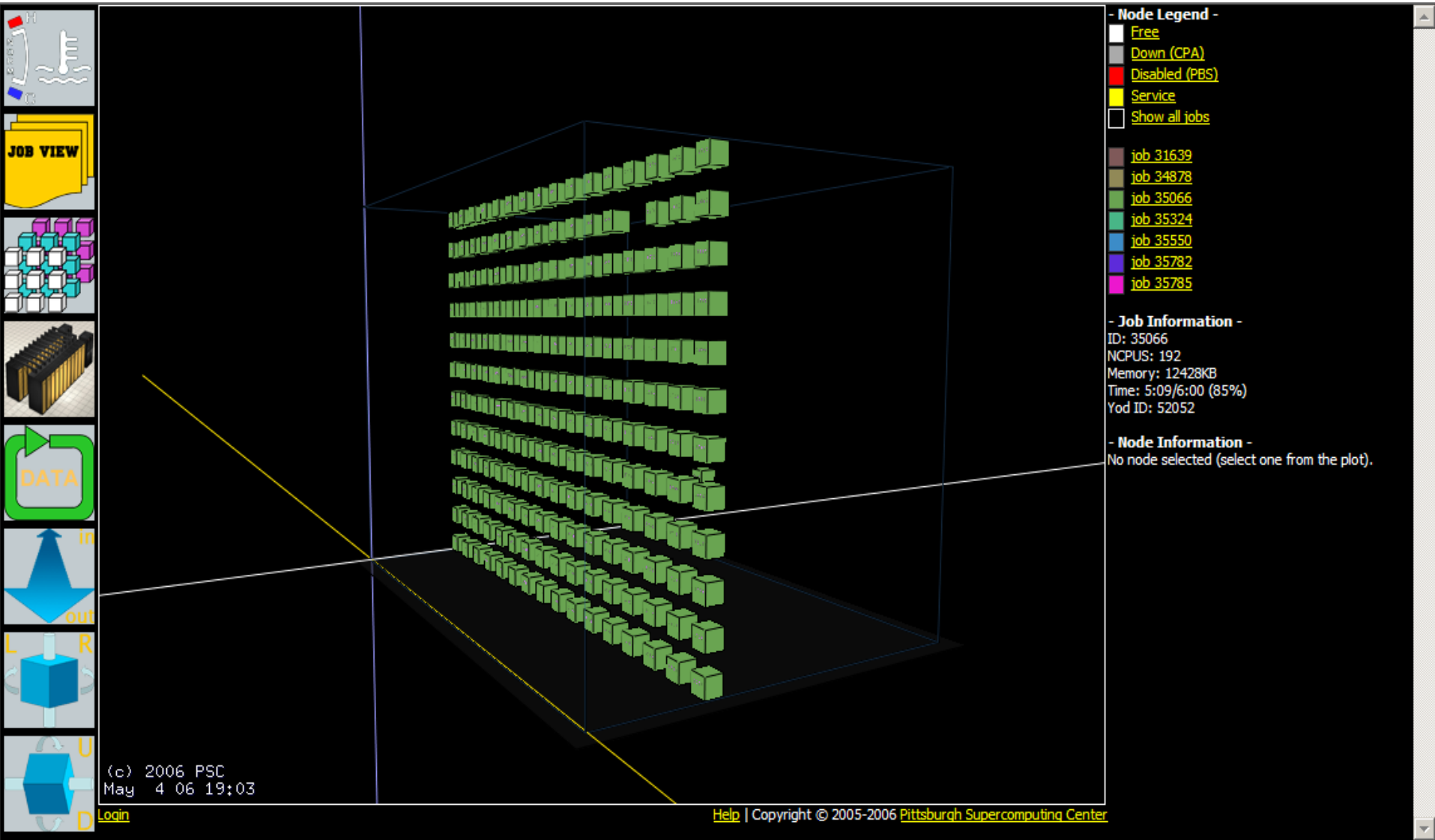
Job placement by node adjacency



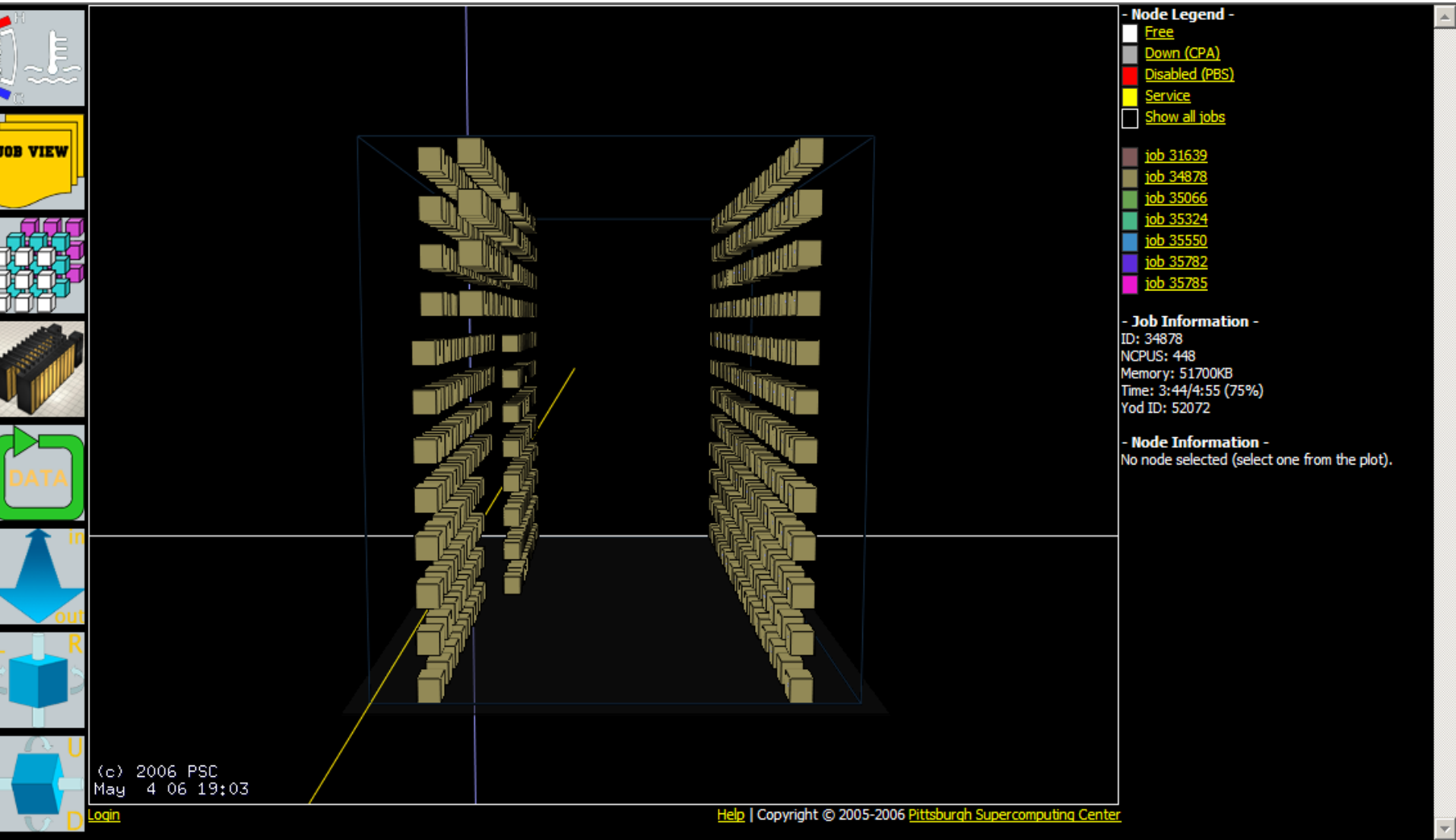
Job placement by node adjacency



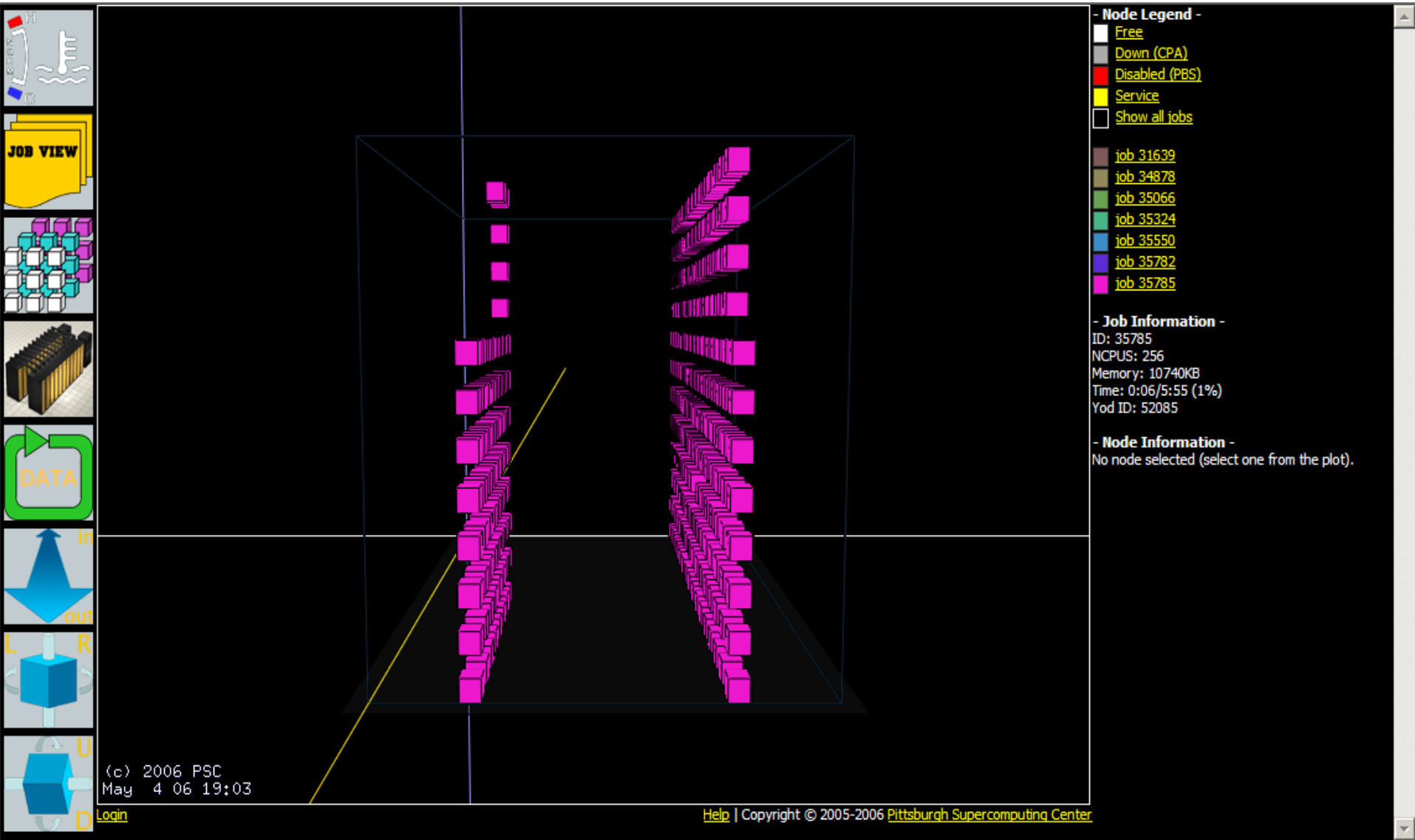
Job placement by node adjacency



Job placement by node adjacency



Job placement by node adjacency



Results

- Broad application mix reflects different types of communication-intensive jobs
- Non-idealized experiments: 1024p PTRANS job running concurrently

Application	p	placed: default scheduling algorithm	placed: optimized for adjacency	production average	production standard deviation	improvement optimized vs. production
PTRANS	1024	129.3 GB/s	146.5 GB/s	131.2 GB/s	21.2 GB/s	11.7%
DNS	512	316.5 s	296.0 s	310.5 s	9.2 s	4.7%
DNS	192	198.0 s	163.0 s	181.7 s	23.5 s	10.3%
NAMD	512	161.4 s	150.1 s	167.1 s	13.1 s	9.8%
NAMD	32	252.7 s	228.3 s	252.0 s	12.2 s	9.4%

Integration with job scheduler

- Default assignment of jobs to nodes is in CPA only, not the scheduler, which just keeps track of free node counts
- The default CPA algorithm allocates free nodes numerically from low to high
- PSC (Chad Vizino) developed customized infrastructure to dynamically support different scheduling algorithms. Decision of which nodes a job runs on is made by the scheduler, not the CPA
    - MOM obtains and parses a list of nodes from the scheduler
    - new CPA data structure added to contain list from MOM

Ongoing work

- Investigation and implementation of job placement algorithms
  - optimized for particular applications: specify optimal dimensions
    - requirement (performance necessary, e.g. real-time steering)
    - preference (e.g. 8 x 8 x 8 preferred if available)
    - default (schedule from available nodes, but in an optimized fashion)
  - minimize external job traffic
  - minimize sending messages through nodes belonging to other jobs
- Use of profiling tools to automatically generate good job dimensions based on communication patterns
  - Craypat
  - Detailed analysis of SeaStar performance registers and routing algorithms
  - PSC 3D Monitor to detect and ameliorate high-traffic regions in the interconnect

Generating efficient code with C compilers: an example with PGCC

- C language semantics limit compiler optimizations
  - *unless the application provides additional information*
  - (the fundamental difference between performance of optimized C and Fortran code, limiting the performance of many C, C++, and UPC applications)
  - e.g. MILC with key routines written in assembly is up to 17% faster than pure C

- Joel Welling (PSC); Doug Miles and Dave Norton (PGI)

A tale of two instructions

- Multiply, add
  - the Opteron supports x87 and SSE floating-point instructions
  - SSE instructions can be “packed” (2 instructions per cycle, “vectorized”) or unpacked
  - generation of packed instructions requires data known to be non-overlapping in cache
- Packed instructions: mulpd, addpd
  - 2 floating point operations per cycle
- Unpacked instructions: mulsd, addsd
  - 1 floating point operation per cycle

Example: Matrix Multiplication

```
#define RANK 64
void matmul( double newMatrix[RANK*RANK],
             const double M1[RANK*RANK],
             const double M2[RANK*RANK] ) {
  int i,j,row,column;
  for (row=0; row<RANK; row++) {
    for (column=0; column<RANK; column++) {
      newMatrix[(RANK*row)+column] = 0.0;
    }
    for (i=0;i<RANK;i++) {
      for (j=0; j<RANK; j++) {
        newMatrix[(RANK*row)+j] += M1[(RANK*row)+i]*M2[(RANK*i)+j];
      }
    }
  }
}
```

Beginning: pgcc -S -fastsse

```
# lineno: 10
        movlpd  -8(%rax),%xmm0
        decl    %edx
        mulsd   (%rsi),%xmm0
        addsd   -8(%rcx),%xmm0
        movlpd  %xmm0,-8(%rcx)
        movlpd  (%rsi),%xmm1
        mulsd   (%rax),%xmm1
        addq    $16,%rax
        addsd   (%rcx),%xmm1
        movlpd  %xmm1,(%rcx)
        addq    $16,%rcx
        testl   %edx,%edx
        jg      .LB224
```

pgcc -S -fastsse -Msafeptr

```
# lineno: 10
        movlpd  (%rdi,%rdx),%xmm1
        subl    $8,%eax
        prefetcht0      128(%rdx,%rdi)
        prefetcht0      128(%rdx,%rsi)
        addl    $8,%ecx
        movhpd  8(%rdx,%rdi),%xmm1
        mulpd   %xmm0,%xmm1
        addpd   (%rsi,%rdx),%xmm1
        movapd  %xmm1,(%rsi,%rdx)
        movlpd  16(%rdx,%rdi),%xmm2
        movhpd  24(%rdx,%rdi),%xmm2
        mulpd   %xmm0,%xmm2
        addpd   16(%rdx,%rsi),%xmm2
        movapd  %xmm2,16(%rdx,%rsi)
        movlpd  32(%rdx,%rdi),%xmm1
        movhpd  40(%rdx,%rdi),%xmm1
        mulpd   %xmm0,%xmm1
        addpd   32(%rdx,%rsi),%xmm1
                …
```

-Msafeptr causes the compiler to assume Fortran-like semantics, i.e. in this case that M1 and M2 cannot overlap

A heavy hammer…

In general, this is not a safe assumption to make in C

Pragmas are a more flexible alternative

Declaring Data Independency via #pragma

- Instruct compiler when to assume it is safe to take data directly from cache

- At the routine level:

```
#pragma routine -Msafeptr=arg
void matmul(double newMatrix[RANK*RANK],
            const double M1[RANK*RANK],
            const double M2[RANK*RANK]) {
```

- At the pointer level:

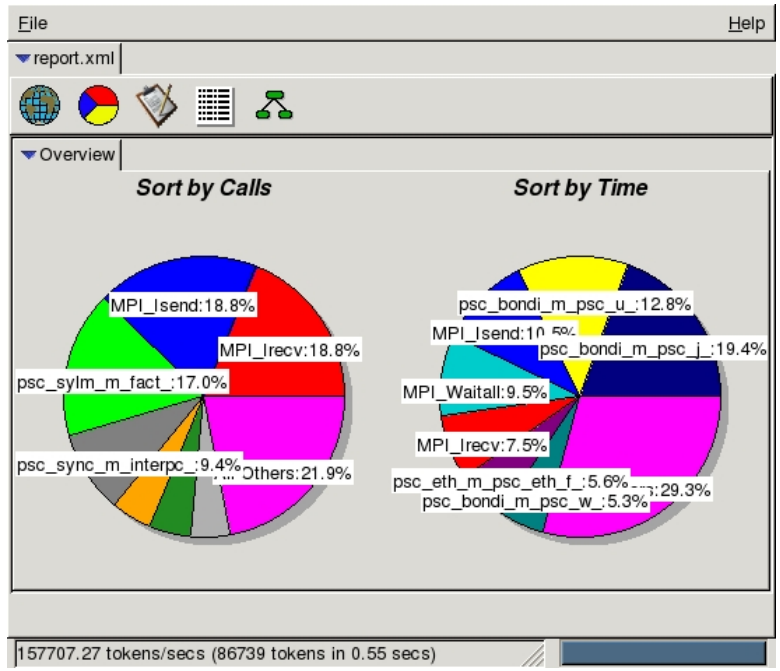
```
void matmul(double newMatrix[RANK*RANK],
            const double M1[RANK*RANK],
            const double M2[RANK*RANK]) {
#pragma safe ( newMatrix, M1, M2 ) ...
```

Using Craypat to detect inefficiencies

- LEO: A numerical relativity code developed by Roberto Gomez, Simonetta Frittelli, and Willians Barreto

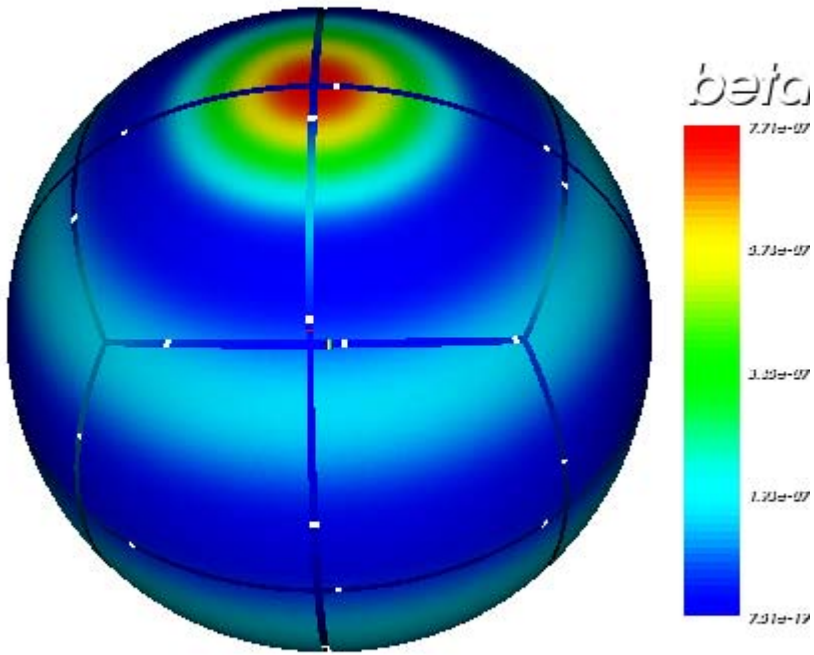
- Through Craypat, discovered that 27.5% of time was being spent in MPI_Isend(), MPI_Irecv(), and MPI_Waitall()

- Many small messages ⇒ performance suffers from effects of latency

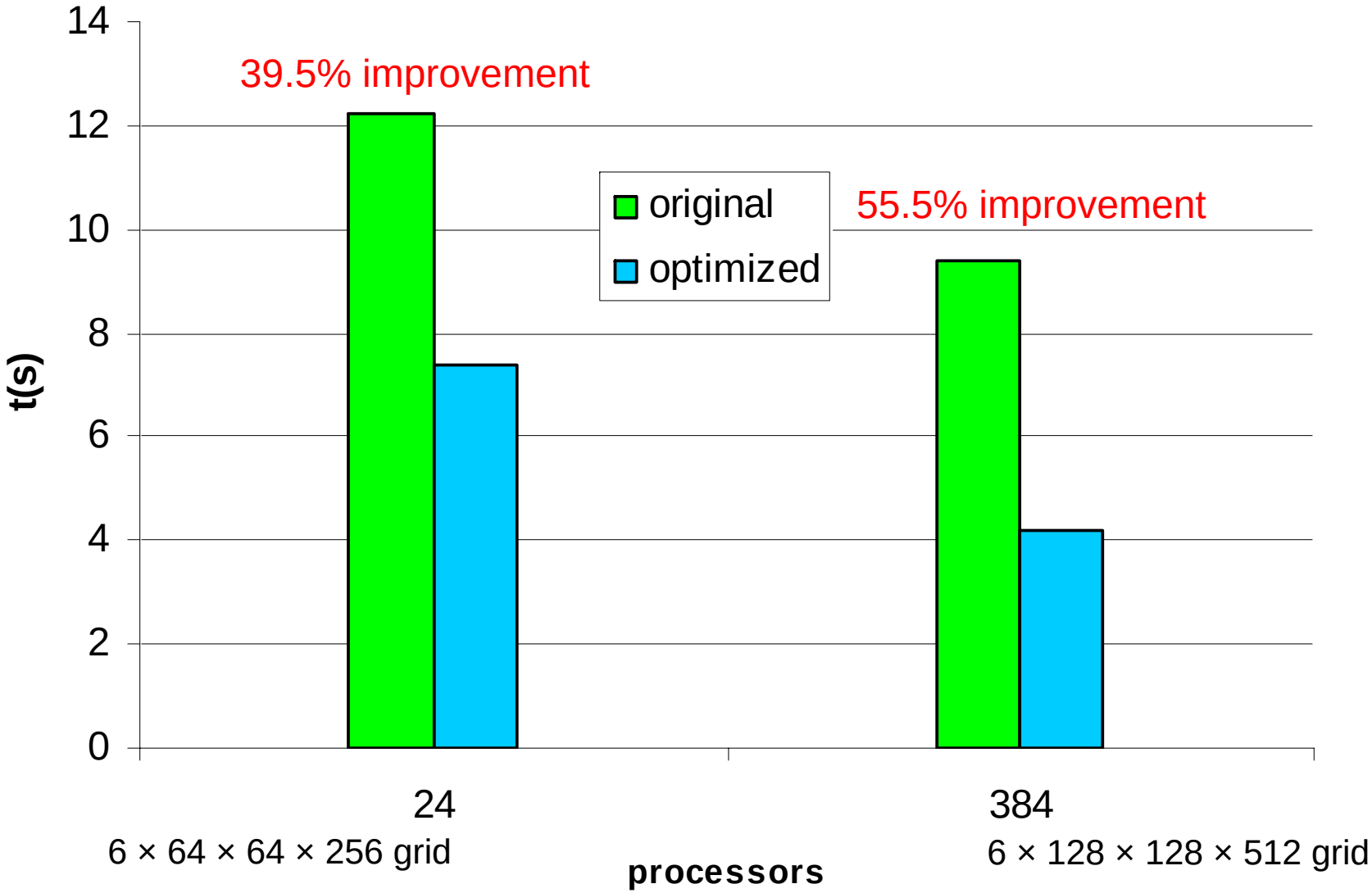


LEO Optimization

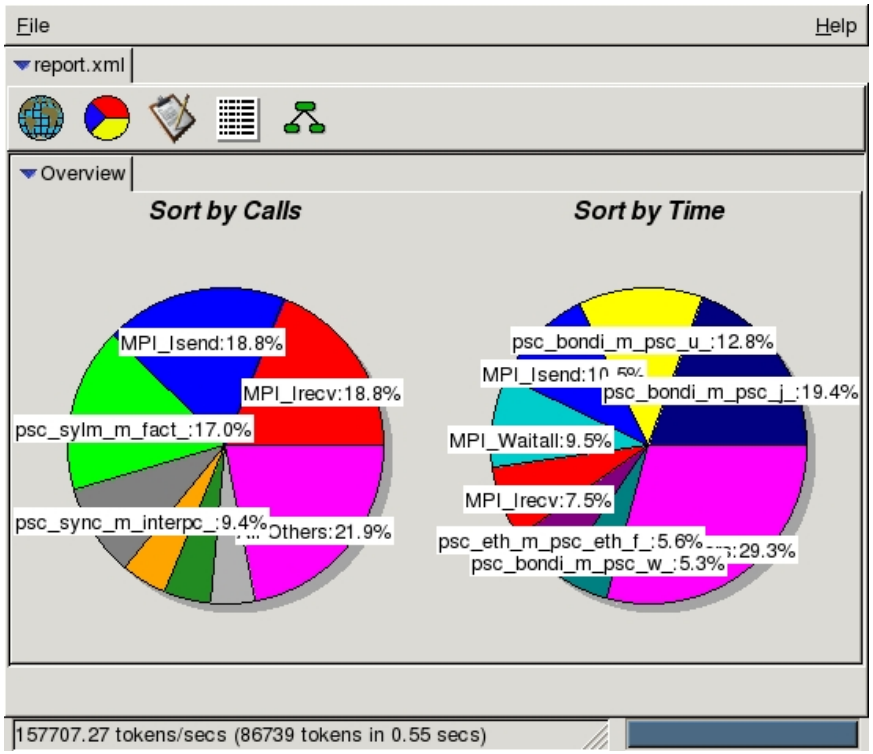
- Focused attention on reducing communication
- Moved the communication from inside the derivative routing to only where absolutely necessary
- Time spent in MPI_Isend(), MPI_Irecv(), and MPI_Waitall() decreased from 27.5% to 5.1% of the total time

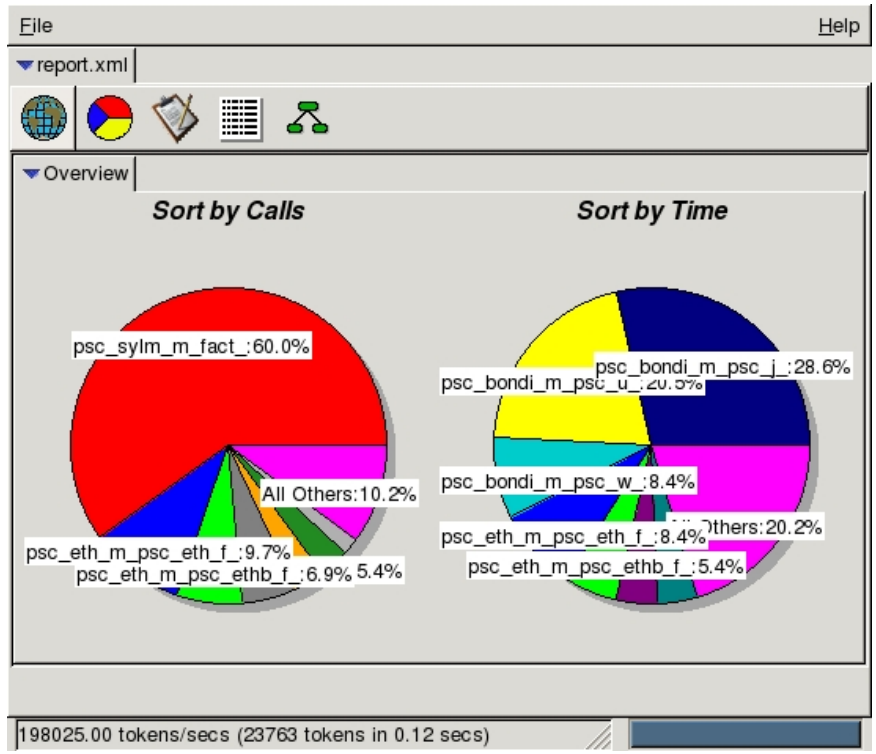


Effect of Optimizing Communications in LEO



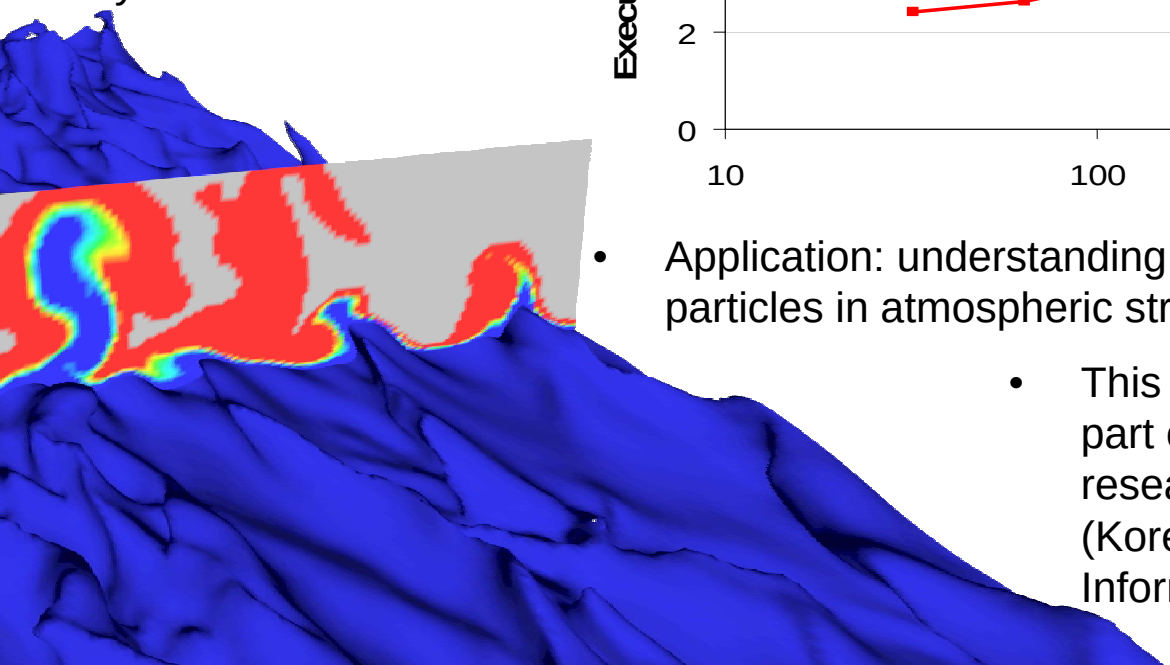
LEO: Before and After

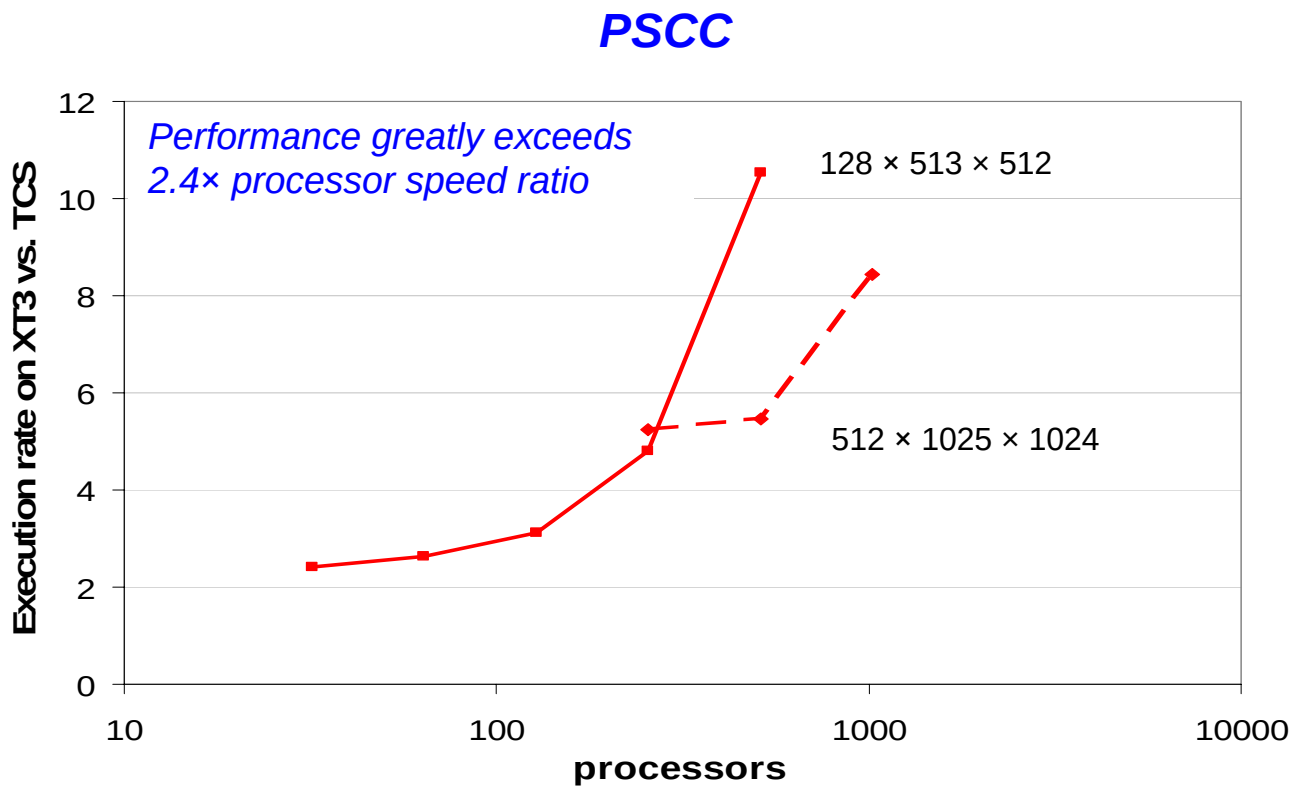




The Effect of Bandwidth

PSCC (Parallel Spectral Channel Code), developed by Junwoo Lim at PSC, is a highly scalable parallel flow solver to enable more realistic simulations of turbulent boundary flow, including tracking the traces of massless particles under stably stratified conditions.

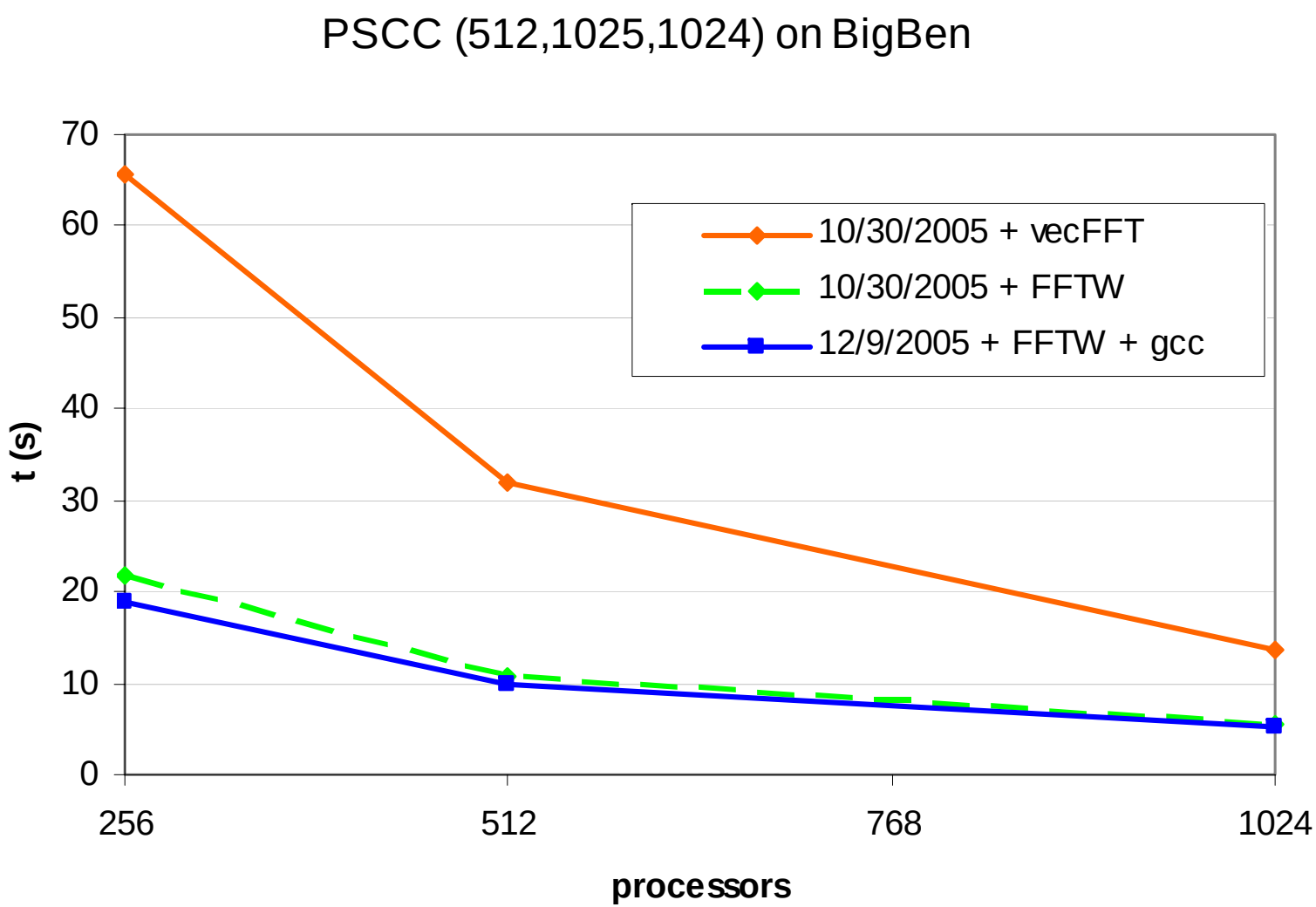




- Application: understanding the dynamics of pollutant particles in atmospheric stratified turbulent boundary layers.

- This project is sponsored by NSF and is a part of the international collaboration research program between PSC and KISTI (Korea Institute of Science and Technology Information) Supercomputing center.

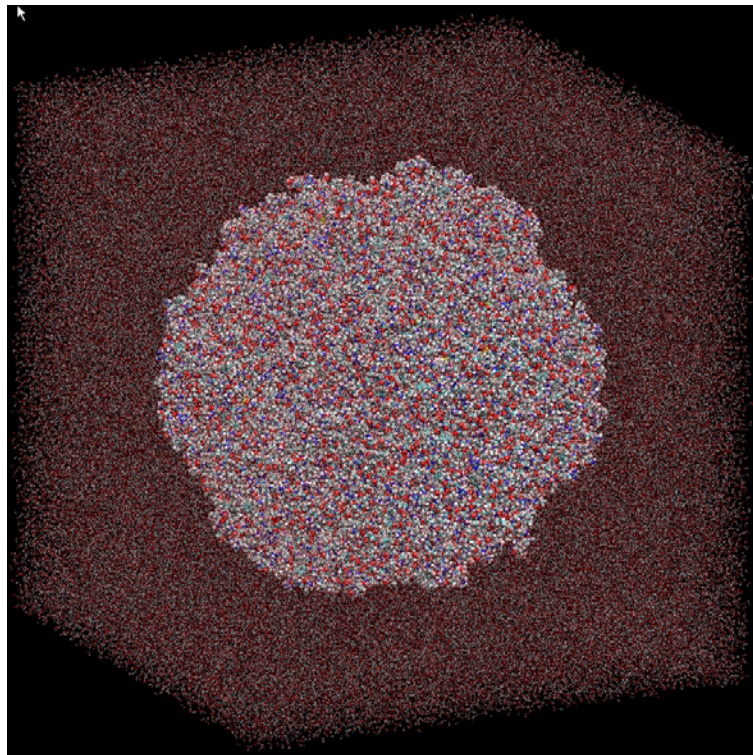
The effect of different FFT libraries



PITTSBURGH SUPERCOMPUTING CENTER

NAMD on the Cray-XT3

- Scalable simulation of large biomolecular systems
- 2002 Gordon Bell performance prize
- Built on Charm++: dynamic load balancing, latency hiding
- Accounts for a large proportion of usage on bigben
- Scaled to 5000 XT3 processors
  ("friendly user" account at ORNL)



- Performance Enhancements
  - gnu malloc (-lgmalloc): 20%
  - -small_pages: 20-40%

- Shawn Brown, PSC

The journey continues

- We are working closely with Prof. L. V. Kalé and his group to improve the efficiency and scaling of NAMD on the XT3.
  - developers of the CHARM++ programming language
  - profiling NAMD with the “Projections” application
- Based on these profiles, we are moving forward with a number of performance enhancements.
  - Improved MPI performance
  - Native Portals implementation of Charm++
- Optimal process mapping

PITTSBURGH SUPERCOMPUTING CENTER

Grazie!