Matrix-Vector Multiplication

Parallel and Distributed Computing

Department of Computer Science and Engineering (DEI)
Instituto Superior Técnico

November 13, 2012

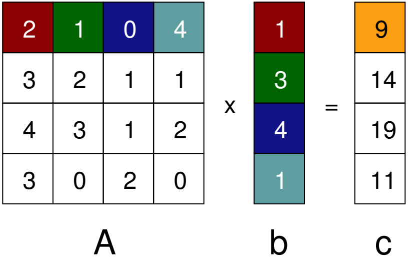
Outline

- Matrix-vector multiplication
  - rowwise decomposition
  - columnwise decomposition
  - checkerboard decomposition

- Gather, scatter, alltoall

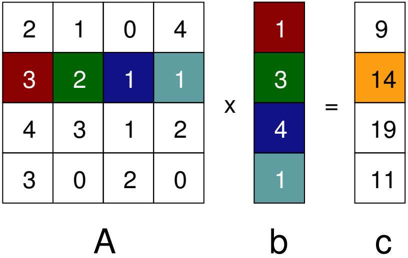
- Grid-oriented communications

Matrix-Vector Multiplication



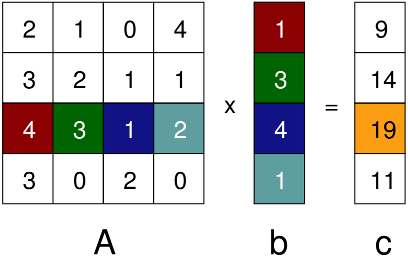
$$\begin{bmatrix}2&1&0&4\\3&2&1&1\\4&3&1&2\\3&0&2&0\end{bmatrix} \text{x} \begin{bmatrix}1\\3\\4\\1\end{bmatrix} = \begin{bmatrix}9\\14\\19\\11\end{bmatrix}$$

Matrix-Vector Multiplication



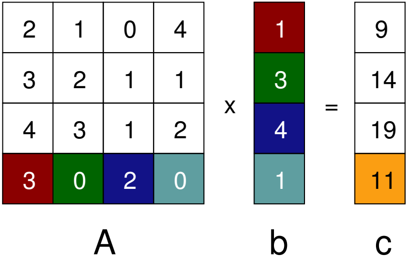
$$\underset{A}{\begin{bmatrix} 2 & 1 & 0 & 4 \\ 3 & 2 & 1 & 1 \\ 4 & 3 & 1 & 2 \\ 3 & 0 & 2 & 0 \end{bmatrix}} \times \underset{b}{\begin{bmatrix} 1 \\ 3 \\ 4 \\ 1 \end{bmatrix}} = \underset{c}{\begin{bmatrix} 9 \\ 14 \\ 19 \\ 11 \end{bmatrix}}$$

Matrix-Vector Multiplication



$$\begin{bmatrix} 2 & 1 & 0 & 4 \\ 3 & 2 & 1 & 1 \\ 4 & 3 & 1 & 2 \\ 3 & 0 & 2 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 14 \\ 19 \\ 11 \end{bmatrix}$$

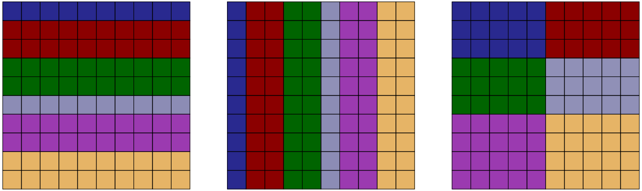
Matrix-Vector Multiplication



$$\underbrace{\begin{bmatrix} 2 & 1 & 0 & 4 \\ 3 & 2 & 1 & 1 \\ 4 & 3 & 1 & 2 \\ 3 & 0 & 2 & 0 \end{bmatrix}}_{A} \text{ x } \underbrace{\begin{bmatrix} 1 \\ 3 \\ 4 \\ 1 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} 9 \\ 14 \\ 19 \\ 11 \end{bmatrix}}_{c}$$

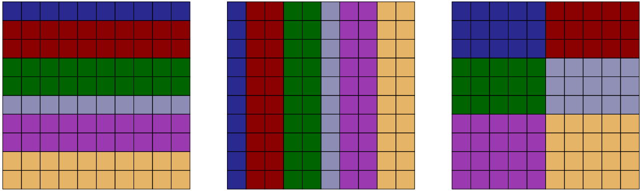
Matrix Decomposition

- rowwise decomposition
- columnwise decomposition
- checkered-board decomposition



Matrix Decomposition

- rowwise decomposition
- columnwise decomposition
- checkered-board decomposition

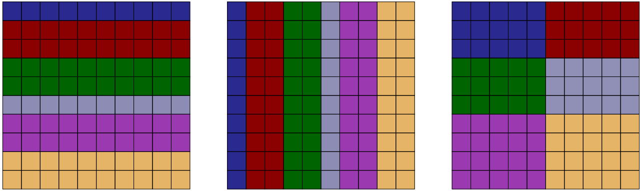


Storing vectors:

- Divide vector elements among processes
- Replicate vector elements

Matrix Decomposition

- rowwise decomposition
- columnwise decomposition
- checkered-board decomposition



Storing vectors:

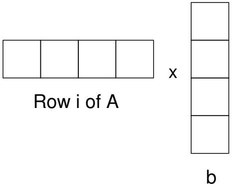
- Divide vector elements among processes
- Replicate vector elements

Vector replication acceptable because vectors have only n elements, versus n^2 elements in matrices.

Rowwise Decomposition

Task associated with

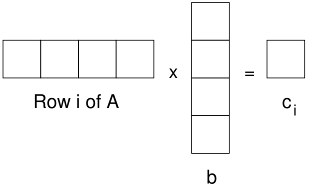
- row of matrix
- entire vector



Rowwise Decomposition

Task associated with

- row of matrix
- entire vector



Rowwise Decomposition

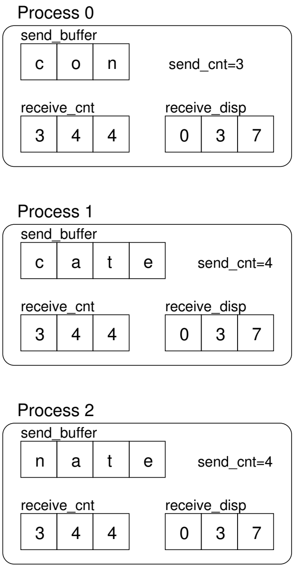
Task associated with

- row of matrix
- entire vector

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
)
```

MPI_Allgatherv



Process 0

send_buffer

c	o	n

send_cnt=3

receive_cnt

3	4	4

receive_disp

0	3	7

Process 1

send_buffer

c	a	t	e

send_cnt=4

receive_cnt

3	4	4

receive_disp

0	3	7

Process 2

send_buffer

n	a	t	e

send_cnt=4

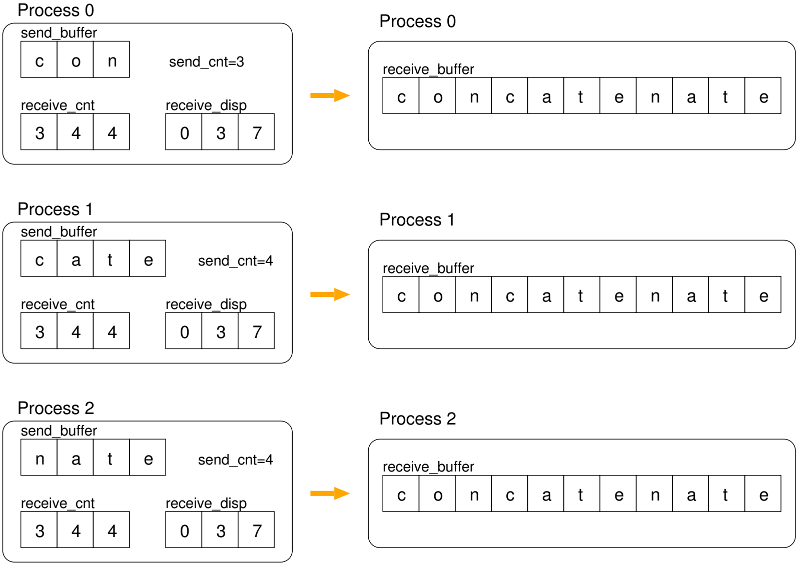
receive_cnt

3	4	4

receive_disp

0	3	7

MPI_Allgatherv



Process 0

send_buffer

c	o	n

send_cnt=3

receive_cnt

3	4	4

receive_disp

0	3	7

Process 0

receive_buffer

c	o	n	c	a	t	e	n	a	t	e

Process 1

send_buffer

c	a	t	e

send_cnt=4

receive_cnt

3	4	4

receive_disp

0	3	7

Process 1

receive_buffer

c	o	n	c	a	t	e	n	a	t	e

Process 2

send_buffer

n	a	t	e

send_cnt=4

receive_cnt

3	4	4

receive_disp

0	3	7

Process 2

receive_buffer

c	o	n	c	a	t	e	n	a	t	e

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$

Communication complexity of all-gather:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$

Communication complexity of all-gather: $\Theta(\log p + n)$

Overall complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$

Communication complexity of all-gather: $\Theta(\log p + n)$

Overall complexity: $\Theta(n^2/p + \log p + n)$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Sequential time complexity: $T(n,1) = \Theta(n^2)$

Parallel overhead is dominated by all-gather:
$T_0(n,p) = \Theta(p(\log p + n)) \overset{\text{large } n}{\rightarrow} \Theta(pn)$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Sequential time complexity: $T(n,1) = \Theta(n^2)$

Parallel overhead is dominated by all-gather:
$T_0(n,p) = \Theta(p(\log p + n)) \overset{\text{large } n}{\rightarrow} \Theta(pn)$

$$n^2 \geq Cpn \quad \Rightarrow \quad n \geq Cp$$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Sequential time complexity: $T(n,1) = \Theta(n^2)$

Parallel overhead is dominated by all-gather:
$T_0(n,p) = \Theta(p(\log p + n)) \overset{\text{large } n}{\rightarrow} \Theta(pn)$

$$n^2 \geq Cpn \quad \Rightarrow \quad n \geq Cp$$

Scalability function: $M(f(p))/p$

$$M(n) = n^2 \quad \Rightarrow \quad \frac{M(Cp)}{p} = \frac{C^2p^2}{p} = C^2p$$

$\Rightarrow$ System is not highly scalable.

Analysis of the Parallel Algorithm

Let α be the time to compute an iteration.

Sequential execution time: αn^2

Computation time of parallel program: $\alpha n \left\lceil \frac{n}{p} \right\rceil$

All-gather requires $\lceil \log p \rceil$ messages with latency λ

Total vector elements transmitted: n

Total execution time:

$$\alpha n \left\lceil \frac{n}{p} \right\rceil + \lambda \lceil \log p \rceil + \frac{8n}{\beta}$$

Benchmarking

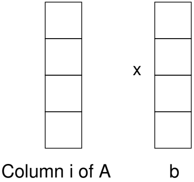
p	Predicted	Actual	Speedup	Mflops
1	63,4	63,4	1,00	31,6
2	32,4	32,7	1,94	61,2
3	22,3	22,7	2,79	88,1
4	17,0	17,8	3,56	112,4
5	14,1	15,2	4,16	131,6
6	12,0	13,3	4,76	150,4
7	10,5	12,2	5,19	163,9
8	9,4	11,1	5,70	180,2
16	5,7	7,2	8,79	277,8

(time in mili-seconds)

Columnwise Decomposition

Primitive task associated with

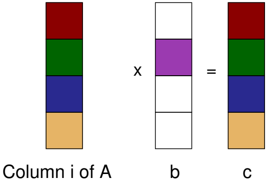
- column of matrix
- vector element



Columnwise Decomposition

Primitive task associated with

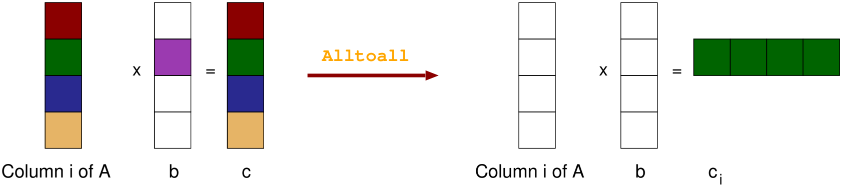
- column of matrix
- vector element



Columnwise Decomposition

Primitive task associated with

- column of matrix
- vector element

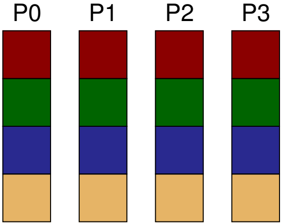


Columnwise Decomposition

Primitive task associated with

- column of matrix
- vector element

All-to-All Operation



All-to-All Operation

MPI_Alltoallv

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

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$

Communication complexity of alltoall:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$

Communication complexity of alltoall: $\Theta(p + n)$
($p - 1$ messages, and a total of n elements)

Overall complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$

Communication complexity of alltoall: $\Theta(p + n)$
($p - 1$ messages, and a total of n elements)

Overall complexity: $\Theta(n^2/p + n + p)$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Algorithm Scalability

Isoefficiency analysis: $T(n, 1) \geq CT_0(n, p)$

Sequential time complexity: $T(n, 1) = \Theta(n^2)$

The parallel overhead is alltoall and vector copying:
$T_0(n, p) = \Theta(p(p + n)) \overset{\text{large } n}{\rightarrow} \Theta(pn))$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Sequential time complexity: $T(n,1) = \Theta(n^2)$

The parallel overhead is alltoall and vector copying:
$T_0(n,p) = \Theta(p(p+n)) \overset{\text{large } n}{\rightarrow} \Theta(pn))$

$$n^2 \geq Cpn \Rightarrow n \geq Cp$$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Sequential time complexity: $T(n,1) = \Theta(n^2)$

The parallel overhead is alltoall and vector copying:
$T_0(n,p) = \Theta(p(p+n)) \overset{\text{large } n}{\rightarrow} \Theta(pn))$

$$n^2 \geq Cpn \Rightarrow n \geq Cp$$

Scalability function: $M(f(p))/p$

$$M(n) = n^2 \quad \Rightarrow \quad \frac{M(Cp)}{p} = \frac{C^2p^2}{p} = C^2p$$

⇒ System is not highly scalable.

Analysis of the Parallel Algorithm

Let α be the time to compute an iteration.

Sequential execution time: αn^2

Computation time of parallel program: $\alpha n \left\lceil \frac{n}{p} \right\rceil$

Alltoall requires $p-1$ messages each of length at most n/p (8 bytes per element `double`).

Total execution time:

$$\alpha n \left\lceil \frac{n}{p} \right\rceil + (p-1)\left(\lambda + \frac{8n}{p\beta}\right)$$

Benchmarking

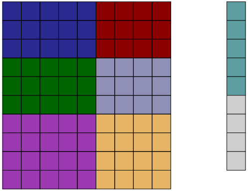
p	Predicted	Actual	Speedup	Mflops
1	63,4	63,8	1,00	31,4
2	32,4	32,9	1,92	60,8
3	22,2	22,6	2,80	88,5
4	17,2	17,5	3,62	114,3
5	14,3	14,5	4,37	137,9
6	12,5	12,6	5,02	158,7
7	11,3	11,2	5,65	178,6
8	10,4	10,0	6,33	200,0
16	8,5	7,6	8,33	263,2

(time in mili-seconds)

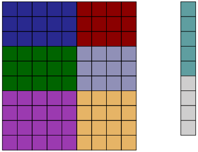
Checkerboard Decomposition

Primitive task associated with

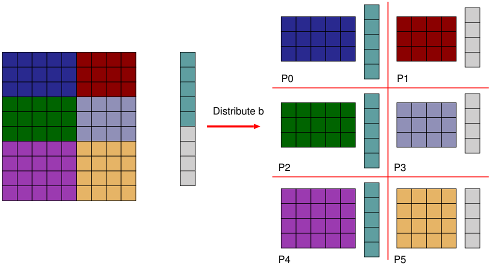
- rectangular blocks of matrix (processes form a 2D grid)
- vector
    - distributed by blocks among processes in first row of grid
    - each block copied to processes in the same column of grid



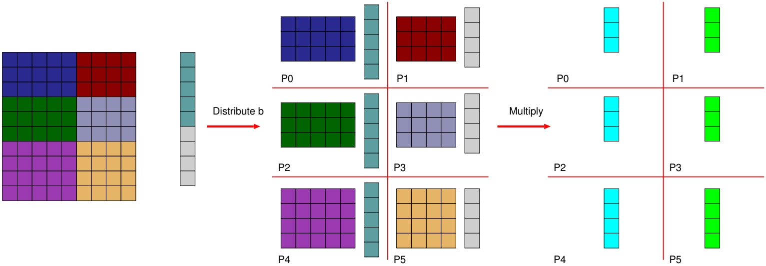
Algorithm Steps



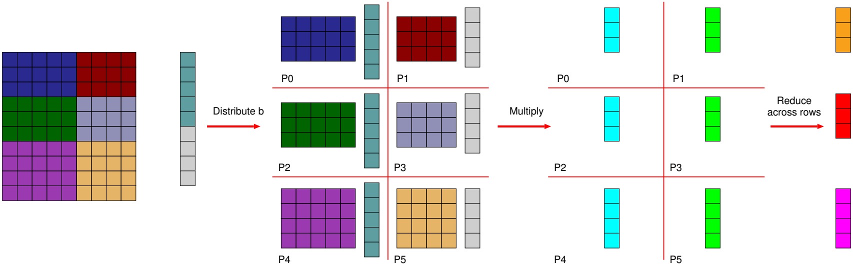
Algorithm Steps



Algorithm Steps



Algorithm Steps



Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Also, assume p is a square number: grid has size $n/\sqrt{p}$.

Sequential algorithm complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Also, assume p is a square number: grid has size $n/\sqrt{p}$.

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Also, assume p is a square number: grid has size $n/\sqrt{p}$.

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$
(each process computes a submatrix $n/\sqrt{p} \times n/\sqrt{p}$)

Communication complexity of reduce:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Also, assume p is a square number: grid has size $n/\sqrt{p}$.

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$
(each process computes a submatrix $n/\sqrt{p} \times n/\sqrt{p}$)

Communication complexity of reduce: $\Theta(\log\sqrt{p}\ (n/\sqrt{p})) = \Theta(n \log p/\sqrt{p})$
($\log\sqrt{p}$ messages, each with $n/\sqrt{p}$ elements)

Overall complexity:

Complexity Analysis

(for simplicity, assume square $n \times n$ matrix)

Also, assume p is a square number: grid has size $n/\sqrt{p}$.

Sequential algorithm complexity: $\Theta(n^2)$

Parallel algorithm computational complexity: $\Theta(n^2/p)$
(each process computes a submatrix $n/\sqrt{p} \times n/\sqrt{p}$)

Communication complexity of reduce: $\Theta(\log\sqrt{p}\ (n/\sqrt{p})) = \Theta(n \log p/\sqrt{p})$
($\log\sqrt{p}$ messages, each with $n/\sqrt{p}$ elements)

Overall complexity: $\Theta(n^2/p + n \log p/\sqrt{p})$

Algorithm Scalability

Isoefficiency analysis: $T(n, 1) \geq CT_0(n, p)$

Algorithm Scalability

Isoefficiency analysis: $T(n, 1) \geq CT_0(n, p)$

Sequential time complexity: $T(n, 1) = \Theta(n^2)$

The parallel overhead is reduce and vector copying:
$T_0(n, p) = \Theta(pn \log p/\sqrt{p}) = \Theta(n\sqrt{p} \log p)$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Sequential time complexity: $T(n,1) = \Theta(n^2)$

The parallel overhead is reduce and vector copying:
$T_0(n,p) = \Theta(pn \log p/\sqrt{p}) = \Theta(n\sqrt{p} \log p)$

$$n^2 \geq Cn\sqrt{p} \log p \Rightarrow n \geq C\sqrt{p} \log p$$

Algorithm Scalability

Isoefficiency analysis: $T(n,1) \geq CT_0(n,p)$

Sequential time complexity: $T(n,1) = \Theta(n^2)$

The parallel overhead is reduce and vector copying:
$T_0(n,p) = \Theta(pn \log p/\sqrt{p}) = \Theta(n\sqrt{p}\log p)$

$$n^2 \geq Cn\sqrt{p}\log p \Rightarrow n \geq C\sqrt{p}\log p$$

Scalability function: $M(f(p))/p$

$$M(n) = n^2 \quad \Rightarrow \quad \frac{M(C\sqrt{p}\log p)}{p} = \frac{C^2 p \log^2 p}{p} = C^2 \log^2 p$$

⇒ This system is much more scalable than the previous two implementations!

Creating Communicators

- collective communications involve all processes in a communicator

Creating Communicators

- collective communications involve all processes in a communicator
- need reductions among subsets of processes

Creating Communicators

- collective communications involve all processes in a communicator

- need reductions among subsets of processes

- processes in a virtual 2-D grid

Creating Communicators

- collective communications involve all processes in a communicator
- need reductions among subsets of processes
- processes in a virtual 2-D grid
- create communicators for processes in same row or same column

Creating a 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 dimension

`MPI_Cart_create()`

Creates communicator with Cartesian topology

MPI_Dims_create

```
int MPI_Dims_create (
   int nodes,       /* In - # procs in grid */
   int dims,        /* In - Number of dims */
   int *size        /* I/O- Size of each grid dim */
)
```

MPI_Cart_create

```
int MPI_Cart_create (
   MPI_Comm old_comm,   /* In - old communicator */
   int dims,            /* In - grid dimensions */
   int *size,           /* In - # procs in each dim */
   int *periodic,       /* In - 1 if dim i wraps around;
                                0 otherwise */
   int reorder,         /* In - 1 if process ranks
                                can be reordered */
   MPI_Comm *cart_comm  /* Out - new communicator */
)
```

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, periodic, 1,
                 &cart_comm);
```

Useful Grid-related Functions

MPI_Cart_rank()

given coordinates of a process in Cartesian communicator, returns process' 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 coordinates */
)
```

Useful Grid-related Functions

`MPI_Cart_coords()`

given rank of a process in Cartesian communicator, returns process' coordinates

```
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 */
)
```

MPI_Comm_split

`MPI_Comm_split()`

- partitions the processes of a communicator into one or more subgroups
- constructs a communicator for each subgroup
- allows processes in each subgroup to perform their own collective communications
- needed for columnwise scatter and rowwise reduce

MPI_Comm_split

```
int MPI_Comm_split (
   MPI_Comm old_comm,    /* In - Existing communicator */
   int partition,        /* In - Partition number */
   int new_rank,         /* In - Ranking order of processes
                                 in new communicator */
   MPI_Comm *new_comm    /* Out - New communicator shared by
                                processes in same partition */
)
```

Example: Create Communicators for Process Rows

```
MPI_Comm grid_comm;          /* 2-D process grid */

int grid_coords[2];          /* Location of process in grid */

MPI_Comm row_comm;           /* Processes in same row */

MPI_Comm_split (grid_comm, grid_coords[0], grid_coords[1],
                &row_comm);
```

Analysis of the Parallel Algorithm

Let α be the time to compute an iteration.

Sequential execution time: αn^2

Computation time of parallel program: $\alpha \left\lceil \frac{n}{\sqrt{p}} \right\rceil \left\lceil \frac{n}{\sqrt{p}} \right\rceil$

Reduce requires $\log \sqrt{p}$ messages each of length $\lambda + 8 \left\lceil \frac{n}{\sqrt{p}} \right\rceil / \beta$ (8 bytes per element `double`).

Total execution time:

$$\alpha \left\lceil \frac{n}{\sqrt{p}} \right\rceil \left\lceil \frac{n}{\sqrt{p}} \right\rceil + \log \sqrt{p} \left(\lambda + \frac{8}{\beta} \left\lceil \frac{n}{\sqrt{p}} \right\rceil \right)$$

Benchmarking

p	Predicted	Actual	Speedup	Mflops
1	63,4	63,4	1,00	31,6
4	17,8	17,4	3,64	114,9
8	9,7	9,7	6,53	206,2
16	6,2	6,2	10,21	322,6

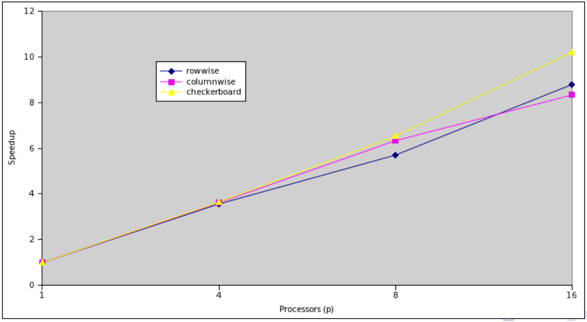
(time in mili-seconds)

Comparison of the Three Algorithms

Rowwise: $$\alpha n \left\lceil \frac{n}{p} \right\rceil + \lambda \lceil \log p \rceil + \frac{8n}{\beta}$$

Columnwise: $$\alpha n \left\lceil \frac{n}{p} \right\rceil + (p-1)\left(\lambda + \frac{8n}{p\beta}\right)$$

Checkerboard: $$\alpha \left\lceil \frac{n}{\sqrt{p}} \right\rceil \left\lceil \frac{n}{\sqrt{p}} \right\rceil + \log p \left(\lambda + \frac{8}{\beta}\left\lceil \frac{n}{\sqrt{p}} \right\rceil\right)$$



Review

- Matrix-vector multiplication
  - rowwise decomposition
  - columnwise decomposition
  - checkerboard decomposition

- Gather, scatter, alltoall

- Grid-oriented communications

Next Class

- load balancing
- termination detection