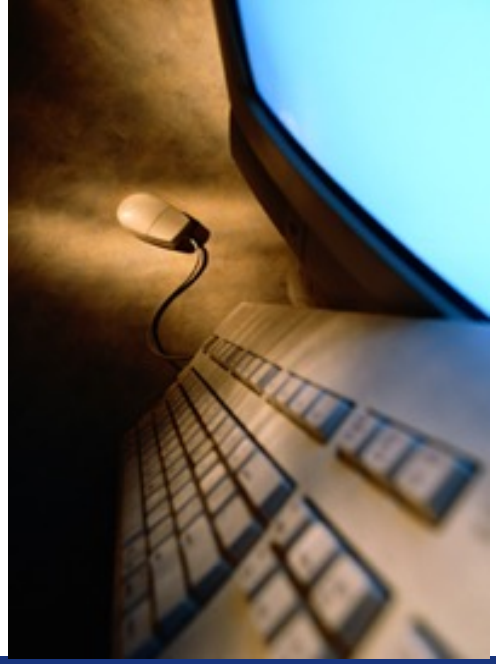
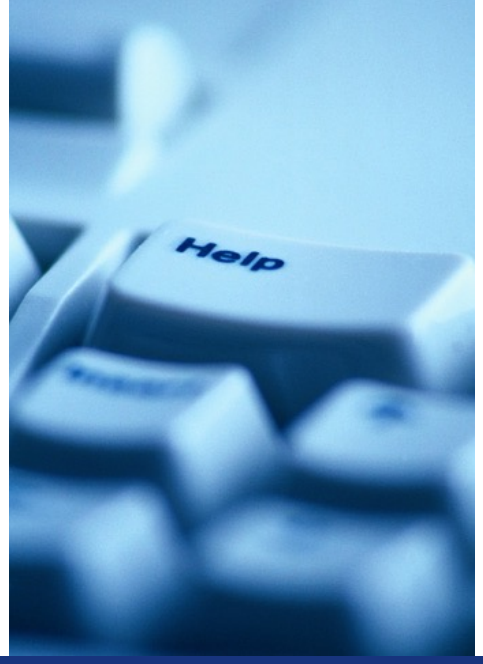
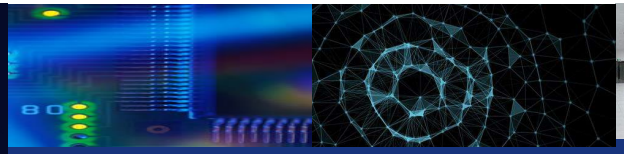



并行与分布式计算

Parallel & Distributed Computing

陈鹏飞
计算机学院
chenpf7@mail.sysu.edu.cn



Lecture 9 — Programming with MPI

- Matrix-Vector Multiplication and MPI + OpenMP

Pengfei Chen

School of Data and Computer Science

chenpf7@mail.sysu.edu.cn

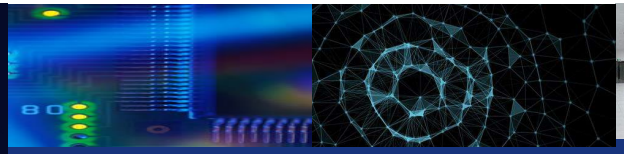
Outline:

- Example: matrix-vector multiplication
  - Sequential algorithm;
  - Design, analysis, and implementation of three parallel programs;
    - Rowwise block striped
    - Columnwise block striped
    - Checkerboard block (棋盘块)
- MPI + OpenMP

Sequential Algorithm

2	1	0	4	1	9
3	2	1	1	3	14
4	3	1	2	× 4	= 19
3	0	2	0	1	11

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

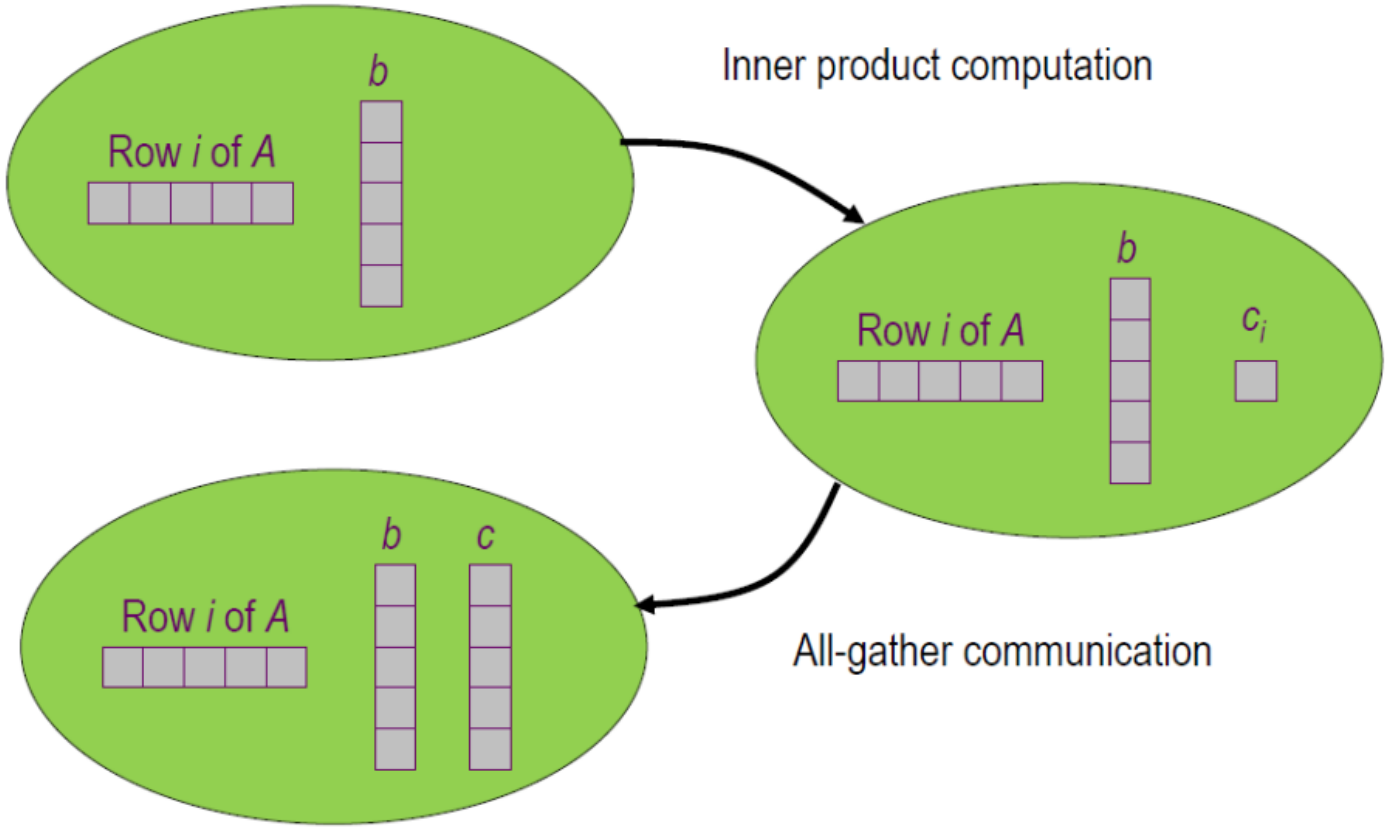


Rowwise Block Striped Matrix

- **Partitioning through domain decomposition**
- **Primitive task associated with**
  - **Row of matrix**
  - **Entire vector**

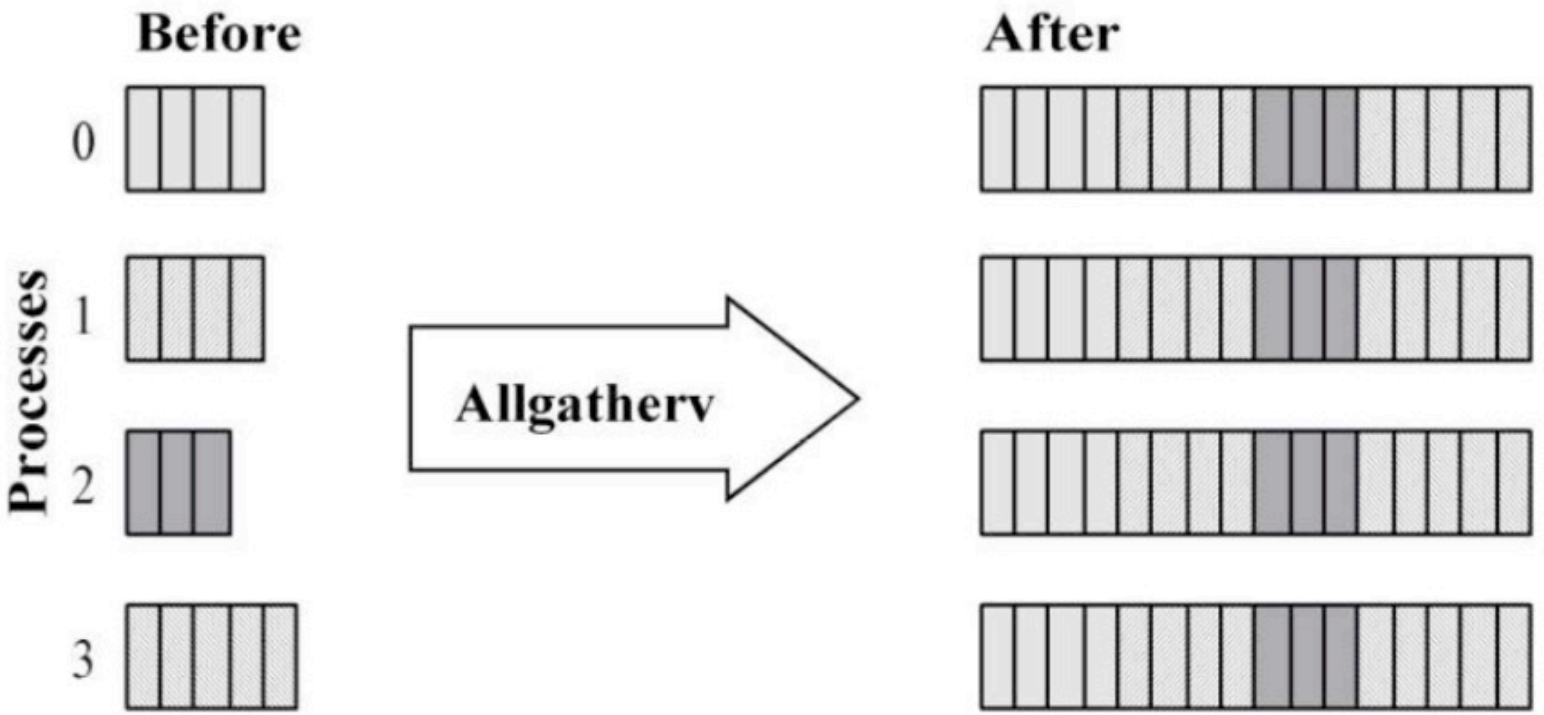
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Phases of Parallel Algorithm



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

MPI_Allgatherv



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

MPI_Allgatherv

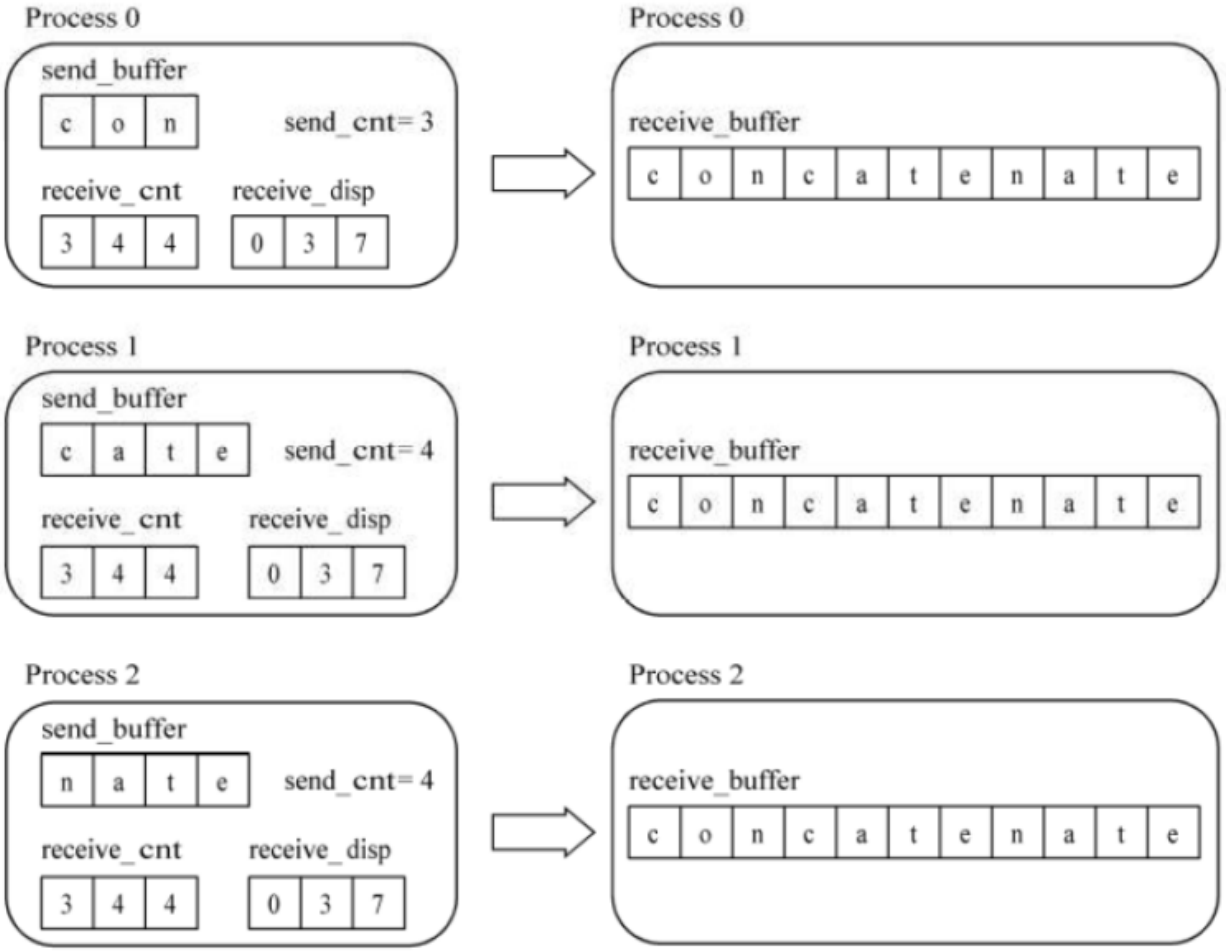
```
int MPI_Allgatherv (
void *send_buffer,
int send_cnt,
MPI_Datatype send_type,
void *receive_buffer,
int *receive_cnt,
int *receive_disp,
MPI_Datatype receive_type,
MPI_Comm communicator)
```

Use a simpler function ***MPI_Allgather(…)***

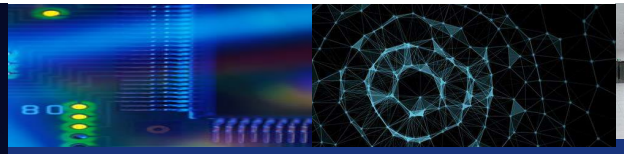
when the number of elements per processor is a constant

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

MPI_Allgatherv in Action



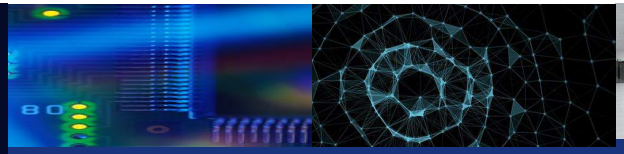
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Agglomeration and Mapping

- Static number of tasks
- Regular communication pattern (all-gather)
- Computation time per task is constant
- Strategy:
  - Agglomerate groups of rows
  - Create one task per MPI process

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

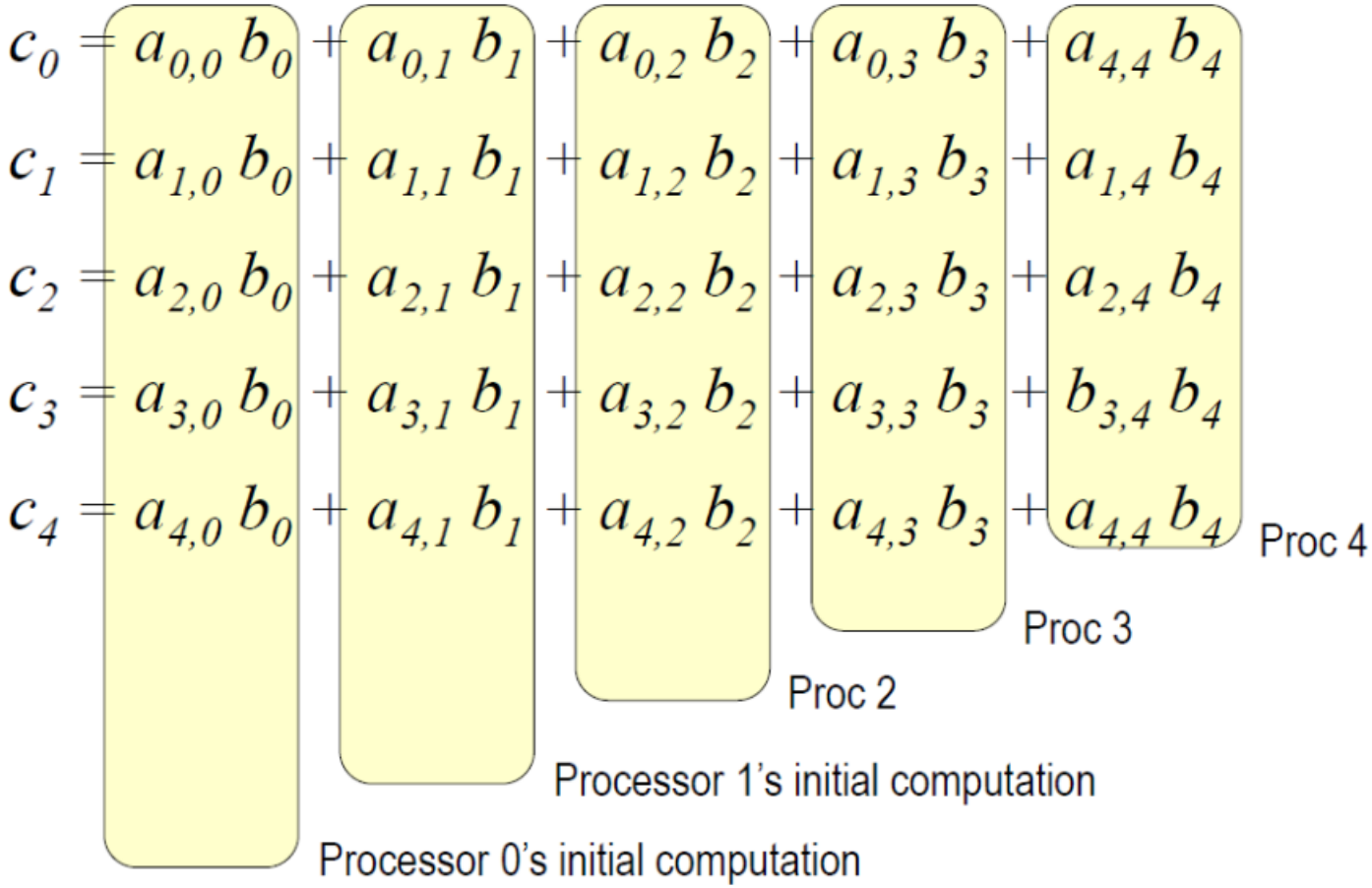


Columnwise Block Striped Matrix

- Partitioning through domain decomposition
- Task associated with
  - Column of matrix
  - Vector element

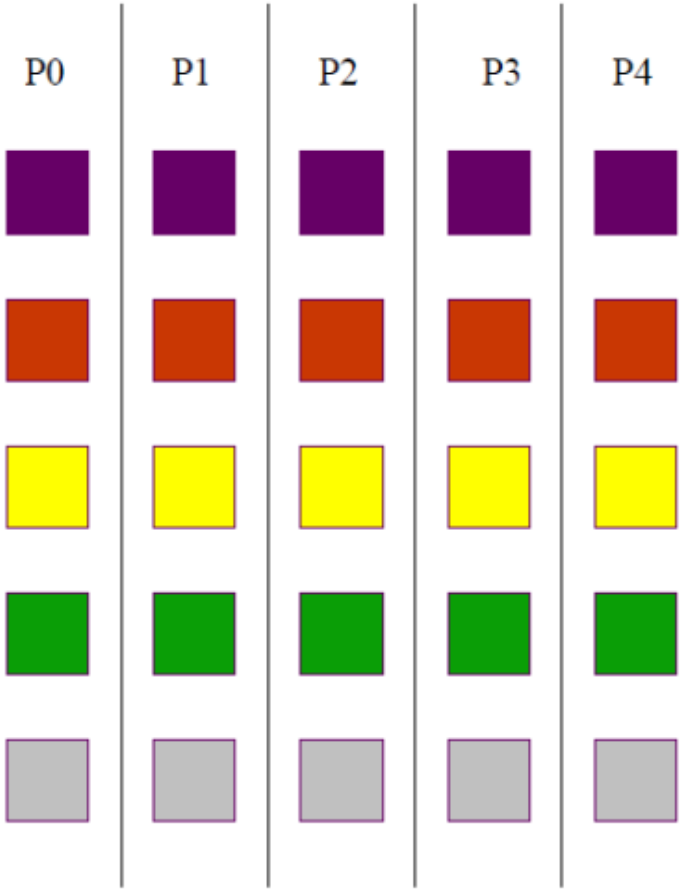
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Matrix-Vector Multiplication



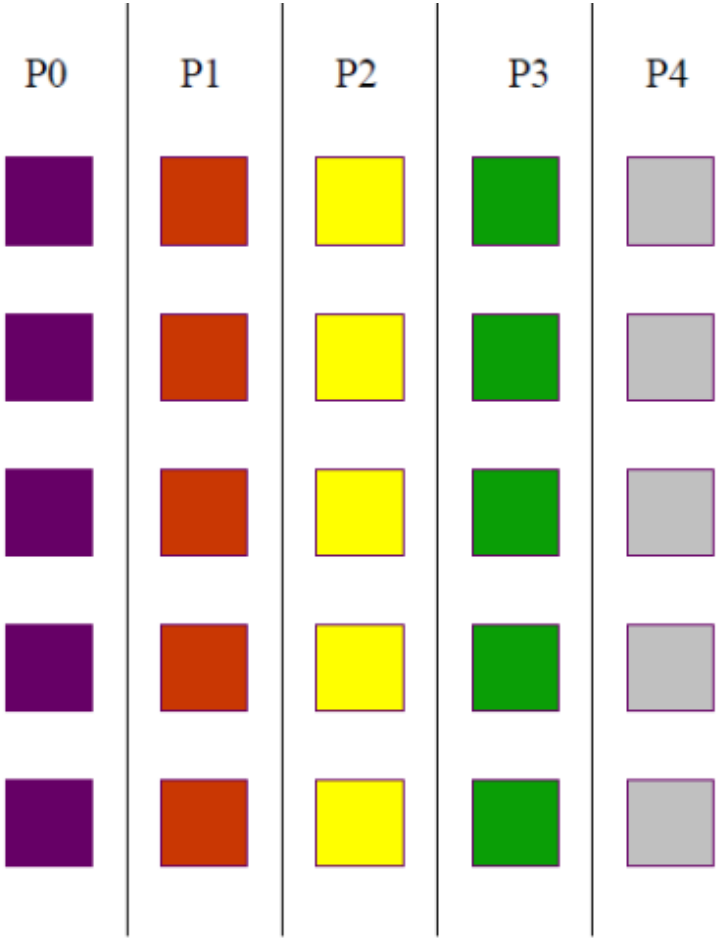
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

All-to-all Exchange (Before)



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

All-to-all Exchange (After)



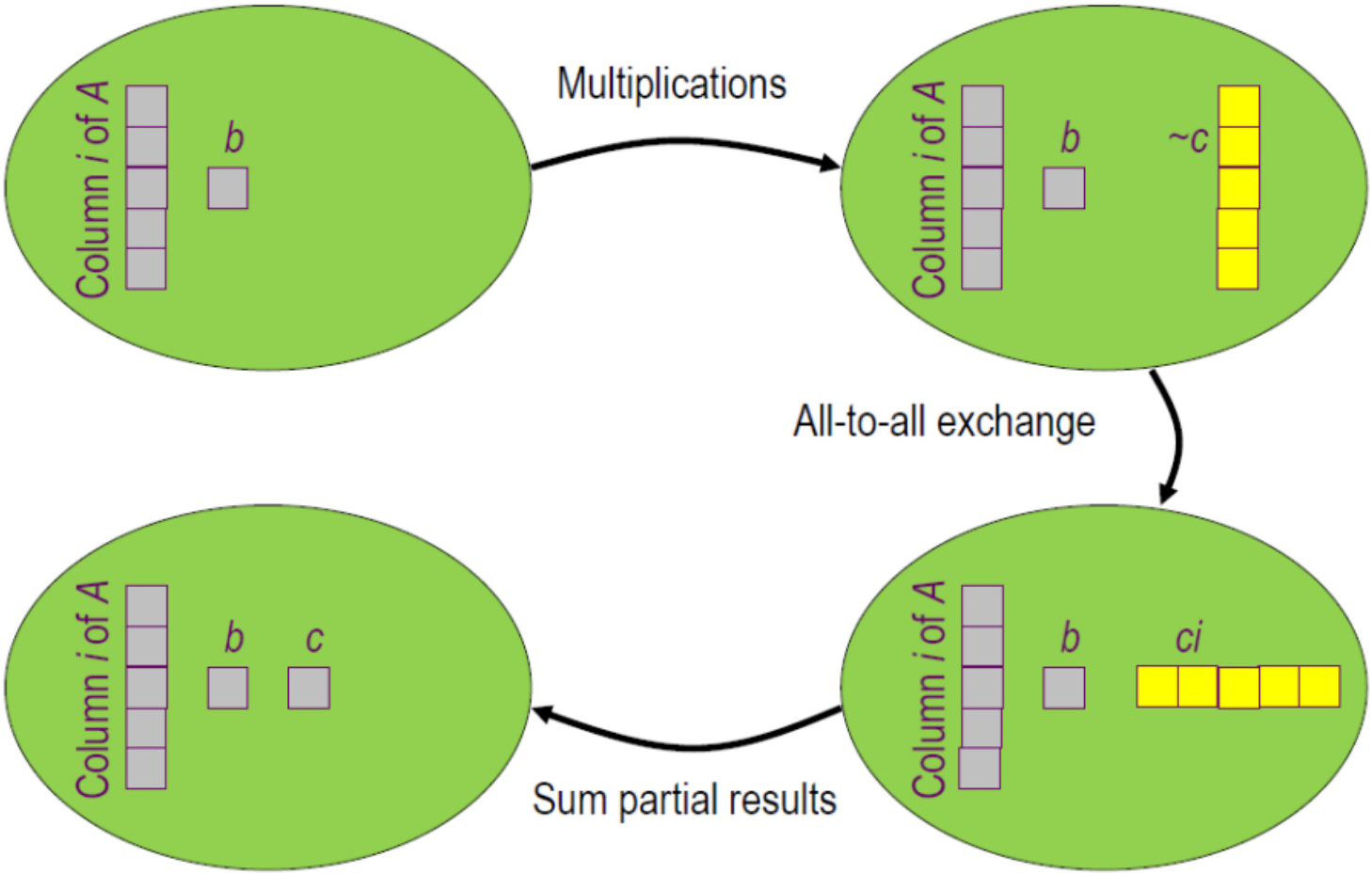
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

All-to-All Exchange (All-to-All Personalized Communication)

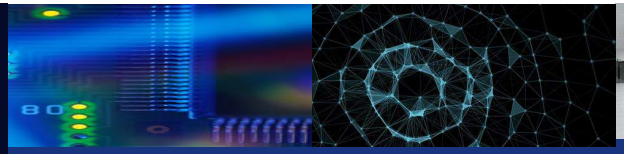
How to improve the communication efficiency?

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Phases of Parallel Algorithm



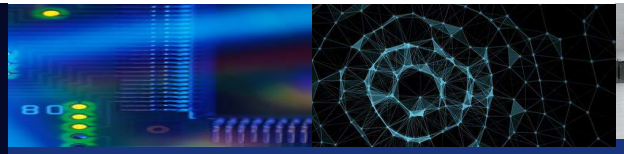
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Agglomeration and Mapping

- Static number of tasks
- Regular communication pattern (all-to-all)
- Computation time per task is constant
- Strategy:
  - Agglomerate groups of columns
  - Create one task per MPI process

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

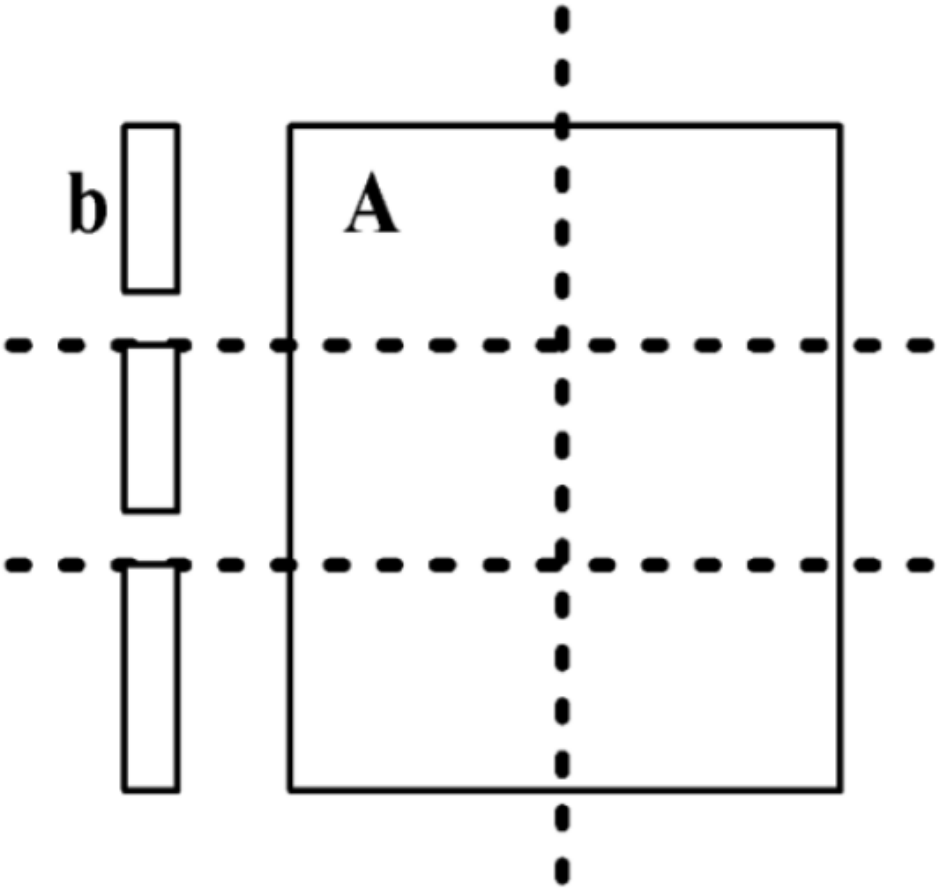


Checkerboard Block Decomposition

- **Associate primitive task with each element of the matrix *A***
- **Each primitive task performs one multiply**
- **Agglomerate primitive tasks into rectangular blocks**
- **Processes form a 2-D grid**
- **Vector *b* distributed by blocks among processes in first column of grid**

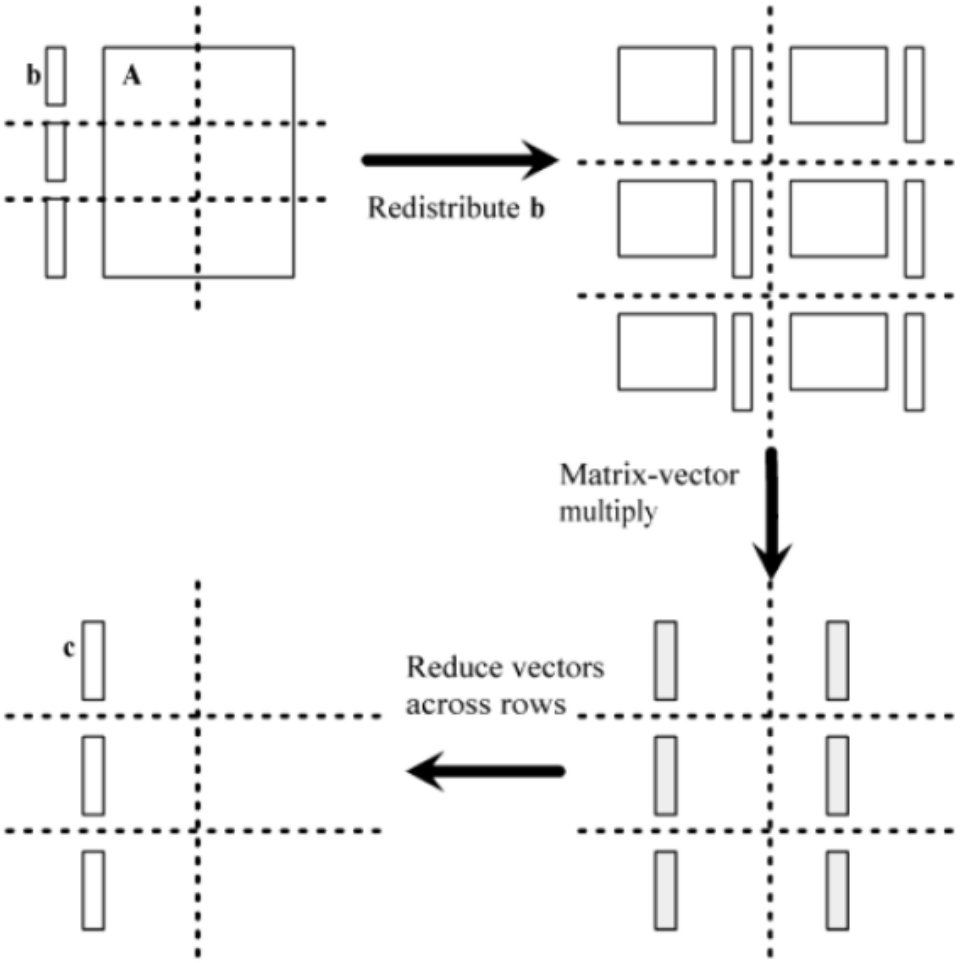
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Tasks after Agglomeration

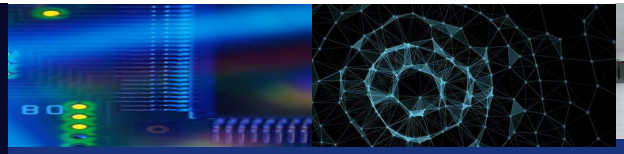


Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Algorithm's Phases



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

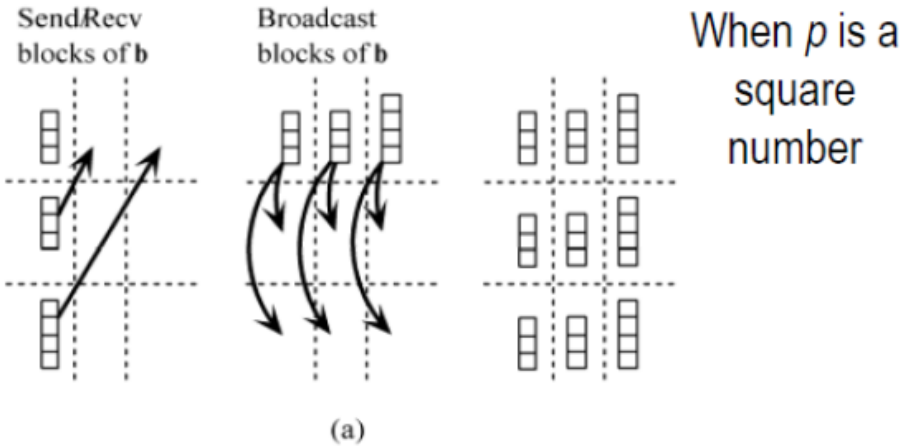


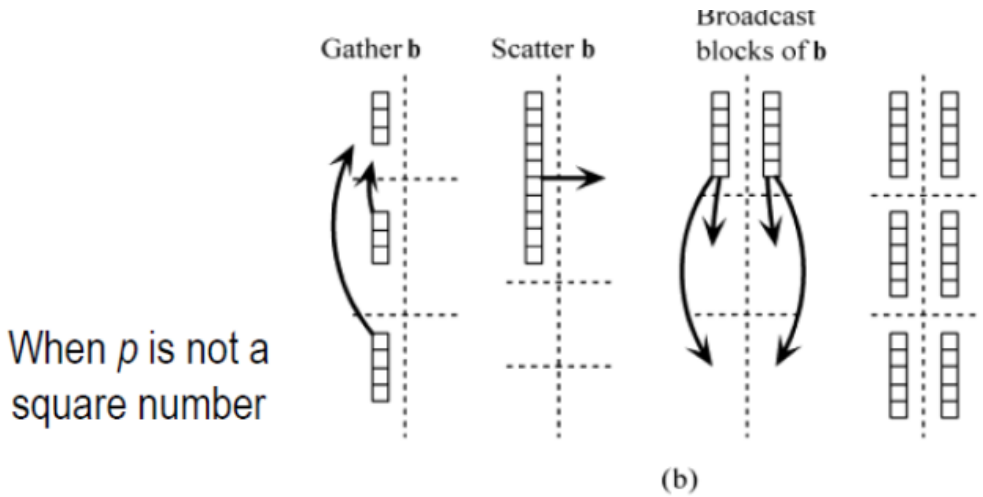
Redistributing Vector *b*

- **Step 1: Move *b* from processes in first row to processes in first column**
  - **If *p* square**
    - **First column/first row processes send/receive portions of *b***
  - **If *p* not square**
    - **Gather *b* on process 0, 0**
    - **Process 0, 0 broadcasts to first row procs**
- **Step 2: First row processes scatter *b* within columns**

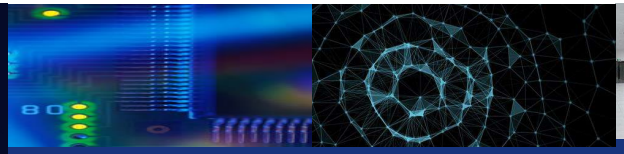
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Algorithm's Phases





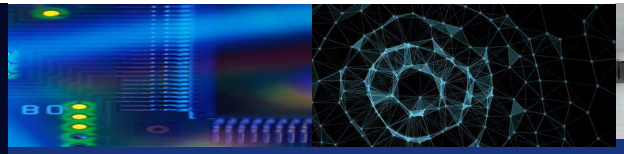
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Creating Communicators

- Want processes in a virtual 2-D grid
- Create a custom communicator to do this
- Collective communications involve all processes in a communicator
- We need to do broadcasts, reductions among subsets of processes
- We will create communicators for processes in same row or same column

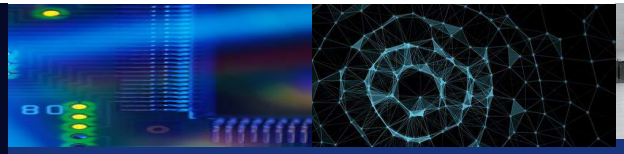
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



What's in a Communicator?

- **Process group**
- **Context**
- **Attributes**
  - **Topology (let's address processes another way)**
  - **Others we won't consider**

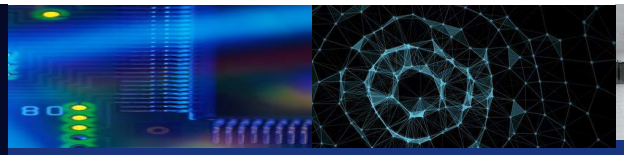
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Creating 2-D Virtual Grid of Processes

- *MPI_Dims_create*
  - **Input parameters**
    - **Total number of processes in desired grid**
    - **Number of grid dimensions**
  - **Returns number of processes in each dim**
- *MPI_Cart_create*
  - **Creates communicator with Cartesian topology (笛卡尔拓扑结构)**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



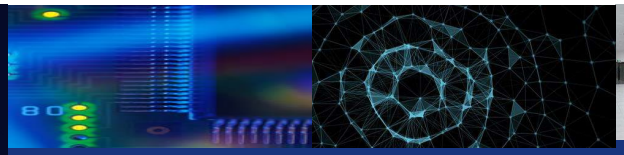
MPI_Dims_create

```
int MPI_Dims_create (
    int nodes,
        /* Input - Procs in grid */
    int dims,
        /* Input - Number of dims */
    int *size)
        /* Input/Output - Size of
           each grid dimension */
```

dims before call	function call	dims on return
(0,0)	MPI_DIMS_CREATE(6, 2, dims)	(3,2)
(0,0)	MPI_DIMS_CREATE(7, 2, dims)	(7,1)
(0,3,0)	MPI_DIMS_CREATE(6, 3, dims)	(2,3,1)
(0,3,0)	MPI_DIMS_CREATE(7, 3, dims)	erroneous call

source: DeinoMPI

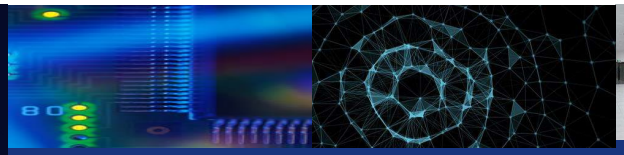
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



MPI_Cart_create （创建笛卡尔拓扑）

```
int MPI_Cart_create (
   MPI_Comm old_comm, /* Input - old communicator */

   int dims, /* Input - grid dimensions */

   int *size, /* Input - # procs in each dim */

   int *periodic,
       /* Input - periodic[j] is 1 if dimension j
          wraps around; 0 otherwise */

   int reorder,
       /* 1 if process ranks can be reordered */

   MPI_Comm *cart_comm)
       /* Output - new communicator */
```

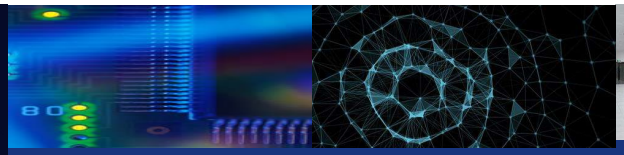
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Using MPI_Dims_create and MPI_Cart_create

```
MPI_Comm cart_comm;
int p;
int periodic[2];
int size[2];
...
size[0] = size[1] = 0;
MPI_Dims_create (p, 2, size);
periodic[0] = periodic[1] = 0;
MPI_Cart_create (MPI_COMM_WORLD, 2, size,
1, &cart_comm);
```

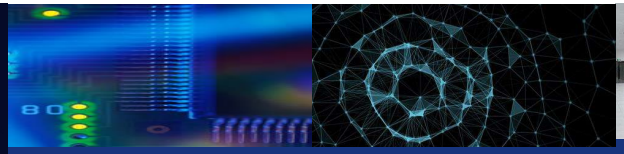
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Using MPI_Dims_create and MPI_Cart_create

```
MPI_Comm cart_comm;
int p;
int periodic[2];
int size[2];
...
size[0] = size[1] = 0;
MPI_Dims_create (p, 2, size);
periodic[0] = periodic[1] = 0;
MPI_Cart_create (MPI_COMM_WORLD, 2, size,
1, &cart_comm);
```

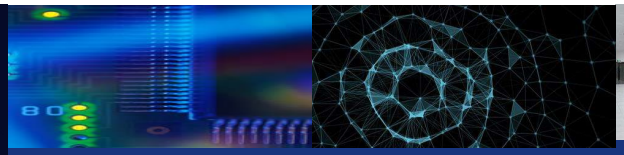
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Useful Grid-related Functions

- *MPI_Cart_rank*
  - **Given coordinates of process in Cartesian communicator, returns process rank**
- *MPI_Cart_cords*
  - **Given rank of process in Cartesian communicator , returns process' coordinates**

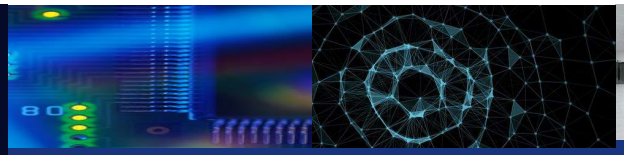
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Header for MPI_Cart_rank

```
int MPI_Cart_rank (
    MPI_Comm comm,
        /* In - Communicator */
    int *coords,
        /* In - Array containing process'
                grid location */
    int *rank)
        /* Out - Rank of process at
           specified coords */
```

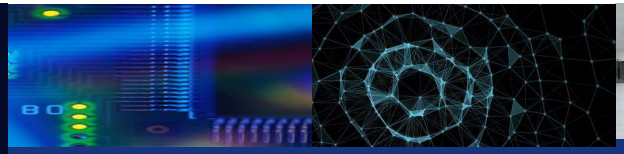
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Header for MPI_Cart_coords

```
int MPI_Cart_coords (
    MPI_Comm comm,
        /* In - Communicator */
    int rank,
        /* In - Rank of process */
    int dims,
        /* In - Dimensions in virtual grid
*/
    int *coords)
        /* Out - Coordinates of specified
           process in virtual grid */
```

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

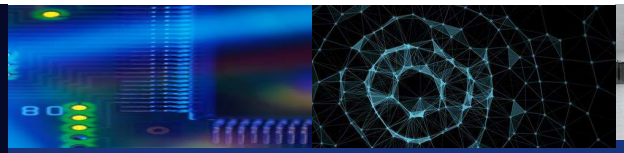


Collective Communications

	Ring	2-D Mesh	Hypercube
One-to-All Broadcast			
All-to-One Reduction			
All-to-All Broadcast			
All-to-All Reduction			
Scatter			
Gather			
All-to-All Personalized Communication			

重点和精华

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

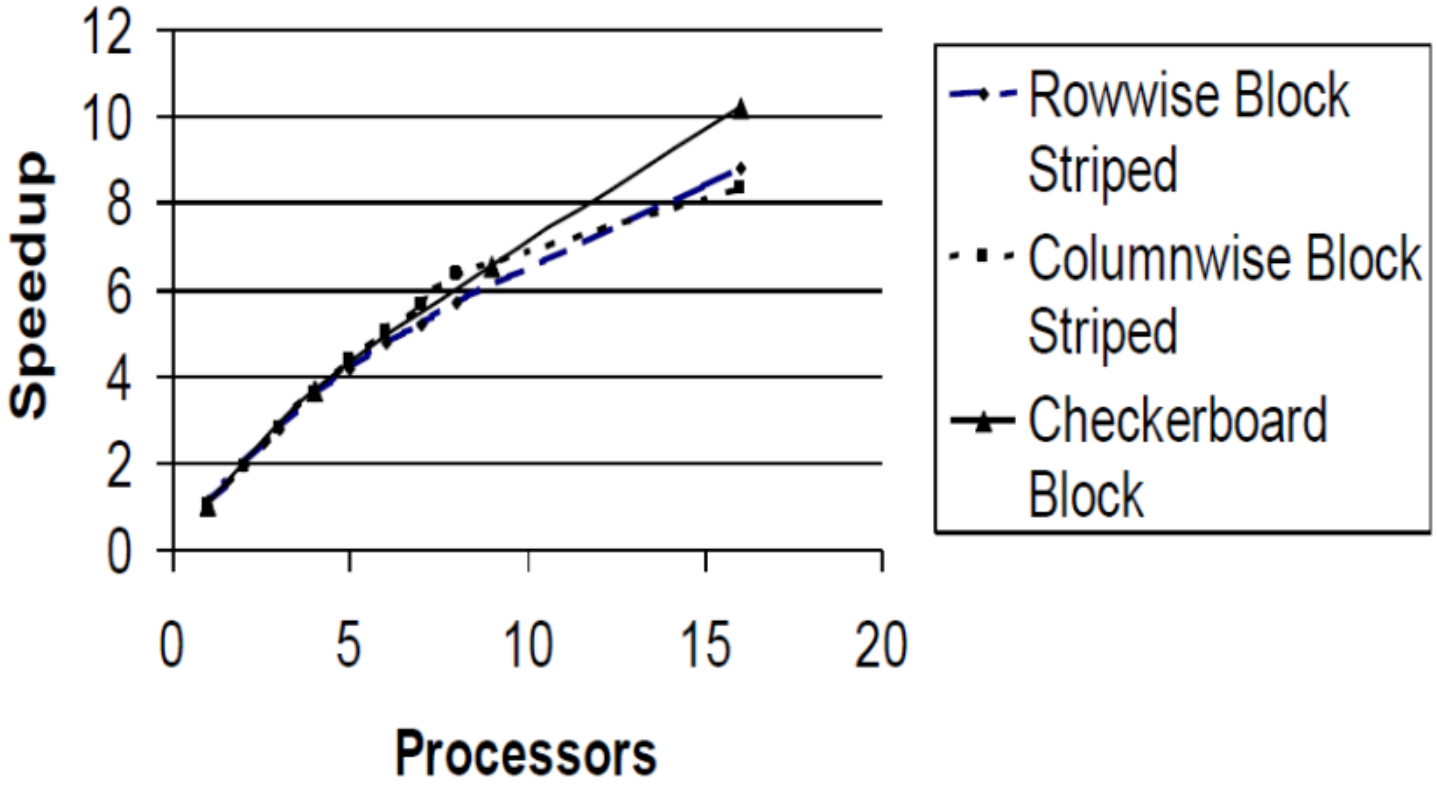


MPI Names of the Various Operations

Operation	MPI Name
One-to-all broadcast	MPI_Bcast
All-to-one reduction	MPI_Reduce
All-to-all broadcast	MPI_Allgather
All-to-all reduction	MPI_Reduce_scatter
All-reduce	MPI_Allreduce
Gather	MPI_Gather
Scatter	MPI_Scatter
All-to-all personalized	MPI_Alltoall

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

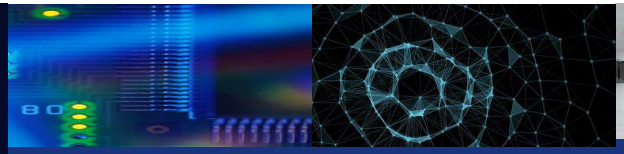
Comparison of Three Algorithms



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Programming with MPI

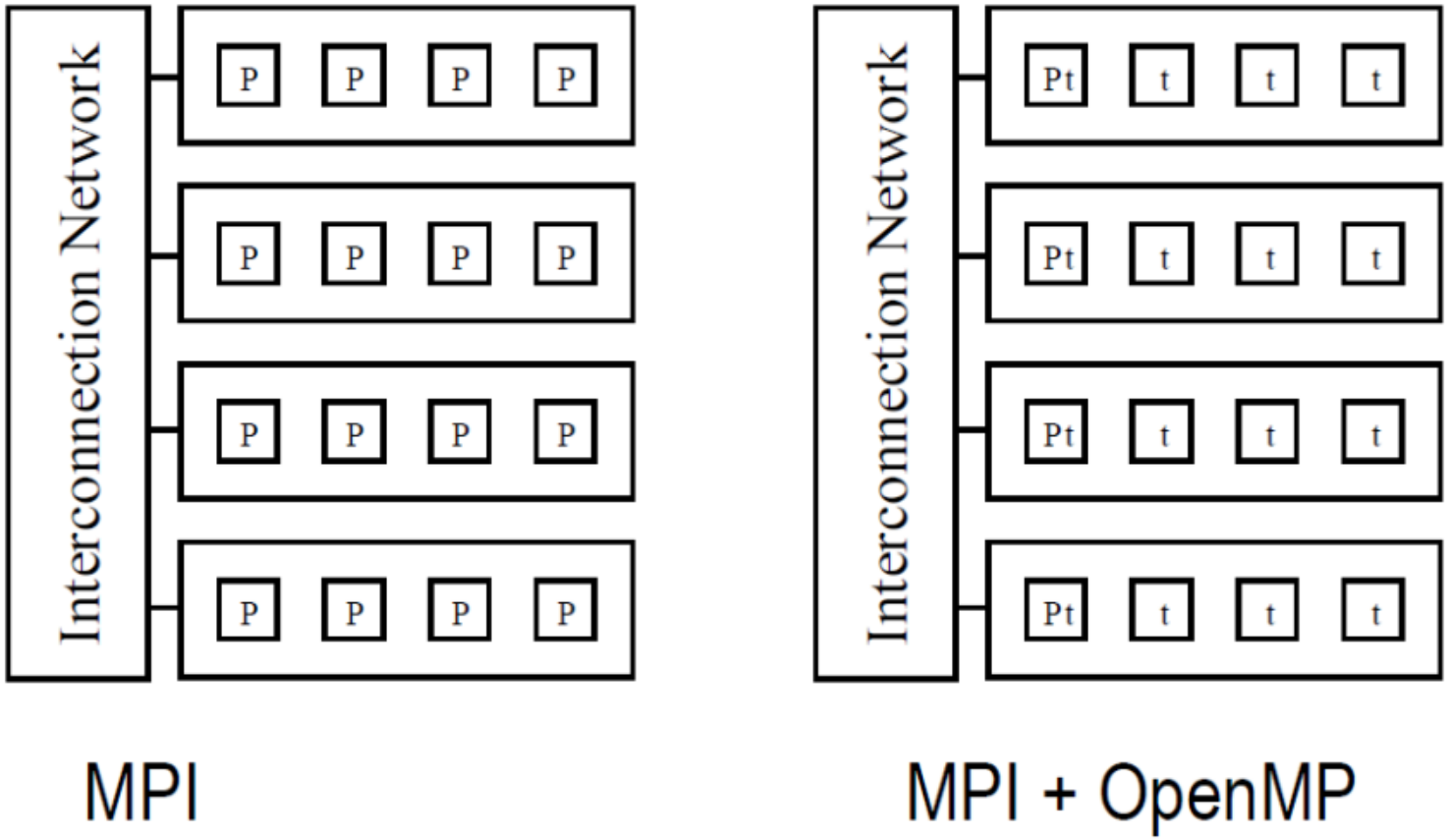
MPI + OpenMP



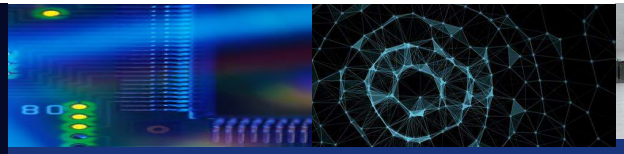
MPI + OpenMP

- **Advantages of using both MPI and OpenMP**
  - **Case Study: Jacobi method**
    - **An iterative methods to solve linear systems**

MPI vs. MPI + OpenMP



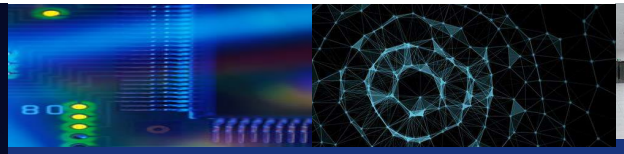
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Why MPI + OpenMP Can Execute Faster?

- Lower communication overhead
  - Message passing with *mk* processes, versus
  - Message passing with *m* processes with *k* threads each
- More portions of program may be practical to parallelize
- May allow more overlap of communications with computations
  - For example, if 3 MPI processes are waiting for messages, and 1 MPI process is active, it is worthwhile to fork some threads to speedup the 4th process

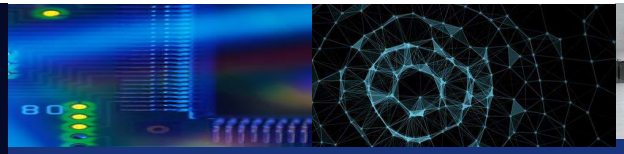
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Hybrid Light-Weight Threads & Heavier-Weight Processes

- For example, a serial program runs in 100s
  - S: 5s is inherent sequential
  - P_1: 5s are parallelizable but not worth message passing
  - P_2: 90s are perfectly parallelizable
- MPI-only with 16 processes
  - Replicate the P_1 part of program on all processes
  - Speedup = 1 / (0.10 + 0.90/16) = 6.4
- Hybrid MPI & threads
  - Execute the replicated P_1 with 2 threads
  - Speedup = 1 / (0.05 + 0.05/2 + 0.90/16) = 7.6
  - 19% faster than MPI-only

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Code for matrix_vector_product

```
void matrix_vector_product (int id, int p,
    int n, double **a, double *b, double *c)
{
    int    i, j;
    double tmp;        /* Accumulates sum */
    for (i=0; i<BLOCK_SIZE(id,p,n); i++) {
        tmp = 0.0;
        for (j = 0; j < n; j++)
            tmp += a[i][j] * b[j];
        piece[i] = tmp;
    }
    new_replicate_block_vector (id, p,
        piece, n, (void *) c, MPI_DOUBLE);
}
```

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Adding OpenMP Directives

- **Want to minimize fork/join overhead by making parallel the outermost possible loop**
- **Outer loop may be executed in parallel if each thread has a private copy of *tmp* and *j***

```
#pragma omp parallel for private(j,tmp)
  for (i=0; i<BLOCK_SIZE(id,p,n); i++) {
```

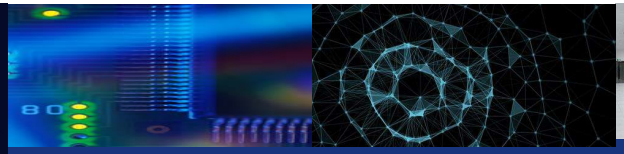
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

User Control of Threads

- **Want to give user opportunity to specify number of active threads per process**
- **Add a call to *omp_set_num_threads* to function main**
- **Argument comes from command line**

```
omp_set_num_threads (atoi(argv[3]));
```

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



What Happened?

- **We transformed a MPI program to a MPI+OpenMP program by adding only two lines to our program!**

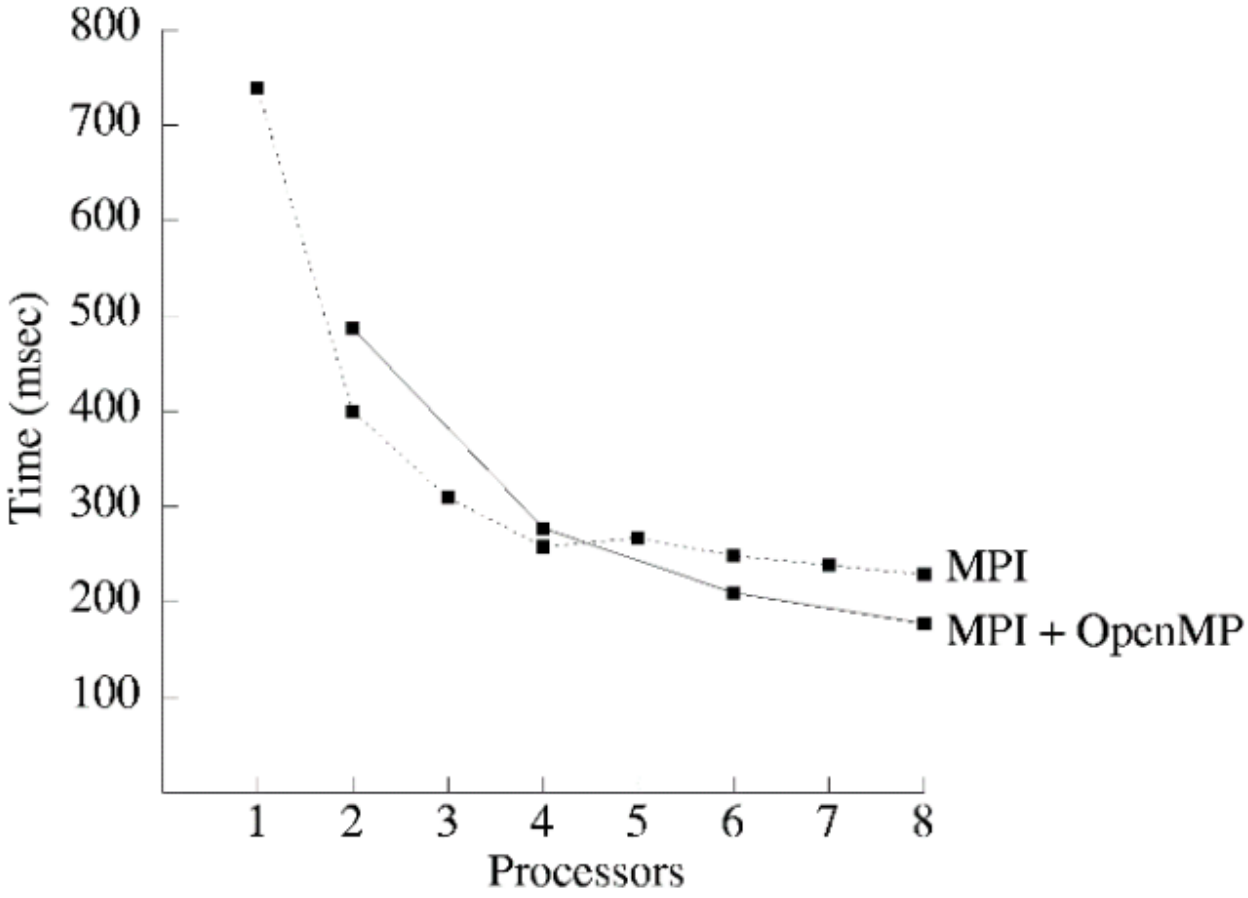
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Benchmarking

- Target system: a commodity cluster with four dual processor nodes
- MPI program executes on 1, 2, ..., 8 CPUs
  - On 1, 2, 3, 4 CPUs, each process on different node, maximizing memory bandwidth per CPU
- MPI+OpenMP program executes on 1, 2, 3, 4 processes
  - Each process has two threads
  - C+MPI+OpenMP program executes on 2, 4, 6, 8 threads

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Results of Benchmarking

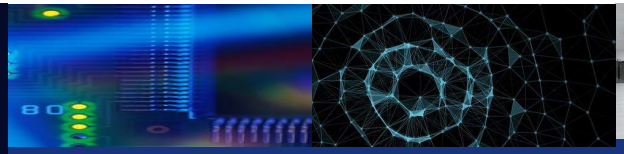


Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Analysis of Results

- **MPI+OpenMP program slower on 2, 4 CPUs because MPI+OpenMP threads are sharing memory bandwidth, while C+MPI processes are not**
- **MPI+OpenMP programs faster on 6, 8 CPUs because they have lower communication cost**

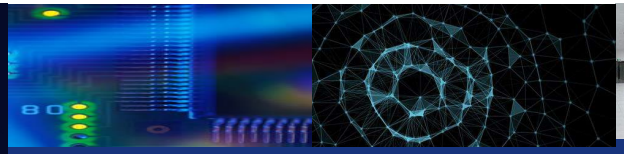
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Case Study: Jacobi Method

- **Begin with MPI program that uses Jacobi method to solve steady state heat distribution problem**
- **Program based on rowwise block striped decomposition of two-dimensional matrix containing finite difference mesh (有限差分网格，FDM)**

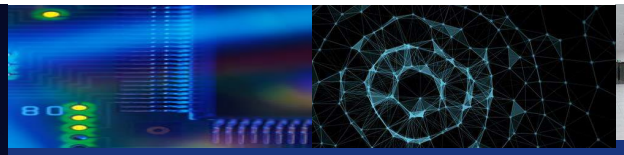
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Gaussian Elimination (高斯消元)

- **Used to solve $Ax = b$ when A is dense**
- **Reduces $Ax = b$ to upper triangular system $Tx = c$**
- **Back substitution can then solve $Tx = c$ for x**

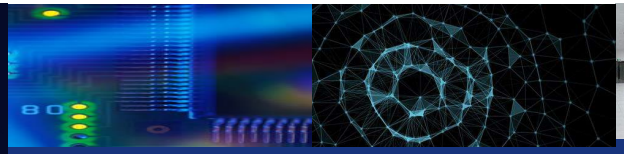
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Gaussian Elimination Example (1/4)

$$
\begin{aligned}
4x_0 + 6x_1 + 2x_2 - 2x_3 &= 8 \\
2x_0 + 5x_2 - 2x_3 &= 4 \\
-4x_0 - 3x_1 - 5x_2 + 4x_3 &= 1 \\
8x_0 + 18x_1 - 2x_2 + 3x_3 &= 40
\end{aligned}
$$

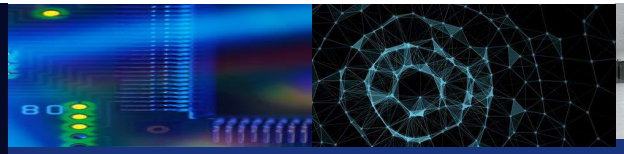
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Gaussian Elimination Example (2/4)

$$
\begin{aligned}
4x_0 + 6x_1 + 2x_2 - 2x_3 &= 8 \\
-3x_1 + 4x_2 - 1x_3 &= 0 \\
+3x_1 - 3x_2 + 2x_3 &= 9 \\
+6x_1 - 6x_2 + 7x_3 &= 24
\end{aligned}
$$

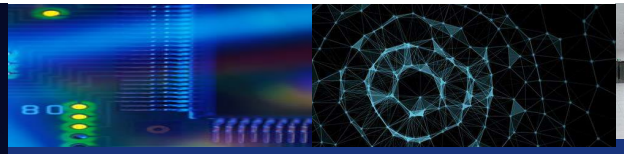
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Gaussian Elimination Example (3/4)

$$
\begin{aligned}
4x_0 \quad +6x_1 \quad +2x_2 \quad -2x_3 \quad &= \quad 8 \\
-3x_1 \quad +4x_2 \quad -1x_3 \quad &= \quad 0 \\
1x_2 \quad +1x_3 \quad &= \quad 9 \\
2x_2 \quad +5x_3 \quad &= \quad 24
\end{aligned}
$$

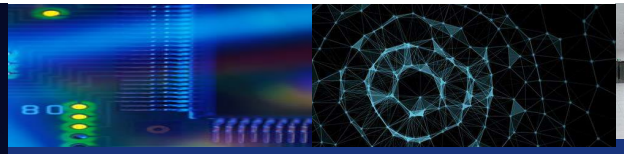
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Gaussian Elimination Example (4/4)

$$
\begin{aligned}
4x_0 \quad +6x_1 \quad +2x_2 \quad -2x_3 &= 8 \\
-3x_1 \quad +4x_2 \quad -1x_3 &= 0 \\
1x_2 \quad +1x_3 &= 9 \\
3x_3 &= 6
\end{aligned}
$$

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

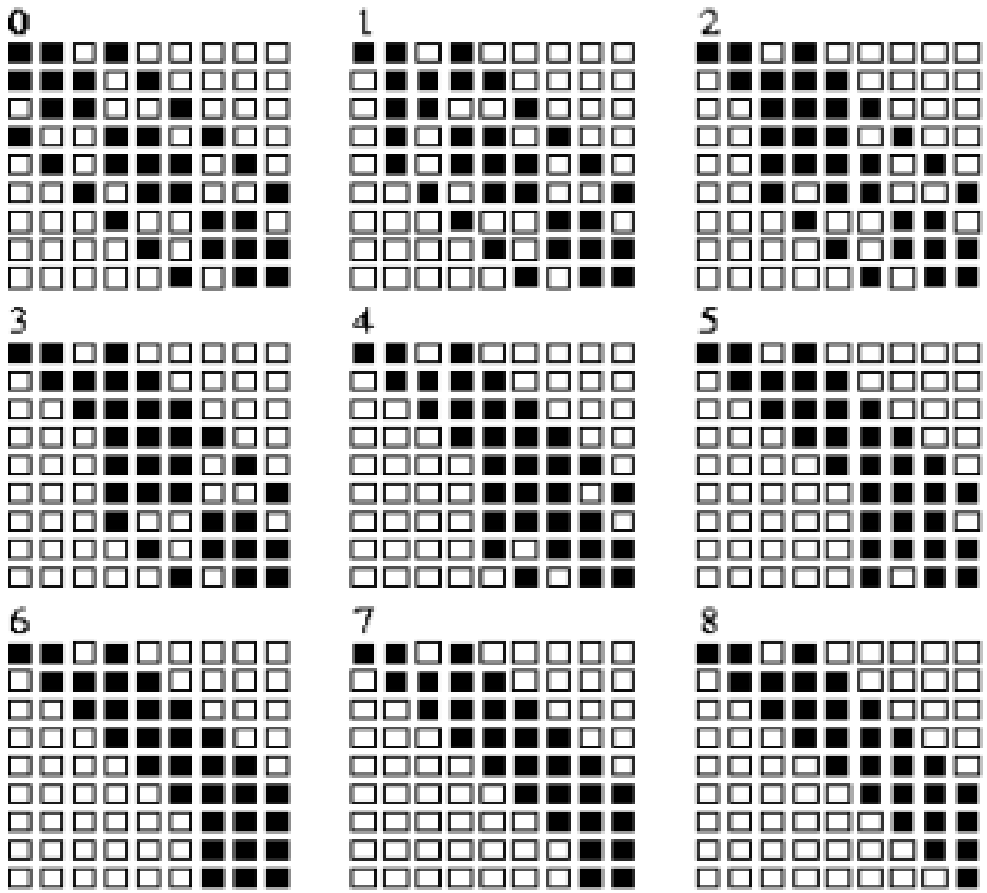


Sparse Systems

- **Gaussian elimination not well-suited for sparse systems**
- **Coefficient matrix gradually fills with nonzero elements**
  - **Increases storage requirements**
  - **Increases total operation count**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Example of "Fill" in Gaussian Elimination

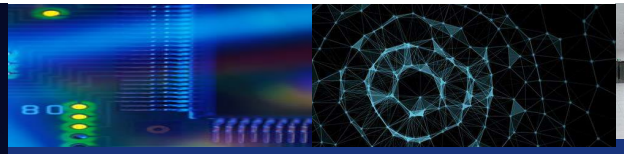


Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Iterative Methods

- **Iterative method: algorithm that generates a series of approximations to solution's value**
- **Require less storage than direct methods**
- **Since they avoid computations on zero elements, they can save a lot of computations**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Jacobi Method

Let

$$A\mathbf{x} = \mathbf{b}$$

be a square system of n linear equations, where:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Then A can be decomposed into a diagonal component D, and the remainder R:

$$A = D + R \quad \text{where} \quad D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \text{ and } R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}.$$

The solution is then obtained iteratively via

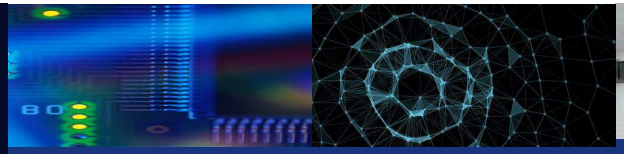
$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - R\mathbf{x}^{(k)}),$$

where $\mathbf{x}^{(k)}$ is the kth approximation or iteration of $\mathbf{x}$ and $\mathbf{x}^{(k+1)}$ is the next or $k + 1$ iteration of $\mathbf{x}$. The element-based formula is thus:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

Values of elements of vector x at iteration k+1 depend upon values of vector x at iteration k

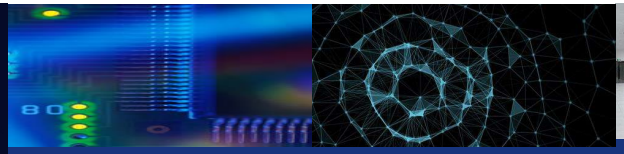
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Rate of Convergence

- **Even when Jacobi method, rate of convergence often too slow to make them practical**
- **We will move on to an iterative method with much faster convergence**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Methodology

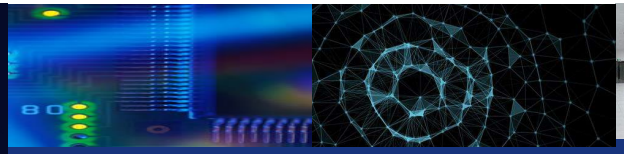
- **Profile execution of MPI program**
- **Focus on adding OpenMP directives to most compute-intensive function**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Summary

- **Hybrid C+MPI+OpenMP program uniformly faster than C+MPI program**
- **Computation/communication ratio of hybrid program is superior**
- **Number of mesh points per element communicated is twice as high per node for the hybrid program**
- **Lower communication overhead leads to 19% better speedup on 8 CPUs**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.



Thank You !