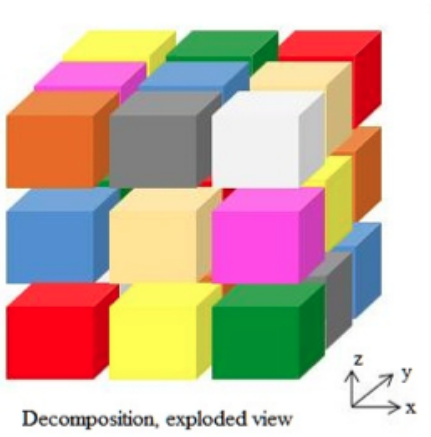
NPB Application Kernels

A 1D partitioning distributed approach

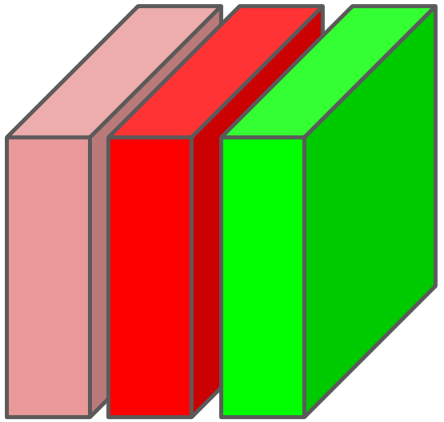
Brook Kassa, Prerna Narayanpure, Verónica Vergara L.

1D-MPI-NPB

Original distributed NPB uses multidimensional data partitioning



The original shared memory NPB uses a 1D data partition



Goal: Implement a distributed version of NPB that uses a 1D data partitioning scheme

NPB Application Kernels

The NPB application kernels simulate processes on real CFD applications. The three solvers are:

- **B**lock **T**ridiagonal (BT): solves multiple, independent systems of non-diagonally dominant, block tridiagonal equations with 5 by 5 block size
- **L**ower **U**pper diagonal (LU): employs a symmetric successive over-relaxation (SSOR) numerical scheme to solve a regular, sparse, 5 by 5 lower and upper triangular systems.
- **S**calar **P**entadiagonal (SP): solves three sets of uncoupled scalar pentadiagonal systems of equations in x, y and z direction.

NPB Application Kernels

- *Number of lines of code*

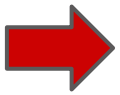
App. Kernel	Main Routine	Total
BT	*328*	*9162*
SP	*245*	*4902*
LU	*199*	*5957*

- *Problem Size*

1) Class 'S' - 12 x 12 x 12
2) Class 'A' - 64 x 64 x 64
3) Class 'C' - 162 x 162 x 162

Translation Steps: `for` loop

```
#pragma omp for
for(k=0; k<Nz; k++)
  for(j=0; j<Ny; j++)
     for(i=0; i<Nx; i++)
        rhs[k][j][i] =
...
```

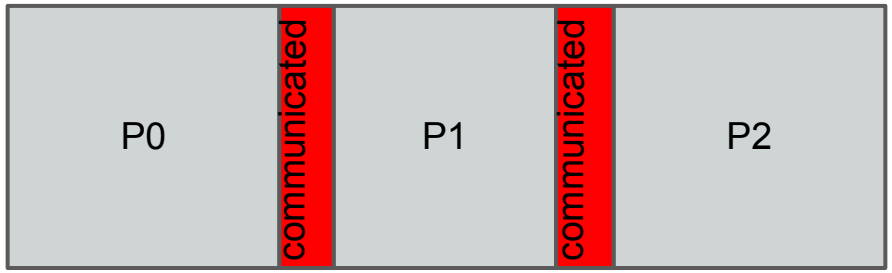


```
size = # proc
rank = my proc ID
kgap = Nz/size
kstart = rank * kgap
kend = kstart + kgap

for(k=kstart; k<kend; k++)
  for(j=0; j<Ny; j++)
     for(i=0; i<Nx; i++)
        rhs[k][j][i] = ...
```

Translation Steps

```
for (k=0; k < Nz; k++)
 for (i=0; i < Nx; i++)
  for (j=0; j < Ny; j++)
    rsd[k][i][j] = ...
...
```




```
MPI_Allgather (rsd[kstart][0][0],rsd[0][0][0])
```

```
if(my_id == 0)
  MPI_Send(&rsd[kend - 2][0][0],...,rank+1,...)
else
  MPI_Recv(&rsd[kstart - 2][0][0],...,rank-1,...)

... = rsd[k-2][i][j] + ...
```

Translation Steps : Other transformation

- Reduction operation

```
#pragma omp atomic
  sum = sum + x
```

```
MPI_Allreduce(gsum,
sum)
```

- Master construct

```
#pragma omp master
  time_start(trhs)
```

```
if(rank == root)
  time_start(trhs)
MPI_Barrier...
```

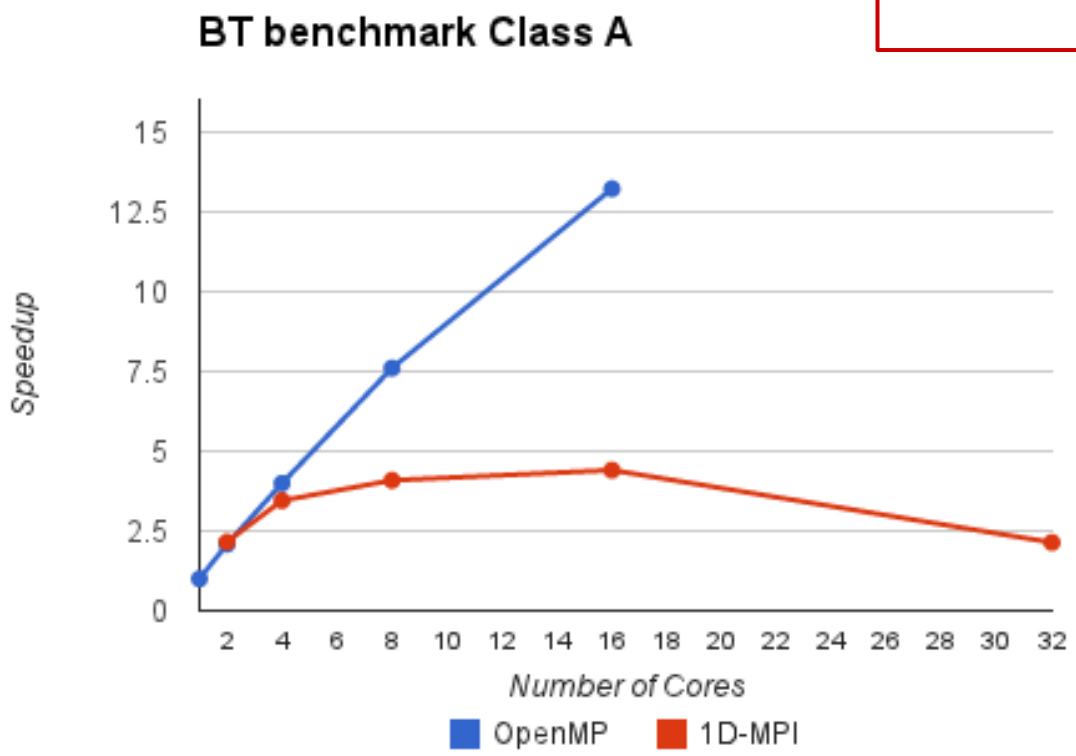
- Flush construct

```
#pragma omp flush
(isynch)
```

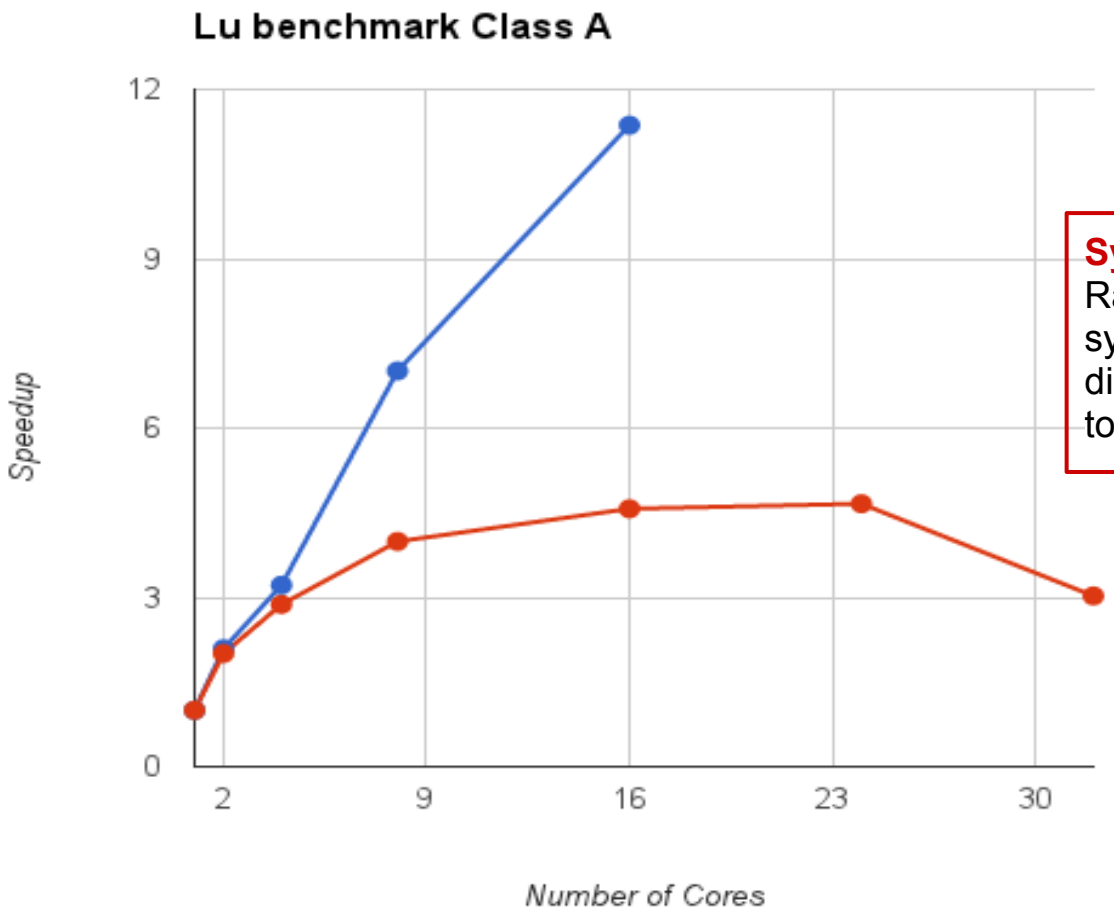
```
MPI_Bcast(isynch,root)
```

Results : BT Benchmark

System for Experiments:
Carter cluster with two 8-core Intel Xeon-E5 nodes with 32 GB of memory and 56 Gbps FDR Infiniband interconnects.



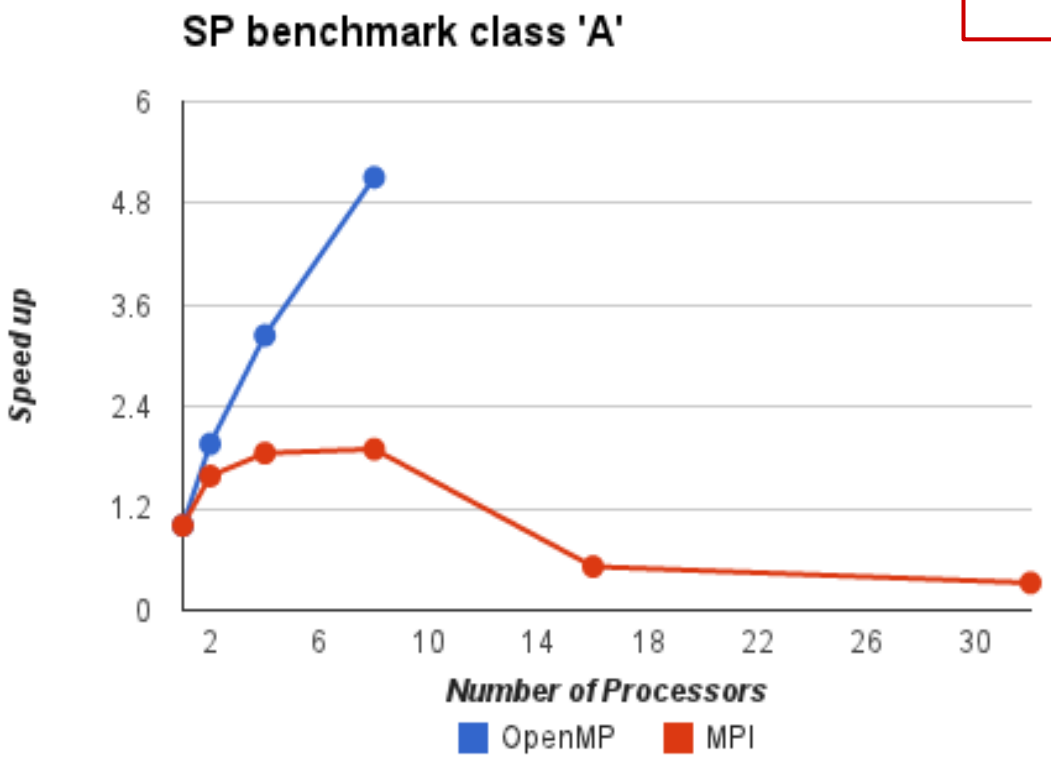
Results : LU Benchmark



System for Experiments:
Radon cluster: 24 64-bit, 8-core Dell 1950 systems with 16 GB RAM, 160 GB of disk, and 1 Gigabit Ethernet (1GigE) local to each node.

Results : SP Benchmark

System for Experiment
ACSL cluster with 10 nodes each having 2 quad core AMD opteron, 16GB RAM and GigE network.

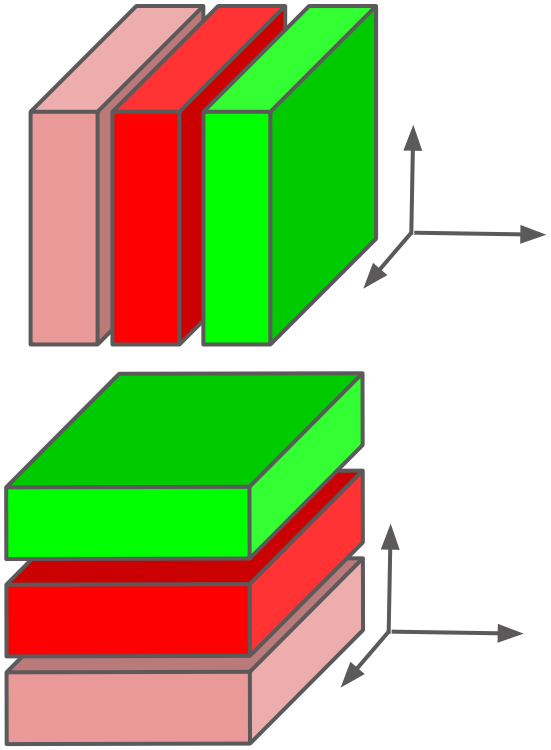


Challenges

- Overhead for MPI communication specially with collective communication.
- OpenMP parallelization across different dimensions

```
#pragma omp for
for(k=0; k<Nz; k++)
  for(j=0; j<Ny; j++)
     for(i=0; i<Nx; i++)
        rhs[k][j][i] = ...
```

```
for(j=jstart; j<jend; j++)
   for(k=0; k<Nz; k++)
     for(i=0; i<Nx; i++)
        rhs[k][j][i] = ...
```



Because of some true dependencies some matrix had to be transposed and partitioned on second dimension of the array.

Future Work

- Increase problem size to class C
- Increase computation and communication overlap by interleaving non-blocking point-to-point communications
- Implement algorithmic changes that will allow us to exploit further parallelism

Questions?

Results : BT Benchmark (with MPI)