



IBM Research

Maximizing Multinode Performance in BlueGene/L

José E. Moreira

IBM Thomas J. Watson Research Center

Goal

- To illustrate techniques that support efficient execution of MPI programs in BlueGene/L
- Focus on concepts, not details – simplifications have been made!

Outline

- Principles of BlueGene/L design
- Programming BlueGene/L with MPI
- Case study – Jacobi relaxation
- Conclusions

Outline

- Principles of BlueGene/L design
- Programming BlueGene/L with MPI
- Case study – Jacobi relaxation
- Conclusions

Principles of BlueGene/L design

■ Simplicity

- ❖ Need an operational machine at 64k nodes, 128k processors
- ❖ Limited purpose machine – enabled simplifications
- ❖ Reliability through simplicity

■ Efficiency

- ❖ BlueGene/L is all about performance
- ❖ Simplicity enables efficiency
- ❖ High performance without sacrificing security

■ Familiarity

- ❖ Standard programming languages and libraries
- ❖ Enough functionality to deliver a familiar system without sacrificing simplicity or high performance

Things to remember

- Strictly space sharing
 - ❖ One job (one user) per electrical partition of the machine
 - ❖ One process per compute node (or virtual node)
 - ❖ One thread of execution per processor
- Flat view for application programs – collection of compute processes
- Compute processes have to be mapped to a nearest-neighbor interconnection hardware
- Fortran, C, C++ with MPI
 - ❖ Full language support
 - ❖ Automatic SIMD FPU exploitation
- Tools – support for debuggers, hardware performance monitors, trace based visualization

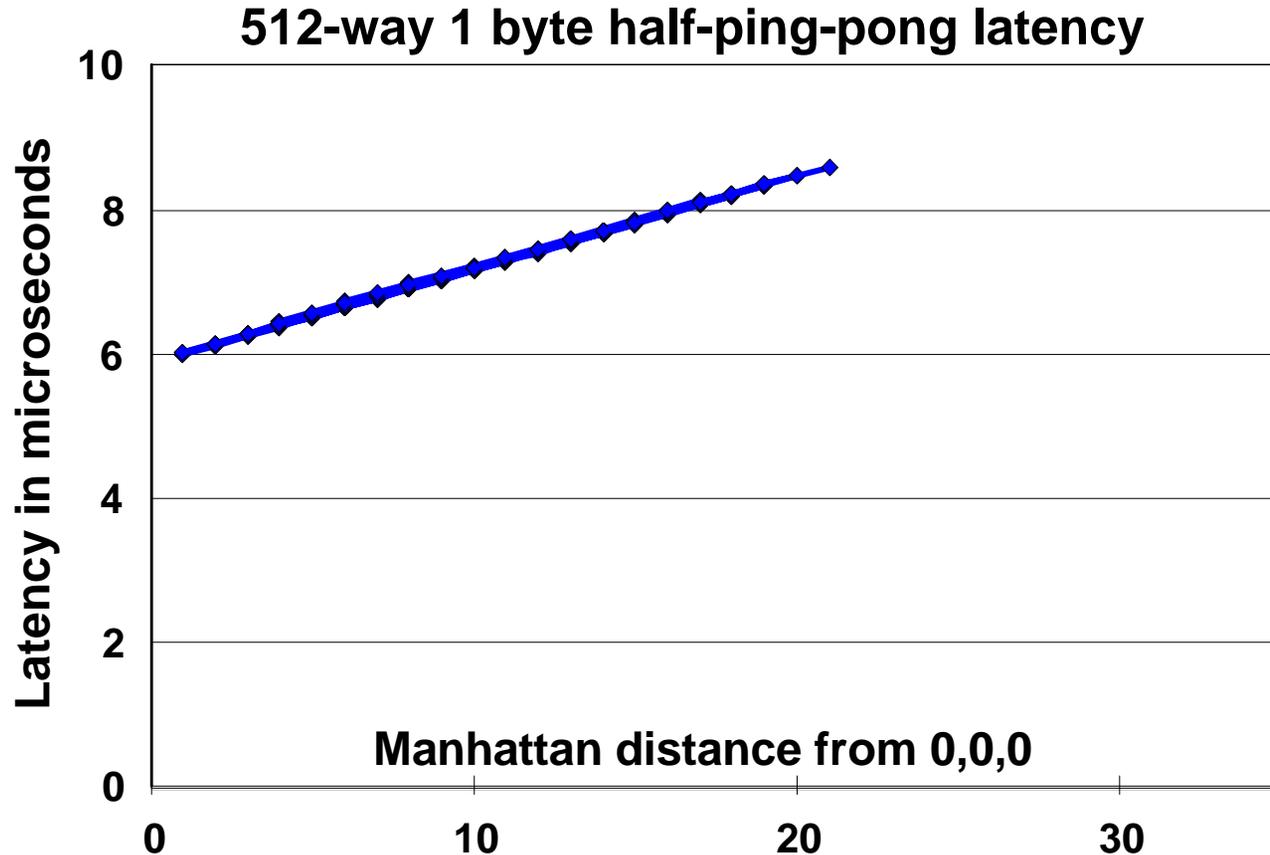
Outline

- Principles of BlueGene/L design
- **Programming BlueGene/L with MPI**
- Case study – Jacobi relaxation
- Conclusions

MPI performance on BlueGene/L

- Proper mapping of MPI applications to machine topology
 - ❖ Better utilization of torus bandwidth (avoid congestion)
 - ❖ Lower latency by avoiding multiple node hops
 - ❖ Leverage hardware support for multicasts – deposit bit in torus supports efficient multicast in rectangular (1D, 2D, 3D) regions
 - ❖ Strategy: MPI plus PMI (process management)
- Parallel communication channels
 - ❖ Each BlueGene/L compute node has six independent torus links
 - ❖ Each link connects to one of its nearest neighbors
 - ❖ The links are not interchangeable!

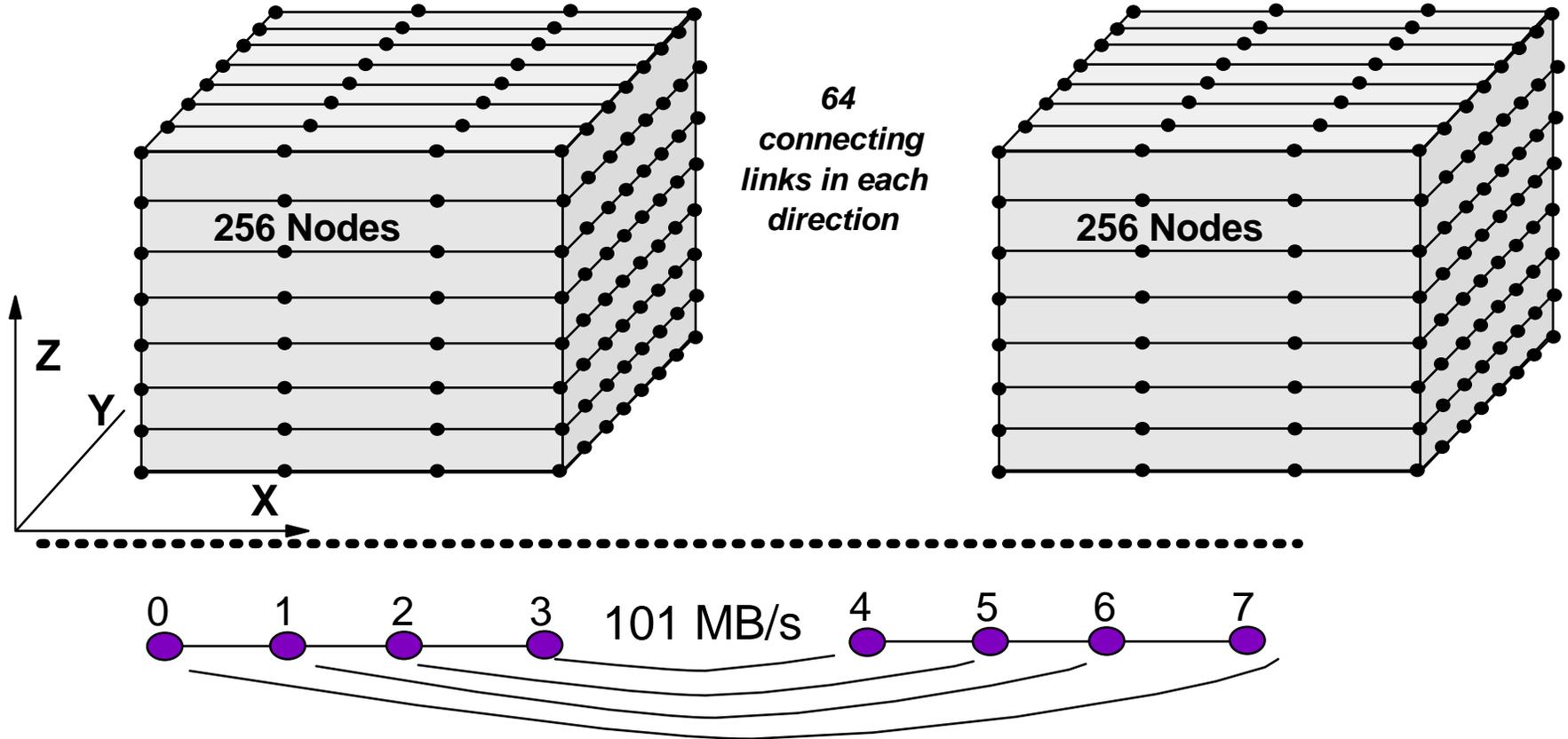
Latency as a function of distance



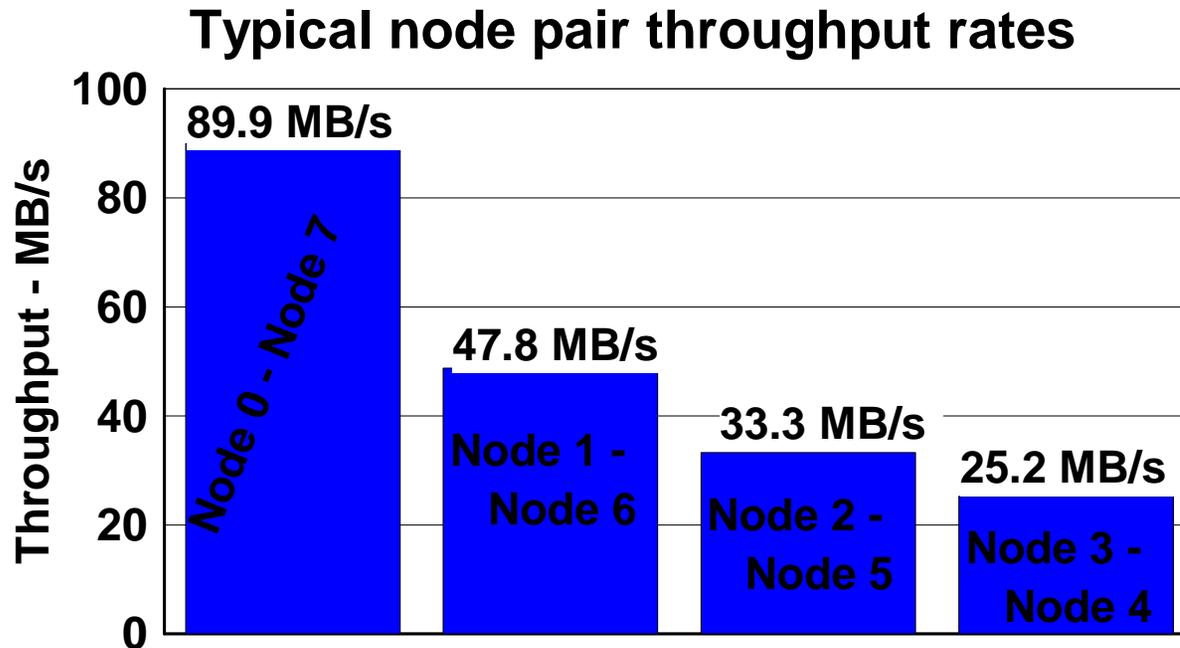
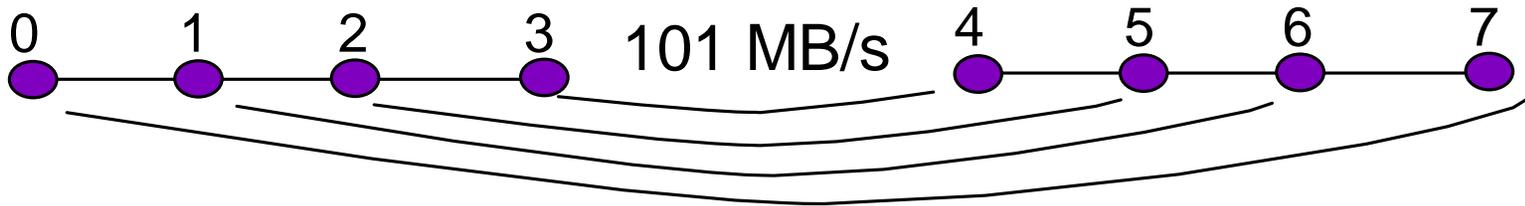
■ Latency @500 MHz = $5.9 + 0.13 * \text{"Manhattan distance"} \mu\text{s}$

■ Latency @700 MHz = $4.2 + 0.09 * \text{"Manhattan distance"} \mu\text{s}$

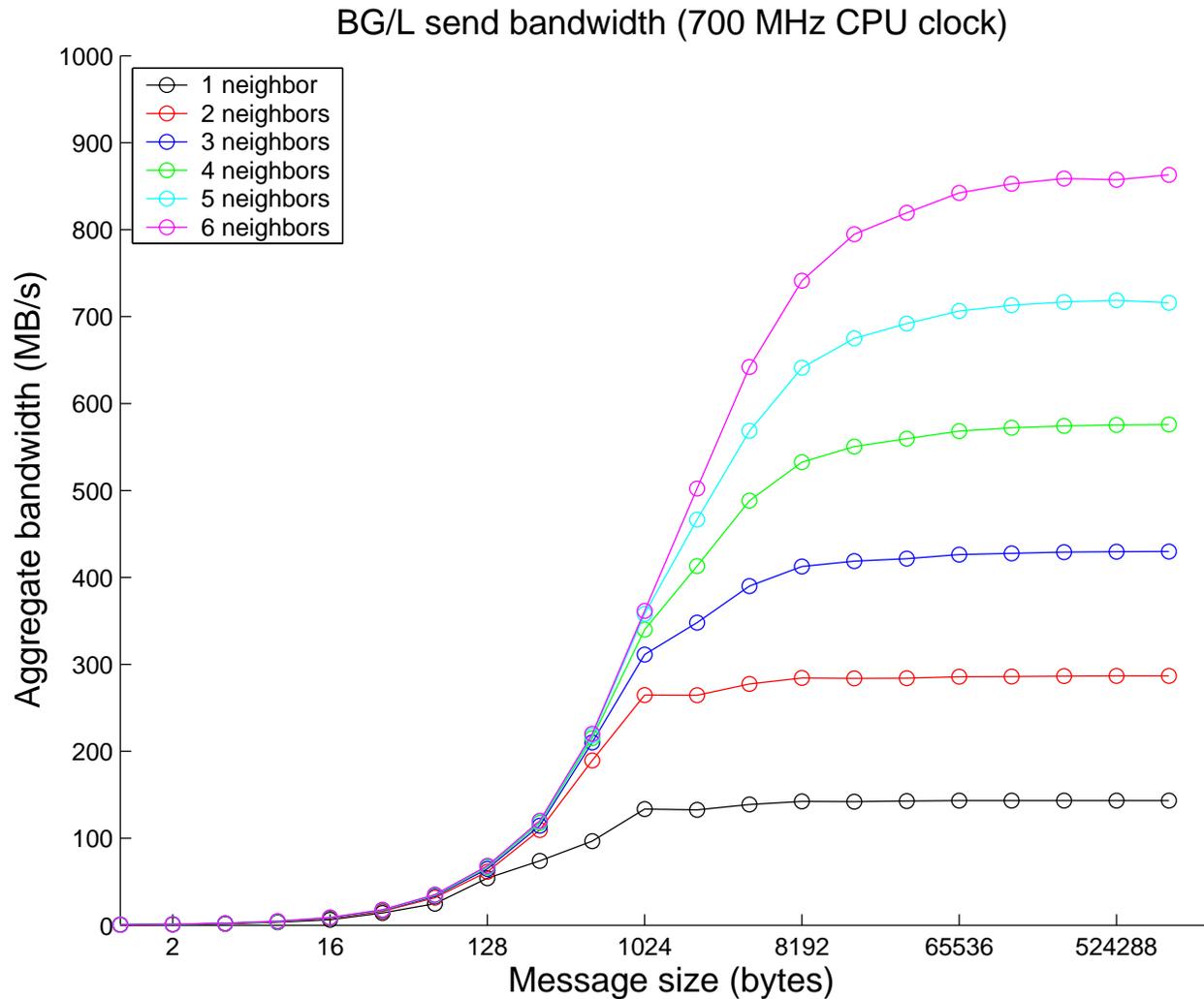
Bi-section bandwidth



The impact of congestion



Parallel communication channels



PMI functions for mapping

- `void PMI_rank2torus (int rank, int *x, int *y, int *z);`
 - ❖ Translates a task MPI rank to its physical torus coordinates
- `int PMI_torus2rank (int x, int y, int z);`
 - ❖ Translates a physical torus coordinates to an MPI rank

Outline

- Principles of BlueGene/L design
- Programming BlueGene/L with MPI
- **Case study – Jacobi relaxation**
- Conclusions

Jacobi relaxation

- At each time step, compute

$$U'_{i,j} = \frac{1}{4}(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1}) + F(i,j)$$

for all i and j

- If the array is block distributed on a two-dimensional processor grid, then each **processor** needs boundary data from its **north**, **south**, **east**, and **west** neighbors:

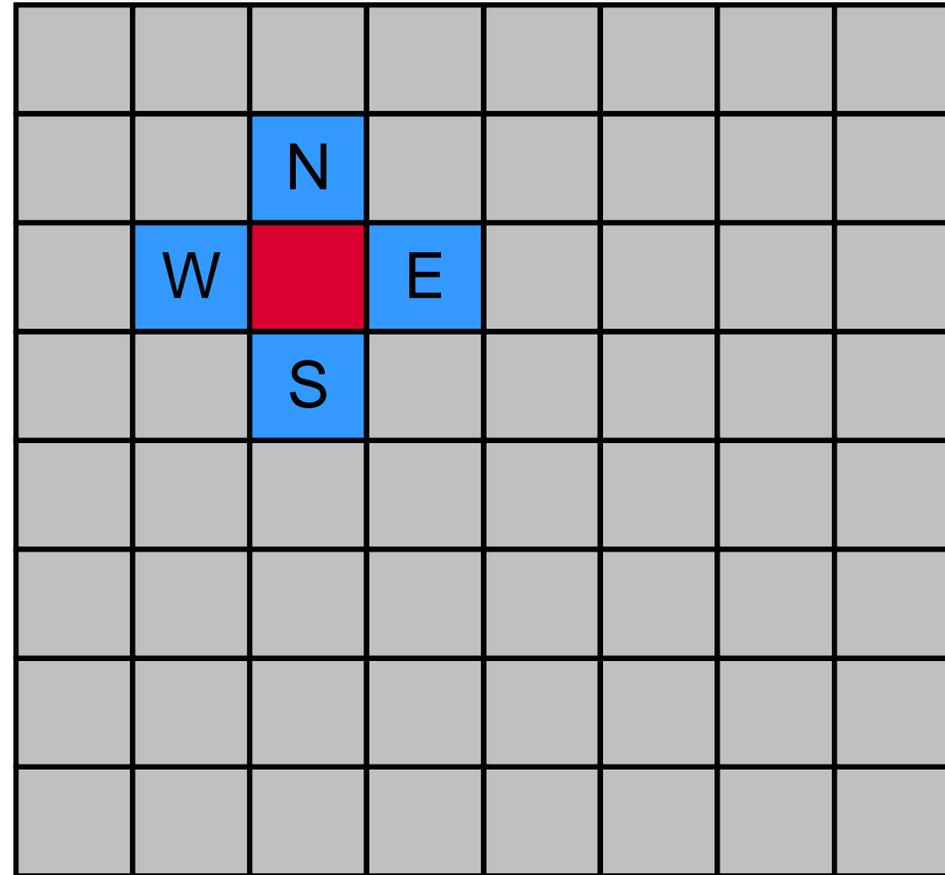
$$U(:,nx+1) \leftarrow U(:,1)[\text{east}]$$

$$U(:,0) \leftarrow U(:,nx)[\text{west}]$$

$$U(ny+1,:) \leftarrow U(1,:)[\text{south}]$$

$$U(0,:) \leftarrow U(nx,:)[\text{north}]$$

(plus the corner data)

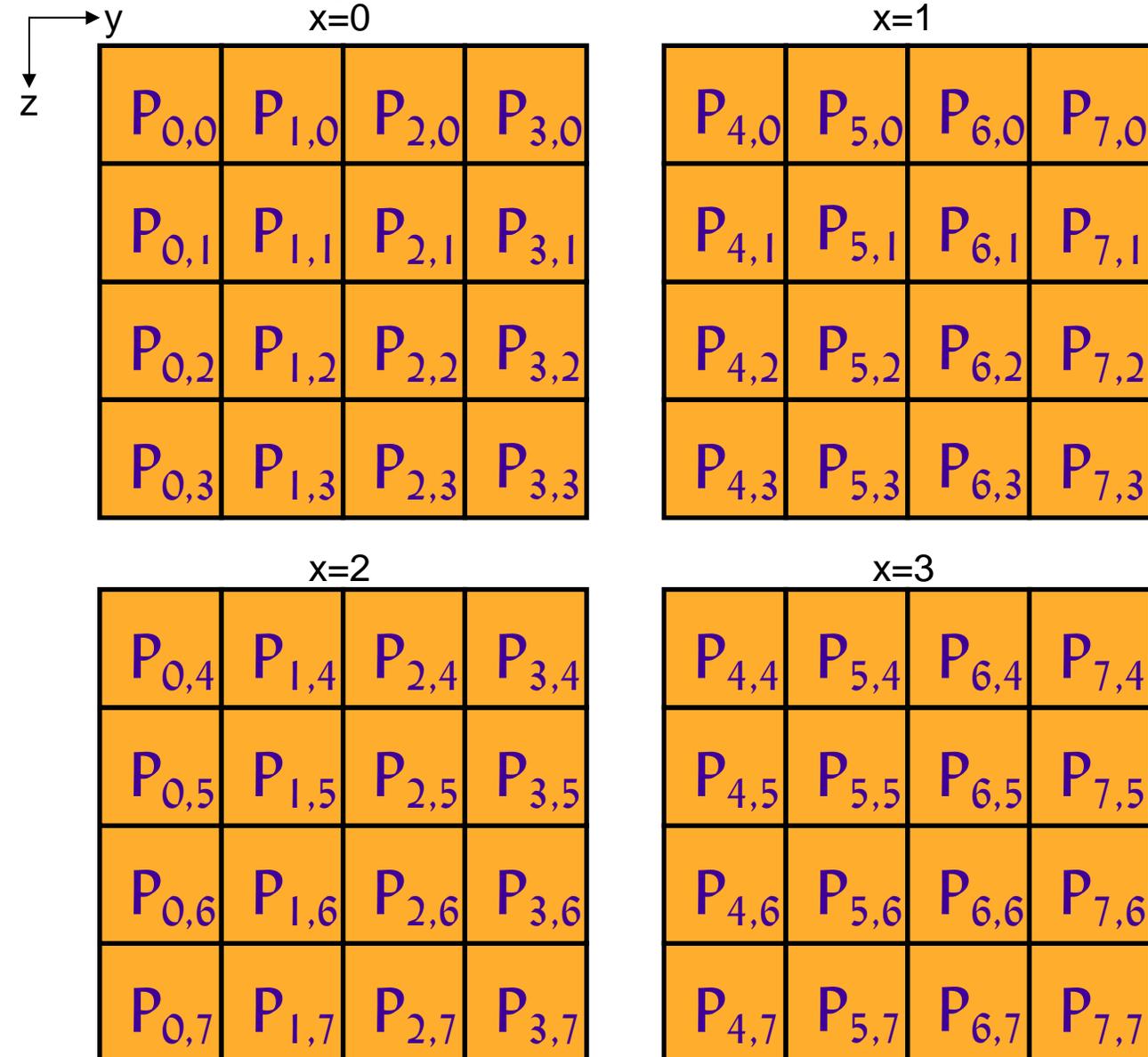


First issue: virtual to physical mapping

- We partition the array as an 8x8 array of blocks
- We execute the relaxation on 8x8 logical grid of processes
- The data is decomposed using a block distribution
- Mapping of processes to physical nodes can be controlled using the PMI functions “**rank2torus**” and “**torus2rank**”

$P_{0,0}$	$P_{1,0}$	$P_{2,0}$	$P_{3,0}$	$P_{4,0}$	$P_{5,0}$	$P_{6,0}$	$P_{7,0}$
$P_{0,1}$	$P_{1,1}$	$P_{2,1}$	$P_{3,1}$	$P_{4,1}$	$P_{5,1}$	$P_{6,1}$	$P_{7,1}$
$P_{0,2}$	$P_{1,2}$	$P_{2,2}$	$P_{3,2}$	$P_{4,2}$	$P_{5,2}$	$P_{6,2}$	$P_{7,2}$
$P_{0,3}$	$P_{1,3}$	$P_{2,3}$	$P_{3,3}$	$P_{4,3}$	$P_{5,3}$	$P_{6,3}$	$P_{7,3}$
$P_{0,4}$	$P_{1,4}$	$P_{2,4}$	$P_{3,4}$	$P_{4,4}$	$P_{5,4}$	$P_{6,4}$	$P_{7,4}$
$P_{0,5}$	$P_{1,5}$	$P_{2,5}$	$P_{3,5}$	$P_{4,5}$	$P_{5,5}$	$P_{6,5}$	$P_{7,5}$
$P_{0,6}$	$P_{1,6}$	$P_{2,6}$	$P_{3,6}$	$P_{4,6}$	$P_{5,6}$	$P_{6,6}$	$P_{7,6}$
$P_{0,7}$	$P_{1,7}$	$P_{2,7}$	$P_{3,7}$	$P_{4,7}$	$P_{5,7}$	$P_{6,7}$	$P_{7,7}$

A mapping on a 4x4x4 physical machine (torus)



Creating the mapping

```
for (p=0; p<nprocs; p++) {  
    int x, y, z;  
    PMI_rank2torus(p, &x, &y, &z);  
    rank2grid[p].x = y + (x%2)*4;  
    rank2grid[p].y = z + (x/2)*4;  
    grid2rank[rank2grid[p].x,rank2grid[p].y] = p;  
}
```

Second issue: multiple communication channels

■ Naïve code:

```
MPI_Irecv(U(:,nx+1), grid2rank[myx+1][myy], request_east);
MPI_Send(U(:,1), grid2rank[myx-1][myy]);
MPI_Wait(request_east, &status);
```

```
MPI_Irecv(U(:,0), grid2rank[myx-1][myy], request_west);
MPI_Send(U(:,nx), grid2rank[myx+1][myy]);
MPI_Wait(request_west, &status);
```

...

■ Better code:

```
MPI_Irecv(U(:,nx+1), grid2rank[myx+1][myy], request[0]);
MPI_Isend(U(:,1), grid2rank[myx-1][myy], request[1]);
MPI_Irecv(U(:,0), grid2rank[myx-1][myy], request[2]);
MPI_Isend(U(:,nx), grid2rank[myx+1][myy], request[3]);
```

...

```
MPI_Waitall(request[], status[]);
```

Outline

- Principles of BlueGene/L design
- Programming BlueGene/L with MPI
- Case study – Jacobi relaxation
- **Conclusions**

Conclusions

- Space sharing with dedicated processor/thread guarantees resources for application processes – fast and deterministic execution
- Standard programming languages + MPI offer familiar programming environment
 - ❖ Importance of mapping tasks to physical nodes
 - ❖ Importance of using multiple channels for communication
- Latency increases with distance between nodes, bandwidth subject to congestion
- PMI (Process Management Interface) offers interfaces for performing task to physical node mapping