Parallel Programming

pauldj@aices.rwth-aachen.de

Collective Communication

```
          Barrier
Broadcast   ↔   Reduce
  Scatter   ↔   Gather
Allgather   ↔   Reduce-scatter
        Allreduce
         Alltoall
            ⋮
```

References

- "Collective Communication: Theory, Practice, and Experience", Chan, Heimlich, Purkayastha, van de Geijn. (FLAME working note #22)
- Collective Communications in MPI

  `http://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/node72.html`

Collective Communication

- Synchronization
  `Barrier` ← **Almost never needed!**
- Data Movement
  `Broadcast, Scatter, Gather, Allgather, Alltoall`
- Reductions
  `Reduce, Reduce-scatter, Allreduce, Scan, ...`

For all collectives: no tags; blocking.

`int MPI_BCast(...)`

Before:

Node_0	Node_1	Node_2	Node_3
		α	

After:

Node_0	Node_1	Node_2	Node_3
α	α	α	α

`int MPI_Reduce(...)`

Before:

Node_0	Node_1	Node_2	Node_3
δ_0	δ_1	δ_2	δ_3

After:

Node_0	Node_1	Node_2	Node_3
		$\text{Op}_{i=0}^{p-1} \delta_i$	

`MPI_Op:`
`MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND, ...,`

`MPI_Datatype:`
`MPI_CHAR, MPI_INT, MPI_UNSIGNED, MPI_FLOAT, MPI_DOUBLE, ...`

`int MPI_Scatter(...)`

Before:

$Node_0$	$Node_1$	$Node_2$	$Node_3$
		v[0] v[1] v[2] v[3]	

After:

$Node_0$	$Node_1$	$Node_2$	$Node_3$
v[0]	v[1]	v[2]	v[3]

`int MPI_Gather(...)`

Before:

Node_0	Node_1	Node_2	Node_3
v[0]	v[1]	v[2]	v[3]

After:

Node_0	Node_1	Node_2	Node_3
		v[0] v[1] v[2] v[3]	

`int MPI_Allgather(...)`

Before:

Node_0	Node_1	Node_2	Node_3
v[0]			
	v[1]		
		v[2]	
			v[3]

After:

Node_0	Node_1	Node_2	Node_3
v[0]	v[0]	v[0]	v[0]
v[1]	v[1]	v[1]	v[1]
v[2]	v[2]	v[2]	v[2]
v[3]	v[3]	v[3]	v[3]

`int MPI_Reduce_scatter(...)`

Before:

Node_0	Node_1	Node_2	Node_3
$v_0[0]$	$v_1[0]$	$v_2[0]$	$v_3[0]$
$v_0[1]$	$v_1[1]$	$v_2[1]$	$v_3[1]$
$v_0[2]$	$v_1[2]$	$v_2[2]$	$v_3[2]$
$v_0[3]$	$v_1[3]$	$v_2[3]$	$v_3[3]$

After:

Node_0	Node_1	Node_2	Node_3
$\underset{i}{\text{Op}}\ v_i[0]$			
	$\underset{i}{\text{Op}}\ v_i[1]$		
		$\underset{i}{\text{Op}}\ v_i[2]$	
			$\underset{i}{\text{Op}}\ v_i[3]$

`int MPI_Allreduce(...)`

Before:

Node_0	Node_1	Node_2	Node_3
δ_0	δ_1	δ_2	δ_3

After:

Node_0	Node_1	Node_2	Node_3
$\bigoplus_{i=0}^{p-1} \delta_i$	$\bigoplus_{i=0}^{p-1} \delta_i$	$\bigoplus_{i=0}^{p-1} \delta_i$	$\bigoplus_{i=0}^{p-1} \delta_i$

`int MPI_Alltoall(...)`

Before:

$Node_0$	$Node_1$	$Node_2$	$Node_3$
$v_0[0]$	$v_1[0]$	$v_2[0]$	$v_3[0]$
$v_0[1]$	$v_1[1]$	$v_2[1]$	$v_3[1]$
$v_0[2]$	$v_1[2]$	$v_2[2]$	$v_3[2]$
$v_0[3]$	$v_1[3]$	$v_2[3]$	$v_3[3]$

After:

$Node_0$	$Node_1$	$Node_2$	$Node_3$
$v_0[0]$	$v_0[1]$	$v_0[2]$	$v_0[3]$
$v_1[0]$	$v_1[1]$	$v_1[2]$	$v_1[3]$
$v_2[0]$	$v_2[1]$	$v_2[2]$	$v_2[3]$
$v_3[0]$	$v_3[1]$	$v_3[2]$	$v_3[3]$

More Collectives

Variable length

- `MPI_Scatterv`
- `MPI_Gatherv`
- `MPI_Allgatherv`
- `MPI_Alltoallv`

Partial reduction

- `MPI_Scan`

Non-blocking collectives

- `MPI_I*`

Multiple communicators

- Example 1)
  "World" (everybody) + "Workers" (everybody-masters) + "Masters" (everybody-workers)
- Example 2)
  $r \times c$ mesh of processes; communication within rows/columns only
- `MPI_Comm_create(...)`
  `MPI_Comm_split(...)`
  ...

More Collectives

Variable length

- `MPI_Scatterv`
- `MPI_Gatherv`
- `MPI_Allgatherv`
- `MPI_Alltoallv`

Partial reduction

- `MPI_Scan`

Non-blocking collectives

- `MPI_I*`

More Communicators

Multiple communicators

- Example 1)
  "World" (everybody) + "Workers" (everybody-masters) + "Masters" (everybody-workers)
- Example 2)
  $r \times c$ mesh of processes; communication within rows/columns only
- `MPI_Comm_create(...)`
  `MPI_Comm_split(...)`
  ...

Collective Communication: Lower Bounds

Cost of communication: $\alpha + n\beta$

Cost of computation: $\gamma \ \#\text{ops}$

$\alpha =$ "latency", "startup" $\beta =$ 1/"bandwidth"

$n =$ length of the message $\gamma =$ cost of 1 flop

$p =$ # of processes

Primitive	Latency	Bandwidth	Computation
`Broadcast`	$\lceil \log_2(p) \rceil \alpha$	$n\beta$	-
`Reduce`	$\lceil \log_2(p) \rceil \alpha$	$n\beta$	$\frac{p-1}{p} n\gamma$
`Scatter`	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	-
`Gather`	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	-
`Allgather`	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	-
`Reduce-Scatter`	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$

Implementation of Bcast and Reduce

- IDEA: recursive doubling / “Minimum Spanning Tree” (MST)
  At each step, double the number of active processes.

- How to map the idea to the specific topology?
  - ring: linear doubling
  - (2d) mesh: 1 dimension first, then another, then another ...
  - hypercube: obvious, same as mesh

- Cost?
  - # steps: $\log_2 p$
  - cost(step): $\alpha + n\beta$
  - total time: $\log_2(p)\alpha + \log_2(p)n\beta$
  - note: cost(p^2) = 2 cost(p)!

  lower bound: $\log_2(p)\alpha + n\beta$

- Reduce
  BCast in reverse; cost(computation) ?

Implementation of Gather (and Scatter)

- IDEA: MST again
  At step i, only $\frac{1}{2^i}$-th of the message is sent
- # steps: $\log_2 p$
- cost(step$_i$): $\alpha + \frac{n}{2^i}\beta$
- total time: $\sum_{i=1}^{\log_2(p)} \alpha + \frac{n}{2^i}\beta = \log_2(p)\alpha + \frac{p-1}{p}n\beta$
- lower bound: $\log_2(p)\alpha + \frac{p-1}{p}n\beta$ optimal!

Implementation of Allgather (and Reduce-scatter)

- IDEA: “Recursive-doubling” (bidirectional exchange)
  Recursive allgather of half data + exchange data between disjoint nodes.

Node_0	Node_1	Node_2	Node_3
v[0]			
	v[1]		
		v[2]	
			v[3]

$\Downarrow$

Node_0	Node_1	Node_2	Node_3
v[0]	v[0]		
v[1]	v[1]		
		v[2]	v[2]
		v[3]	v[3]

$\Downarrow$

Node_0	Node_1	Node_2	Node_3
v[0]	v[0]	v[0]	v[0]
v[1]	v[1]	v[1]	v[1]
v[2]	v[2]	v[2]	v[2]
v[3]	v[3]	v[3]	v[3]

- # steps: $\log_2 p$
- total time:

$$\sum_{i=1}^{\log_2(p)} \alpha + \frac{n}{2^i}\beta = \log_2(p)\alpha + \frac{p-1}{p}n\beta$$

Another Implementation of Allgather

- IDEA: Cyclic algorithm

Node_0	Node_1	Node_2	Node_3
v[0]			
	v[1]		
		v[2]	
			v[3]

$\Downarrow$

Node_0	Node_1	Node_2	Node_3
v[0]	v[0]		
	v[1]	v[1]	
		v[2]	v[2]
v[3]			v[3]

$\Downarrow$

Node_0	Node_1	Node_2	Node_3
v[0]	v[0]	v[0]	
	v[1]	v[1]	v[1]
v[2]		v[2]	v[2]
v[3]	v[3]		v[3]

- # steps: $p-1$
- total time:

$$\sum_{i=1}^{p-1} \alpha + \frac{n}{p}\beta = (p-1)\alpha + \frac{p-1}{p} n\beta$$

Another Implementation of Bcast

- IDEA: Scatter + cyclic algorithm

Dense Matrix-Vector Product: Scalability

1D matrix distribution

$$y := Ax, \qquad x, y \in \mathbb{R}^n, \quad A \in \mathbb{R}^{n \times n}$$

- A is partitioned by rows and distributed over p processes. Process i owns the block of rows A_i.
- Similarly for x: process i owns x_i.
- **Goal:** Compute y and distribute it the same way as x.

$$A = \left[\begin{array}{c} A_0 \\ \hline A_1 \\ \hline \vdots \\ \hline A_{p-1} \end{array}\right], \quad x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{p-1} \end{array}\right), \quad \text{and} \quad y = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_{p-1} \end{array}\right)$$

Note: A_i, x_i and y_i indicate a block of rows, as opposed to a single one.

Dense Matrix-Vector Product: Scalability

1D matrix distribution

Algorithm

1. $x =$ `Allgather`(x_i) — x becomes available to every process
2. $y_i = A_i x$ — local computation

Parallel cost (lower bound for $T_p(n)$)

1. $\lceil \log_2(p) \rceil \alpha + \frac{p-1}{p} n\beta \approx \log_2(p)\alpha + n\beta$
2. $2\frac{n^2}{p}\gamma$

Sequential cost $T_1(n) = 2n^2\gamma$

GEMV: Scalability

1D matrix distribution

Speedup $$S_p(n) = \frac{T_1(n)}{T_p(n)} = \frac{2n^2\gamma}{\log_2(p)\alpha + n\beta + 2\frac{n^2}{p}\gamma}$$

Efficiency $$E_p(n) = \frac{S_p(n)}{p} = \frac{1}{1 + \frac{p\log_2(p)}{2n^2}\frac{\alpha}{\gamma} + \frac{p}{2n}\frac{\beta}{\gamma}}$$

Strong scalability $\lim_{p\to\infty} E_p(n) = 0$ ✖

Weak scalability

M = local memory; Mp = combined memory;
n_M = largest problem solvable with p processes $\Rightarrow n_M^2 = Mp$

$$\lim_{p\to\infty} E_p(n_M) = \frac{1}{1 + \frac{\log_2(p)}{2M}\frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2\sqrt{M}}\frac{\beta}{\gamma}} = 0$$ ✖

Exercise: $y = Ax$, A distributed by columns

- A is partitioned by columns and distributed over p processes. Process i owns the block of columns A_i.
- Process i owns x_i.
- **Goal:** Compute y and distribute it the same way as x.

$$A = \left[\begin{array}{c|c|c|c} A_0 & A_1 & \dots & A_{p-1} \end{array}\right], \quad x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{p-1} \end{array}\right), \quad \text{and} \quad y = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_{p-1} \end{array}\right)$$

Scalability

Algorithm

1. $y^{(i)} = A_i x_i$ — local computation
2. $y =$ `Reduce-Scatter(`$y^{(i)}$`)` — reduction-sum of $y^{(i)}$'s + scatter

Parallel cost

1. $2\frac{n^2}{p}\gamma$
2. $\lceil \log_2(p) \rceil \alpha + \frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma \approx \log_2(p)\alpha + n(\beta + \gamma)$
3. Lower bound for $T_p(n) = \log_2(p)\alpha + n(\beta + \gamma) + 2\frac{n^2}{p}\gamma$

$T_p(n)$ has one extra term ($n\gamma$) with respect to the case of A partitioned by rows, therefore this algorithm is also not scalable.

GEMV: Scalability

2D matrix distribution

- A is partitioned according to a 2D $p = r \times c$ mesh;
  The (i, j) process (P_{ij}) owns the block A_{ij}.
- x and y are partitioned in p chunks;
  x is mapped to the mesh by colums, y by rows.

Example: $p = 2 \times 3$

$$A = \left[\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \end{array}\right], \quad x = \begin{pmatrix} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_5 \end{pmatrix}, \quad \text{and} \quad y = \begin{pmatrix} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_5 \end{pmatrix}$$

- P_{ij} owns A_{ij}, $\begin{array}{l} P_{00} \text{ owns } x_0 \\ P_{10} \text{ owns } x_1 \\ P_{01} \text{ owns } x_2 \\ P_{11} \text{ owns } x_3 \\ P_{02} \text{ owns } x_4 \\ P_{12} \text{ owns } x_5 \end{array}$, and $\begin{array}{l} P_{00} \text{ owns } y_0 \\ P_{01} \text{ owns } y_1 \\ P_{02} \text{ owns } y_2 \\ P_{10} \text{ owns } y_3 \\ P_{11} \text{ owns } y_4 \\ P_{12} \text{ owns } y_5 \end{array}$.

GEMV: Scalability

2D matrix distribution

Algorithm

1. $x_I =$ `Allgather`(x_i) within columns — x_I is a "block"
2. $y_J = A_{ij}x_I$ — local computation
3. $y_j =$ `Reduce-scatter`(y_J) within rows

Parallel cost (lower bound for $T_p(n)$)

1. $\lceil \log_2(r) \rceil \alpha + \frac{r-1}{p} n\beta \approx \log_2(r)\alpha + \frac{n}{c}\beta$
2. $2\frac{n^2}{p}\gamma$
3. $\lceil \log_2(c) \rceil \alpha + \frac{c-1}{p} n\beta + \frac{c-1}{p} n\gamma \approx \log_2(c)\alpha + \frac{n}{r}\beta + \frac{n}{r}\gamma$

Sequential cost $T_1(n) = 2n^2\gamma$

GEMV: Scalability

2D matrix distribution

Parallel cost: $2\frac{n^2}{p}\gamma + (\log_2(r) + \log_2(c))\,\alpha + \left(\frac{n}{c} + \frac{n}{r}\right)\beta + \frac{n}{r}\gamma$

assuming $r = c = \sqrt{p}$: $2\frac{n^2}{p}\gamma + \log_2(p)\alpha + \frac{n}{\sqrt{p}}(2\beta + \gamma)$

Speedup $S_p(n) = \dfrac{p}{1 + \frac{p\log_2(p)}{2n^2}\frac{\alpha}{\gamma} + \frac{\sqrt{p}}{2n}\frac{2\beta+\gamma}{\gamma}}$

Efficiency $E_p(n) = S_p(n)/p = \ldots$

Strong scalability $\lim_{p\to\infty} E_p(n) = 0$ ✖

Weak scalability

M = local memory; Mp = combined memory;
n_M = largest problem solvable with p processes $\Rightarrow n_M^2 = Mp$

$\lim_{p\to\infty} E_p(n_M) = \dfrac{1}{1 + \frac{\log_2(p)}{2M}\frac{\alpha}{\gamma} + \frac{1}{2\sqrt{M}}\frac{2\beta+\gamma}{\gamma}} = 0$ ✖ … ✓

Exercise

2D matrix distribution

- A is partitioned according to a 2D $p = r \times c$ mesh;
  The (i, j) process (P_{ij}) owns the block A_{ij}.
- Both x and y are partitioned in p chunks,
  and mapped to the mesh by colums.

Example: $p = 2 \times 3$

$$A = \left[\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \end{array}\right], \quad x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_5 \end{array}\right), \quad \text{and} \quad y = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_5 \end{array}\right)$$

- P_{ij} owns A_{ij}, $\begin{array}{l} P_{00} \text{ owns } x_0 \\ P_{10} \text{ owns } x_1 \\ P_{01} \text{ owns } x_2 \\ P_{11} \text{ owns } x_3 \\ P_{02} \text{ owns } x_4 \\ P_{12} \text{ owns } x_5 \end{array}$, and $\begin{array}{l} P_{00} \text{ owns } y_0 \\ P_{10} \text{ owns } y_1 \\ P_{01} \text{ owns } y_2 \\ P_{11} \text{ owns } y_3 \\ P_{02} \text{ owns } y_4 \\ P_{12} \text{ owns } y_5 \end{array}$.

Parallel Matrix Distribution

How to store a (large) matrix using $p = r \times c$ processes?

- 1D, block of rows (or columns) — **Bad idea**
- 1D (block) cyclic, either by rows or columns — **Bad idea**
- 2D, $r \times c$ quadrants — **Not so good**
- 2D (block) cyclic — **Good idea!**

References

- Applet:
  `http://acts.nersc.gov/scalapack/hands-on/datadist.html`
- "Introduction to High-Performance Scientific Computing" by Victor Eijkhout (free download).
- "A Comprehensive Approach to Parallel Linear Algebra Libraries" (Technical Report, University of Texas). Chapter 3, pages 19–40.