Sputnik: Automated Decomposition on Heterogeneous Clusters of Multiprocessors

Photo courtesy of NASA.

Sean Philip Peisert

peisert@sdsc.edu

http://www.sdsc.edu/~peisert

LLNL — June 8, 2000

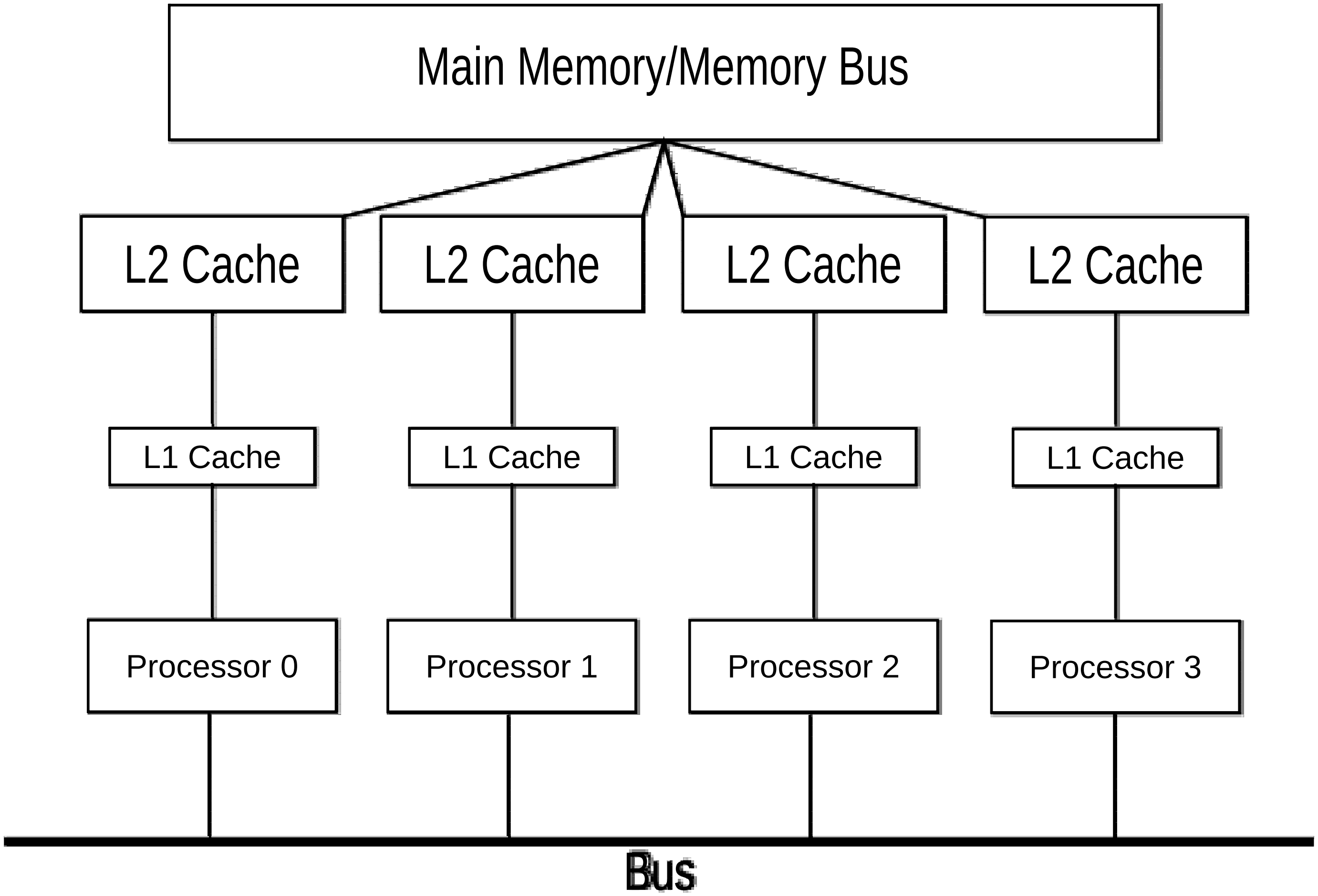
Collaboration

- This work was done under the advice of Professor Scott B. Baden at University of California, San Diego.

Motivation

- Supercomputers in science are evolving such that fewer and fewer are vector machines and mainframe multi*computers*. Most are clusters of multi*processors*.
- A multi*processor* is a shared-memory machine whereas a multi*computer* is “shared-nothing,” or distributed memory machine.

Multiprocessor

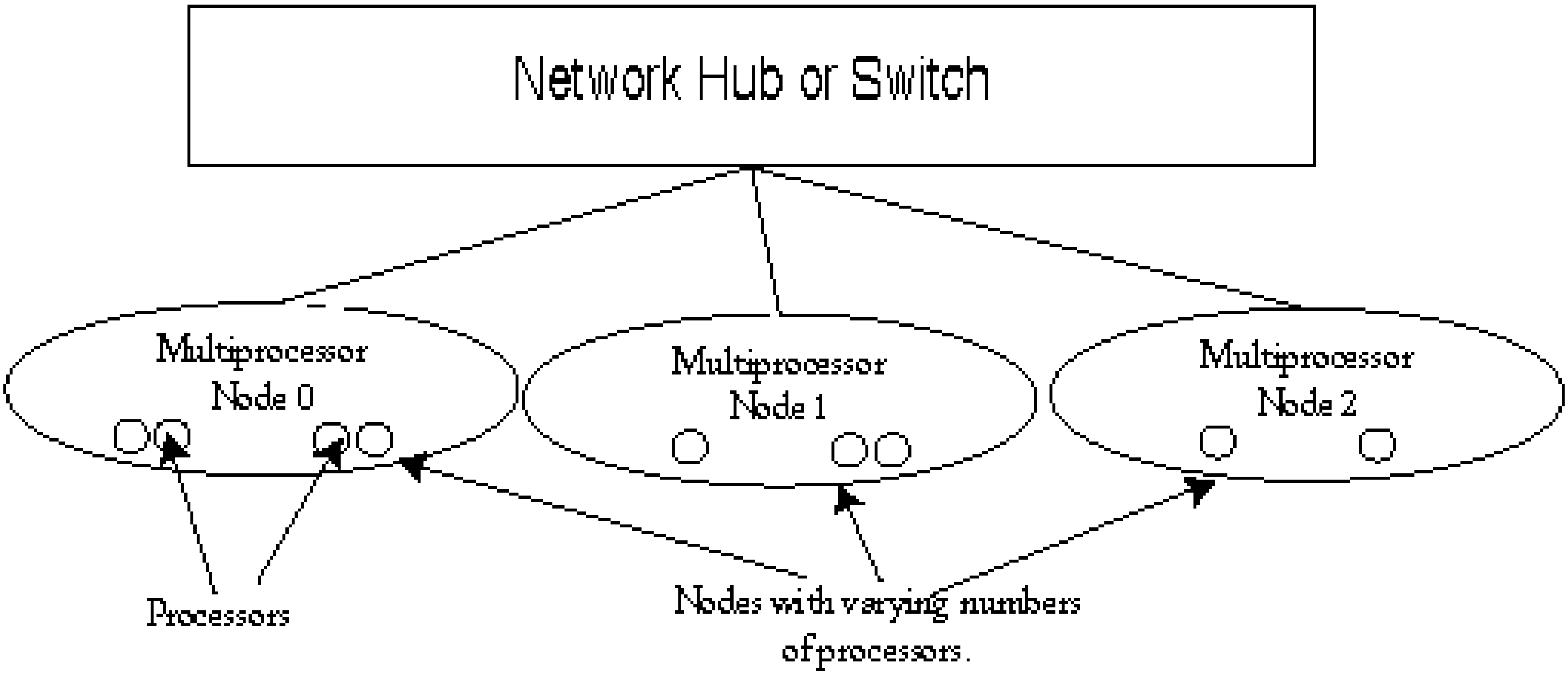


Clusters of Multiprocessors

- Multiprocessors are built largely using component parts.   They are also very modular.
- Easy to upgrade a portion of the nodes in the cluster with new nodes.
- Having different speed nodes in the cluster mean that programs have to be written differently.

Photo courtesty of SGI

Heterogeneous Cluster of Multiprocessors



Heterogeneity Problem

- *A parallel program will only run as fast as the slowest node.*
- For example, if one adds new nodes to a cluster that run faster than the existing nodes, the new nodes will finish first and idle until the slower nodes finish.

Heterogeneity Problem

- If processors are idle, they are wasting processing power when they could be processing data.
- The program is not performing optimally and can be run faster.
- The machine is not being used efficiently.

Related Work

- Fink and Baden: *KeLP2*
- Foster and Karonis: *Grid-Enabled MPI*
- Anglano, Schopf, Wolski and Berman: *Zoom*
- Crandall and Quinn: *Decomposition Advisory System*
- Wolski, Spring and Peterson: *Network Weather Service*

Heterogeneity Solution

- Optimize the program individually for each separate node based on prior information about each node.
- This is not easy.

Goals of *Sputnik*

- Allow a programmer to write software for a heterogeneous cluster as if the cluster is homogeneous. In other words, without adding much more complexity.
- Improve performance of the program being run and the utilization and efficiency of the cluster.

The *Sputnik* Model

- Two-stage process for optimizing performance on a heterogeneous cluster.
- *ClusterDiscovery*
- *ClusterOptimization*

ClusterDiscovery

- Performs a “resource discovery” — a search of a defined parameter space — to understand how the application in question runs on each individual node in the cluster.
- Runs the kernel repeatedly inside a “shell” to determine the best performing optimizations.

Cluster Discovery

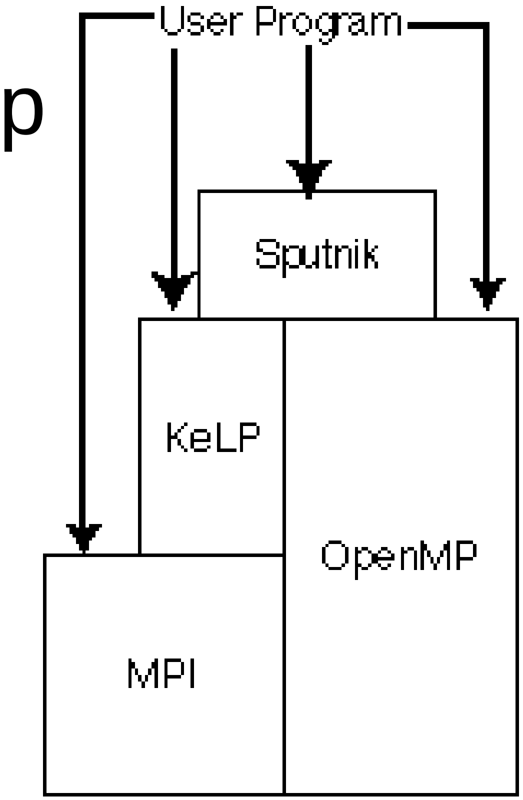
- Saves the best optimization data inside a file for future use.

ClusterOptimization

- Makes the specific optimizations for each node based on what the first stage has discovered.
- Some possible optimizations include: Adjusting the number of threads per node, cache tiling, data partitioning, machine and data “class” optimization.

The *Sputnik* API

- Focuses on just a few specific optimizations.
- The Sputnik API is built on top of KeLP, which is used for data description and inter-node communication.
- OpenMP is used for intra-node communication.



ClusterDiscovery (API)

- Instead of searching the entire parameter space for possible optimizations, I focused on two:
  - Adjusting the number of OpenMP threads.
  - Repartitioning the data.

ClusterDiscovery (API)

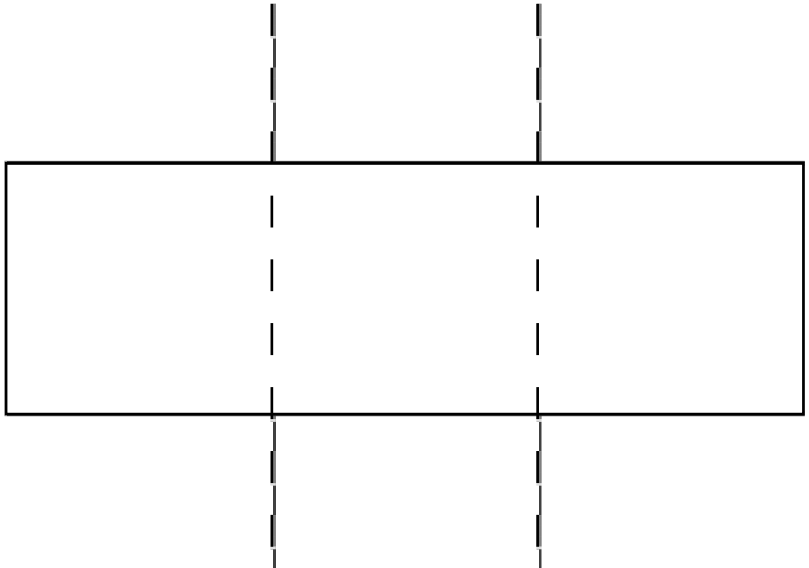
- Runs the kernel repeatedly with different numbers of threads on each node. When it finds the optimal number of threads for a given node, it saves the timing for that node.
- The saved timing is compared with the other timings in the cluster and ratios are formed.

ClusterOptimization (API)

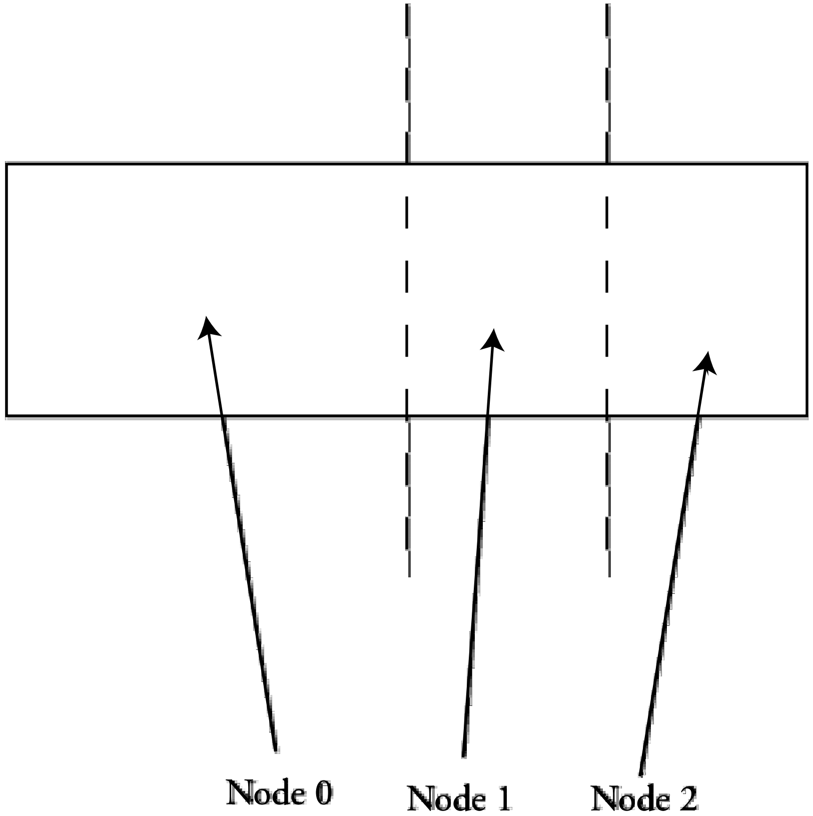
- The ratios from ClusterDiscovery are used to re-partition the data so that the *ratio of a given node’s power in relation to the rest of the cluster is the same fraction of the data that it works on.*

ClusterOptimization (API)

Original Partitioning:



New Partitioning:



Example: Node 0 runs twice as fast as node 1 and node 2.

API Limitations/Assumptions

- One-dimensional decomposition.
- Confined to two tiers of parallelism.
- Assumes that a node given a smaller chunk of data will run at the same MFLOPS rate as with the original size chunk.
- The problems do not fit into the highest level of cache.

API Limitations/Assumptions

- Assume no node is less than half as fast as any other node.

main()

```
int main(int argc, char **argv) {
   MPI_Init(&argc, &argv); // Initialize MPI
   InitKeLP(argc,argv);    // Initialize KeLP

   // Call Sputnik's main routine, which in turn will
   // then call SputnikMain().
   SputnikGo(argc,argv);
   MPI_Finalize();         // Close off MPI
   return (0);
}
```

SputnikGo()

```
while(i > 0 && time[last iteration] < time[second-to-last iteration]) {
    omp_set_num_threads(i);
    time[i] = SputnikMain(int argc, char **argv, NULL);
    i = i / 2;
}
i = iteration before the best we found in the previous loop;
while (time[last iteration] < time[2nd-to-last iteration]) {
    omp_set_num_threads(i);
    time[i] = SputnikMain(int argc, char **argv, NULL);
    i = i - 2;
}
omp_set_num_threads(optimal number);
time[i] = SputnikMain(int argc, char **argv, bestTimes);
```

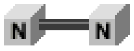
SputnikMain()

```
double SputnikMain(int argc,char ** argv, double * SputnikTimes) {
    double start, finish;
    ...
    <declarations, initializations>
    ...
    start = MPI_Wtime();  // start timing
    kernel();             // call the kernel function
    finish = MPI_Wtime(); // finish timing
    ...
    return finish-start;
}
```

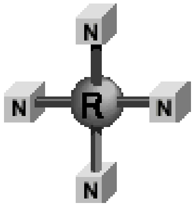
Application Study

- The purpose of the experiment is to determine the effect of these optimizations.
- I use a kernel that solves Poisson's equation using Gauss-Seidel's method with red-black ordering was used.

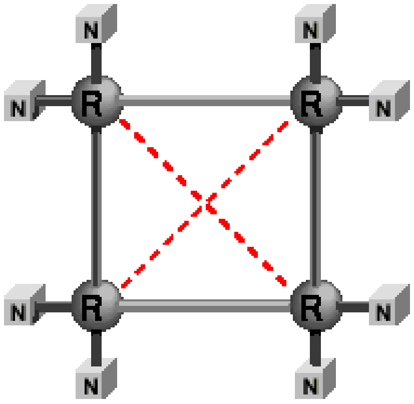
```
!$OMP PARALLEL PRIVATE(jj,ii,k,j,i,jk)
 do jj = ul1+1, uh1-1, sj
   do  ii = ul0+1, uh0-1, si
!$OMP DO SCHEDULE(STATIC)
     do k = ul2+1, uh2-1
       do  j = jj, min(jj+sj-1,uh1-1)
         jk = mod(j+k,2)
         do  i = ii+jk, min(ii+jk+si-1,uh0-1), 2
            u(i,j,k) = c *
     2       ((u(i-1,j,k) + u(i+1,j,k)) + (u(i,j-1,k) +
     3       u(i,j+1,k)) + (u(i,j,k+1) + u(i,j,k-1) -
     4       c2*rhs(i,j,k)))
    end do
       end do
     end do
!$OMP END DO
    end do
  end do
!$OMP END PARALLEL
```



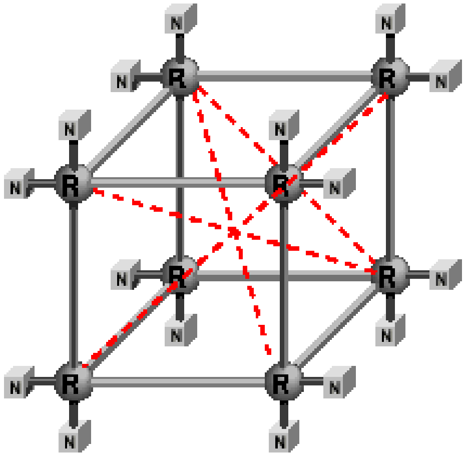
4 processor system



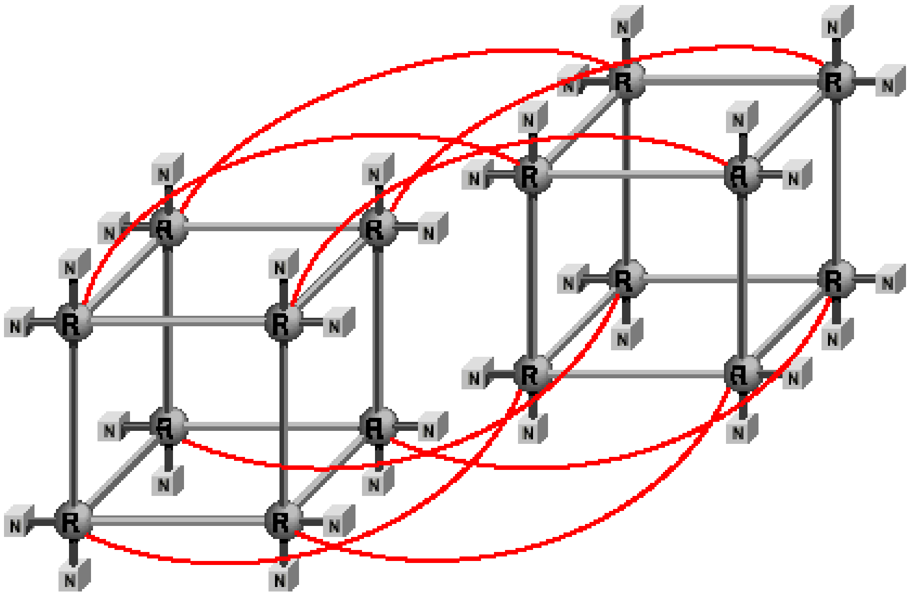
8 processor system



16 processor system



32 processor system



64 processor system

Image courtesy of SGI.

Computing Hardware

SGI Origin2000’s

balder.ncsa.uiuc.edu

- 256 250-MHz R10000 processors
- 128 GB main memory

aegir.ncsa.uiuc.edu

- 128 250-MHz R10000 processors
- 64 GB main memory

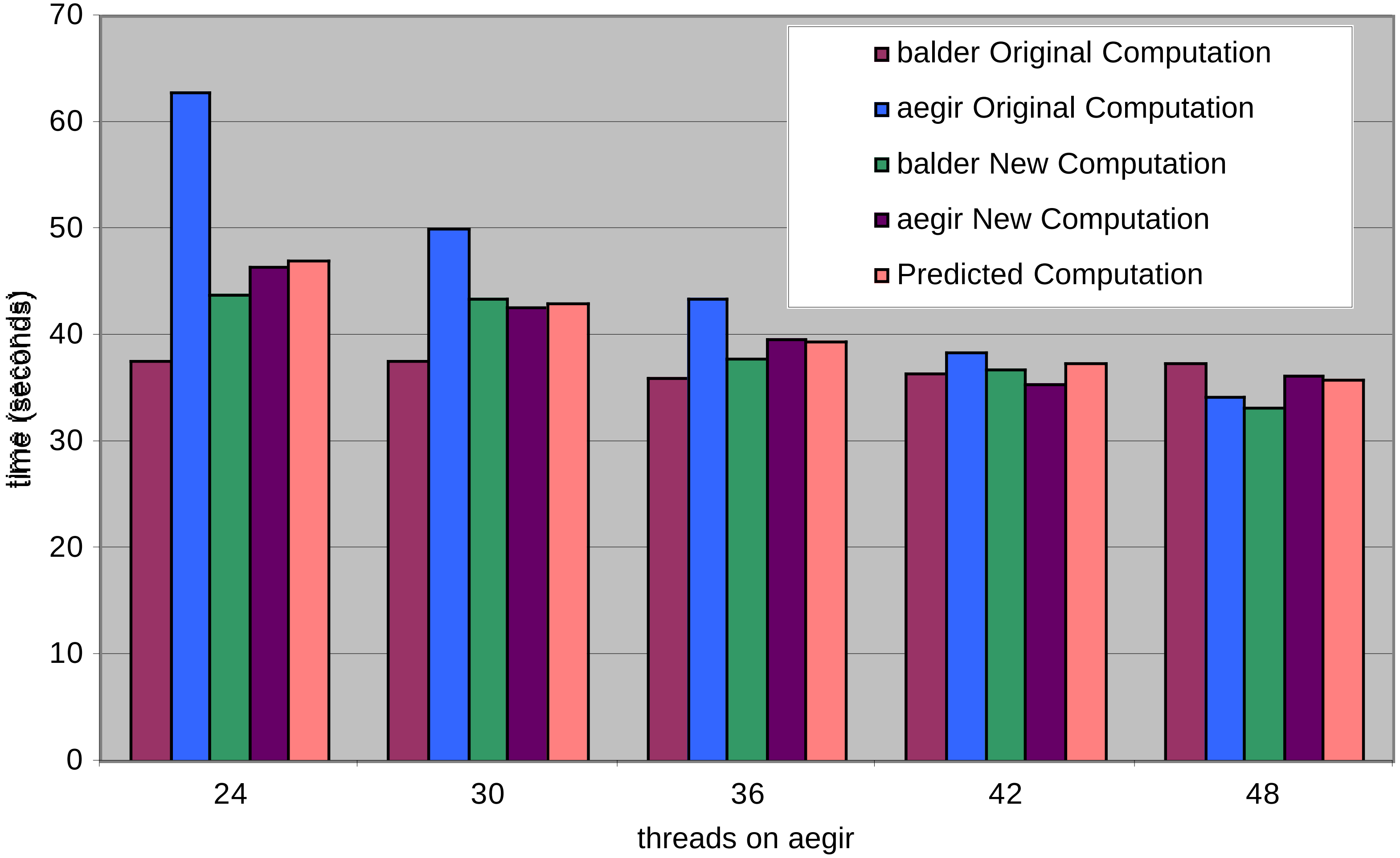
Virtual Cluster

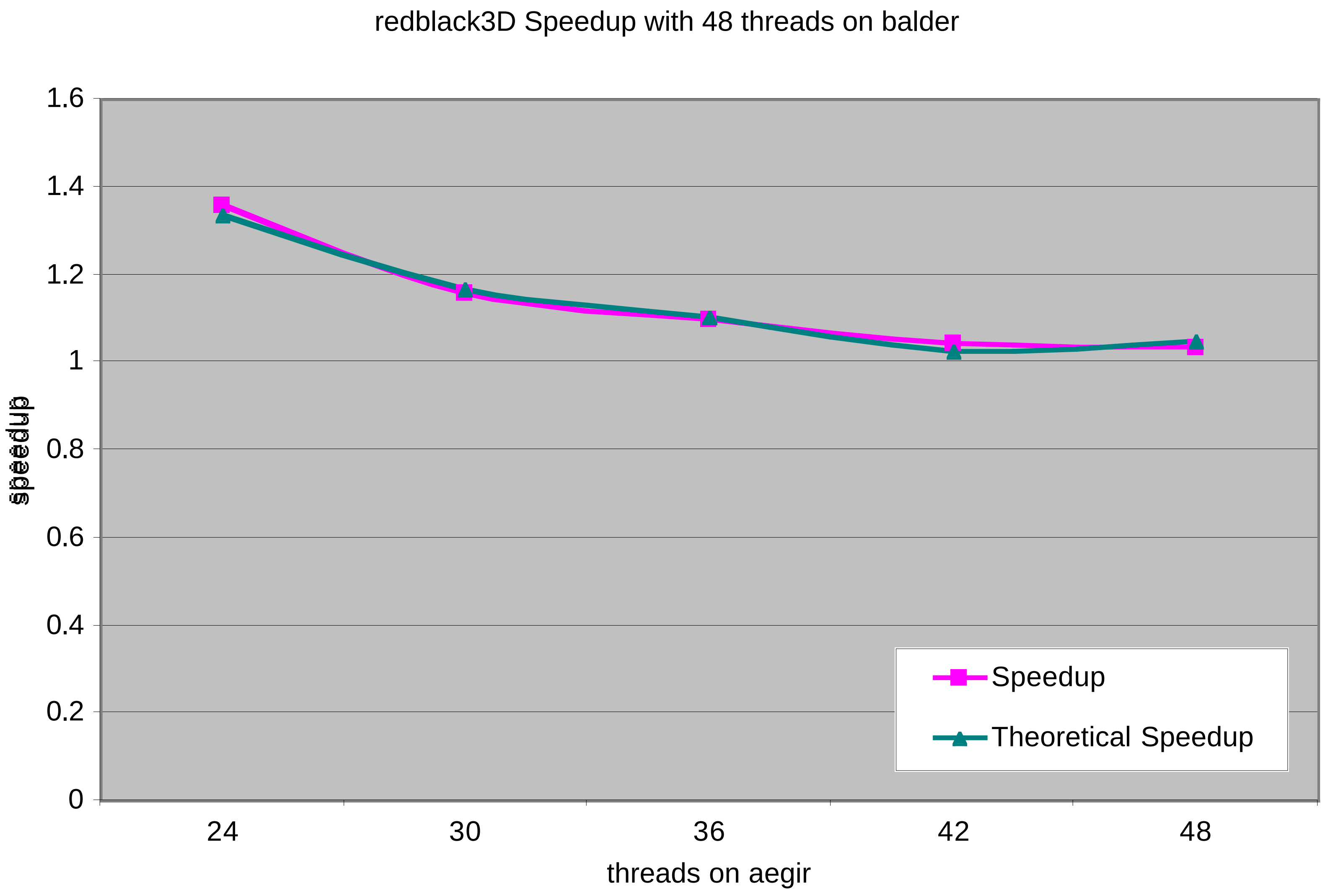
- *Sputnik* API is designed for commodity clusters. None were available, so a pair of SGI Origin 2000’s at NCSA were used.
- The API allows the number of OpenMP threads to be set manually.
- Different numbers of threads used on each Origin to simulate heterogeneity.

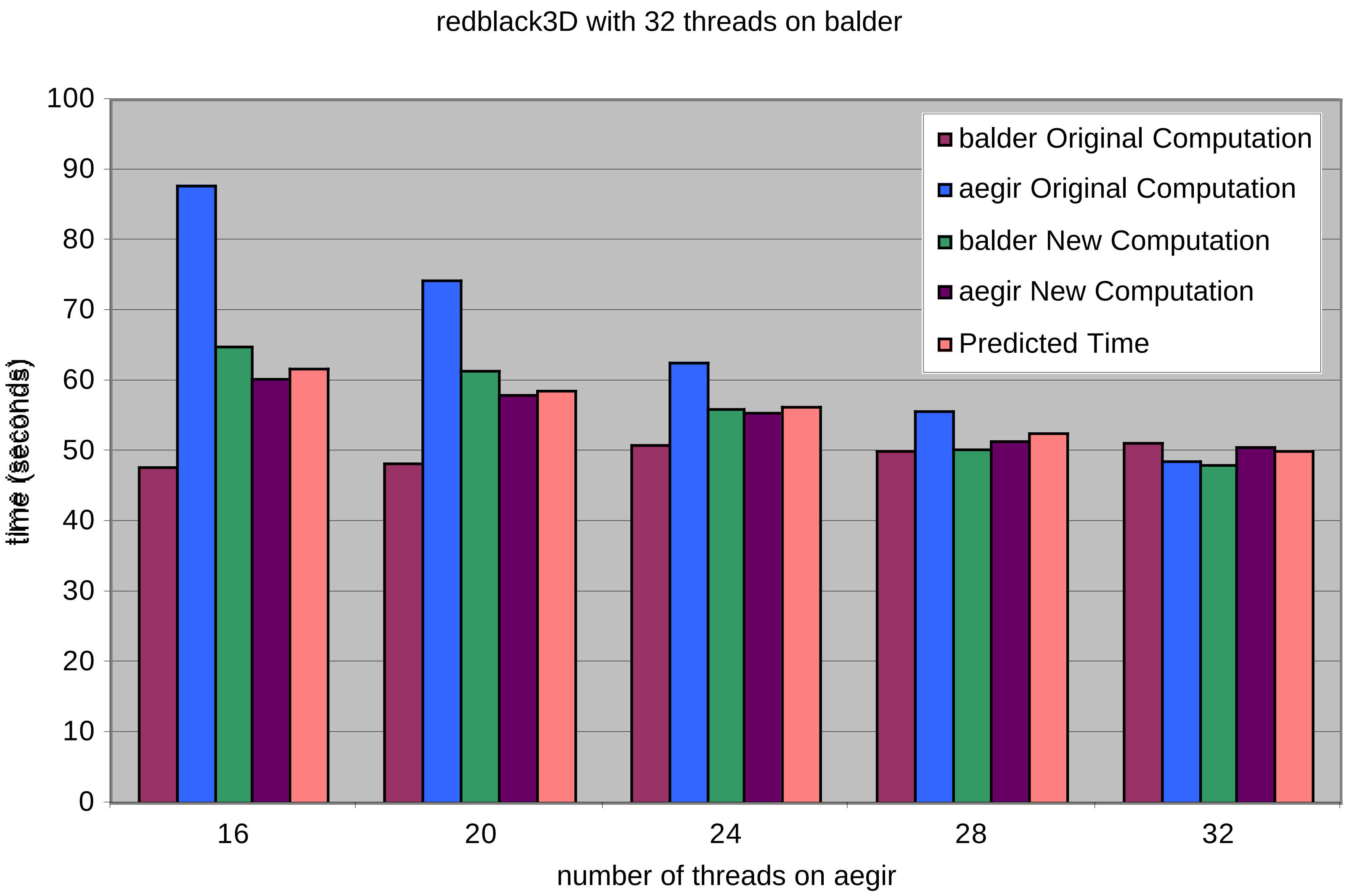
Predicted Time

$$
\begin{aligned}
T_{optimal} &= T_{i,orig} * \frac{\mathrm{newamountofdatafornode}i}{\mathrm{originalamountofdatafornode}i} \\
&= \frac{work_{total}}{work_{i,orig}} * \frac{\sum_{j=0}^{N-1} T_j}{\sum_{k=0}^{N-1} \frac{\sum_{j=0}^{N-1} T_j}{T_{k,orig}}}
\end{aligned}
$$

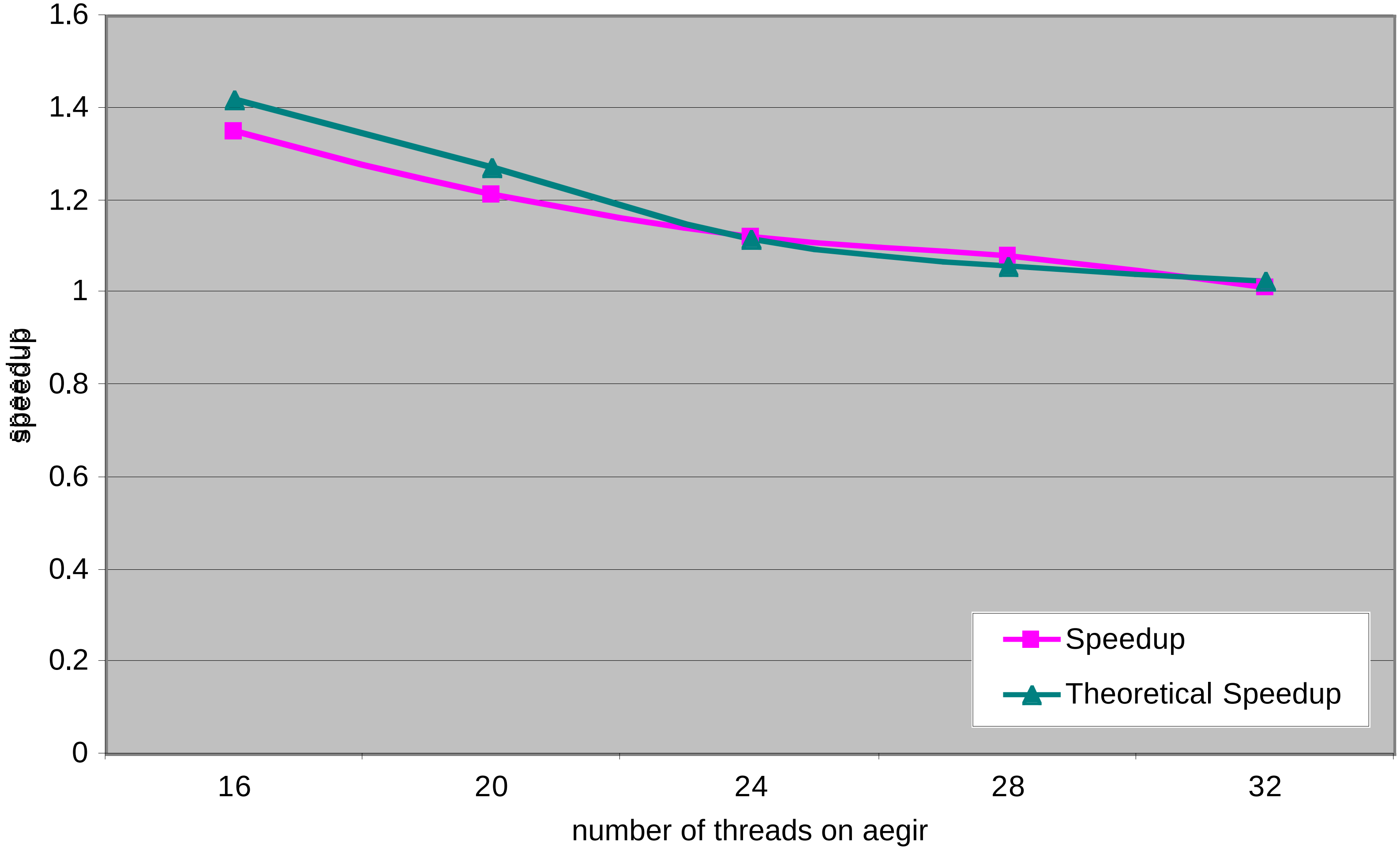
redblack3D - 48 threads on balder




redblack3D Speedup with 48 threads on balder
speedup
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
24
30
36
42
48
threads on aegir
Speedup
Theoretical Speedup


redblack3D with 32 threads on balder
time (seconds)
100
90
80
70
60
50
40
30
20
10
0
16
20
24
28
32
number of threads on aegir
balder Original Computation
aegir Original Computation
balder New Computation
aegir New Computation
Predicted Time

redblack3D Speedup with 32 threads on balder

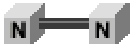


Validation

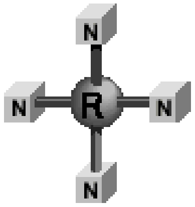
- Results for the API indicate better than 35% improvement in the situations where *balder* is running twice as many threads as *aegir*.
- The model and the API both succeed in the goal of being easy to program and improving performance.

Anomalies

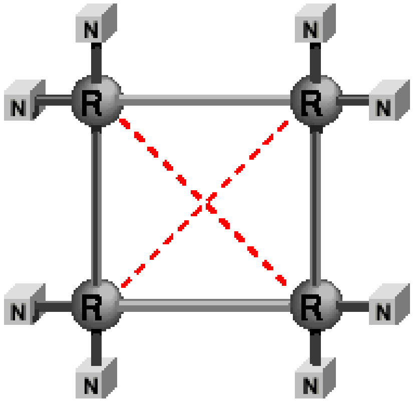
- The code demonstrated scaling, but ran 50% slower than with MPI alone, (without OpenMP).
- OpenMP thread binding and memory distribution are both complex issues on the Origin 2000 that are the probable causes of the slowdown.
- Real target of *Sputnik* API is commodity cluster.



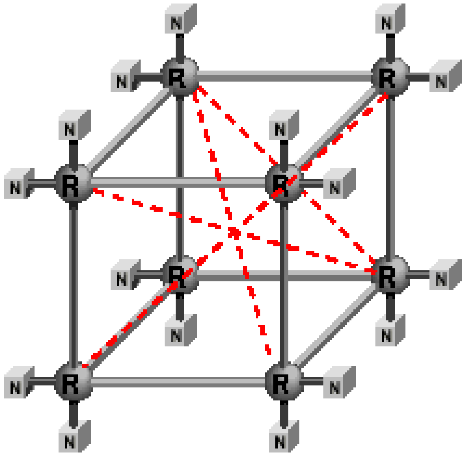
4 processor system



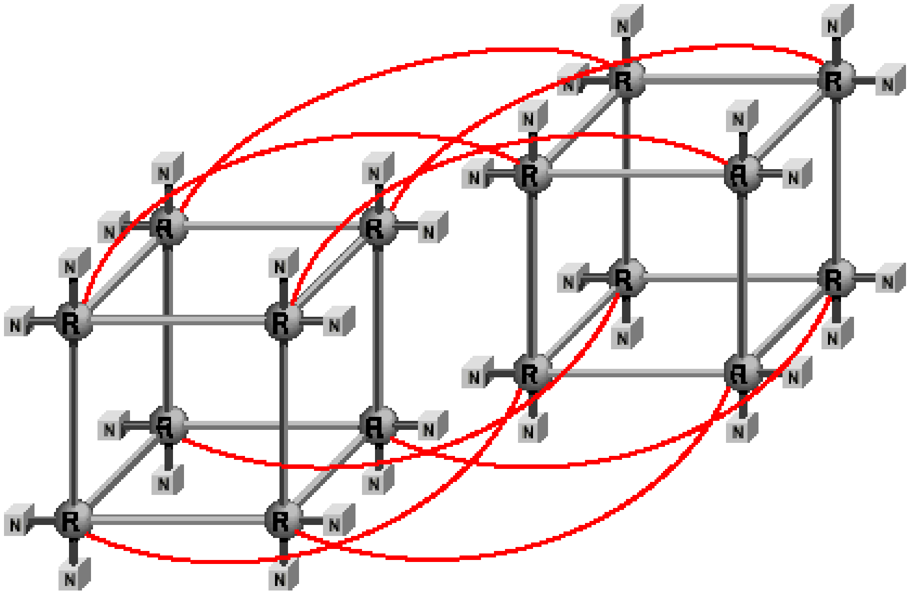
8 processor system



16 processor system



32 processor system



64 processor system

Image courtesy of SGI.

Future Work

- Tests on more applications and computing hardware, especially a cluster of Sun servers and Blue Horizon at SDSC.
- Dynamic repartitioning for grid/metacomputing applications.
- Supporting Phenomenally Heterogeneous Clusters (PHCs) — not just multicomputer-based clusters.

Future Work

- Different types of optimizations (not just repartitioning and adjusting the number of OpenMP threads).

Sean Philip Peisert

peisert@sdsc.edu

http://www.sdsc.edu/~peisert

Lawrence Livermore National Labs

Sputnik Talk

June 8, 2000