innovate achieve lead

BITS Pilani
Pilani Campus

Solving Linear Systems

Solving Linear Systems

- Many scientific and engineering problems can take the form of a system of linear equations. Here is a sampling of the domains from which these problems arise:
  - structural analysis (civil engineering)
  - heat transport (mechanical engineering)
  - analysis of power grids (electrical engineering)
  - production planning (economics)
  - regression analysis (statistics)
- Because linear systems derived from realistic problems are often quite large, there is good reason to learn how to solve them efficiemly on parallel computers

Solving Linear Systems

- Upper triangular systems
  - can be solved using the back substitution algorithm
- Dense systems of linear equations
  - Type equation here.Gaussian elimination algorithm transforms a dense system into an upper triangular system, which can then be solved using back substitution

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

$$
\begin{array}{cccc}
1 & 1 & -1 & 4 \\
0 & -2 & -3 & 1 \\
0 & 0 & 2 & -3 \\
0 & 0 & 0 & 2
\end{array}
$$

Solving Linear Systems

- System of linear equations
  - Solve $Ax = b$ for x
- Special matrices
  - Symmetrically banded
  - Upper triangular
  - Lower triangular
  - Diagonally dominant
  - Symmetric

Back Substitution

- Used to solve upper triangular system $Tx = b$ for x
- Methodology: one element of x can be immediately computed
- Use this value to simplify system, revealing another element that can be immediately computed
- Repeat

Back Substitution

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

Back Substitution

$$
\begin{array}{lllll}
1x_0 & +1x_1 & -1x_2 & +4x_3 & = & 8 \\
 & -2x_1 & -3x_2 & +1x_3 & = & 5 \\
 & & 2x_2 & -3x_3 & = & 0 \\
x_3 = 2 & & & 2x_3 & = & 4
\end{array}
$$

Back Substitution

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

Back Substitution

$$
\begin{array}{llll}
1x_0 + 1x_1 - 1x_2 & = & 0 \\
- 2x_1 - 3x_2 & = & 3 \\
2x_2 & = & 6 \\
x_2 = 3 \qquad 2x_3 & = & 4
\end{array}
$$

Back Substitution

$$
\begin{aligned}
1x_0 \quad & +1x_1 & & & = & \quad 3 \\
& -2x_1 & & & = & \quad 12 \\
& & 2x_2 & & = & \quad 6 \\
& & & 2x_3 & = & \quad 4
\end{aligned}
$$

Back Substitution

$$
\begin{aligned}
1x_0 \quad & +1x_1 & & & = & \quad 3 \\
& -2x_1 & & & = & \quad 12 \\
& & 2x_2 & & = & \quad 6 \\
x_1 = -6 \quad & & & 2x_3 & = & \quad 4
\end{aligned}
$$

Back Substitution

$$
\begin{aligned}
1x_0 &= 9 \\
-2x_1 &= 12 \\
2x_2 &= 6 \\
2x_3 &= 4
\end{aligned}
$$

Back Substitution

$$
\begin{aligned}
1x_0 &= 9 \\
-2x_1 &= 12 \\
2x_2 &= 6 \\
2x_3 &= 4
\end{aligned}
$$

$$x_0 = 9$$

Back Substitution – Sequential Algorithm

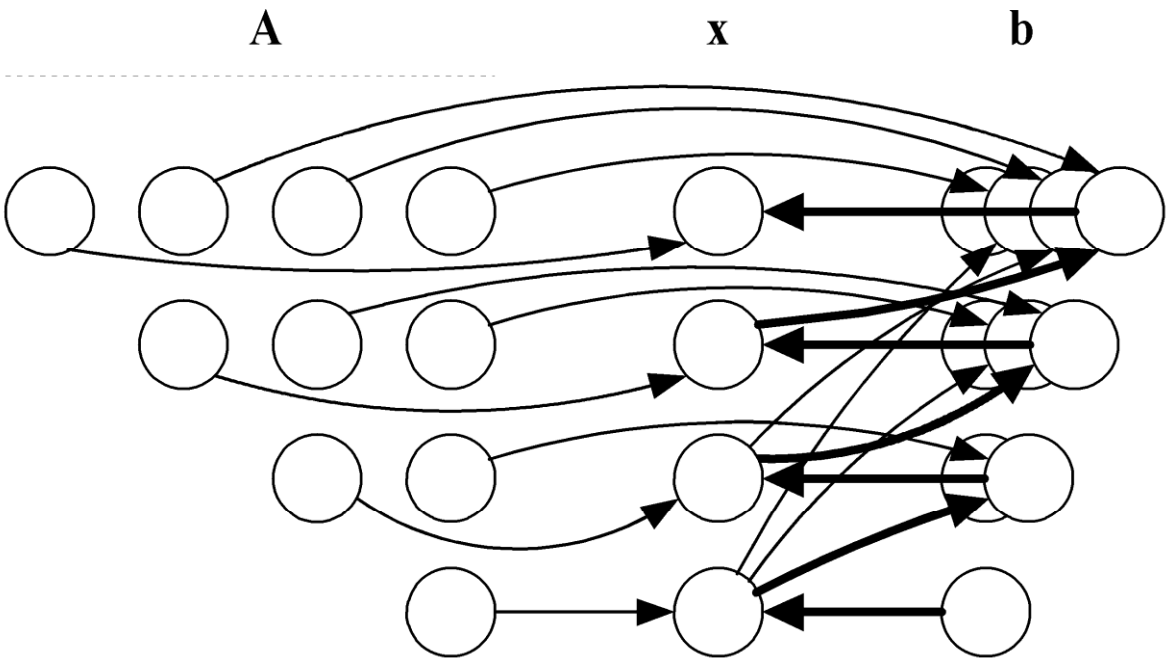
```
for i ← n – 1 down to 1 do
        x [ i ] ← b [ i ] / a [ i, i ]
        for j ← 0 to i – 1 do
                b [ j ] ← b [ j ] – x [ i ] × a [ j, i ]
        endfor
endfor
```

Time complexity: $\Theta(n^2)$

- We cannot execute the outer for loop in paralle
- We can execute the inner for loop in parallel

Back Substitution – Sequential Algorithm

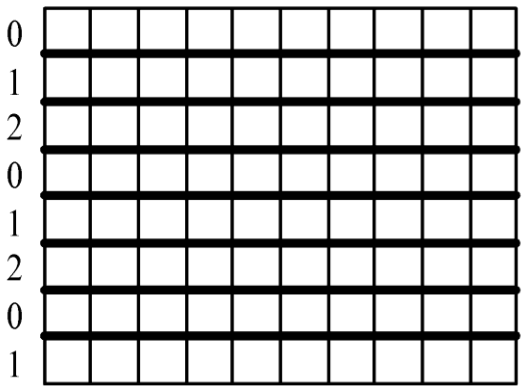
- Data dependence diagram



- We cannot execute the outer loop in parallel.
- We can execute the inner loop in parallel.

Back Substitution – row oriented algorithm

- Row-oriented Algorithm
  - Associate primitive task with each row of *A* and corresponding elements of *x* and *b*
  - During iteration *i* task associated with row *j* computes new value of b_j
  - Task *i* must compute x_i and broadcast its value
  - Agglomerate
    - As the work is not uniform for all processes, go for cyclic distribution
    - Process k controls all rows i where i mod p =k
    - This is called rowwise interleaved striped decomposition



Back Substitution – row oriented algorithm

- Communication complexity
  - During each iteration the process controlling row i broadcasts x_i to the other processes
  - → (logp)(t_s+t_w)
  - For n-1 iterations,
  - → nlogp (t_s+t_w)
- Computational complexity
  - Total no of iterations done in j loop
    - 0+….+n-3+n-2 → (n-2)+(n-1)/2
  - Average per process
    - n^2/p
- Total parallel runtime
  - $T_p = \frac{n^2}{p} + nlogp(t_s + t_w)$

Back Substitution – row oriented algorithm

- Scalability analysis
  - $W=n^2$
  - $T_0=pTp-W$
  - $T_p = \frac{n^2}{p} + nlogp(ts + tw)$
  - $T_0=pnlogp(t_s+t_w)$
- $W=Kpnlogp(t_s+t_w)$
- → $n^2=Kpnlogp(t_s+t_w)$
- $n=kplogp(ts+tw)$
- $n^2=k^2p^2log^2p(ts+tw)^2$
- Isoefficiency function is p^2log^2p

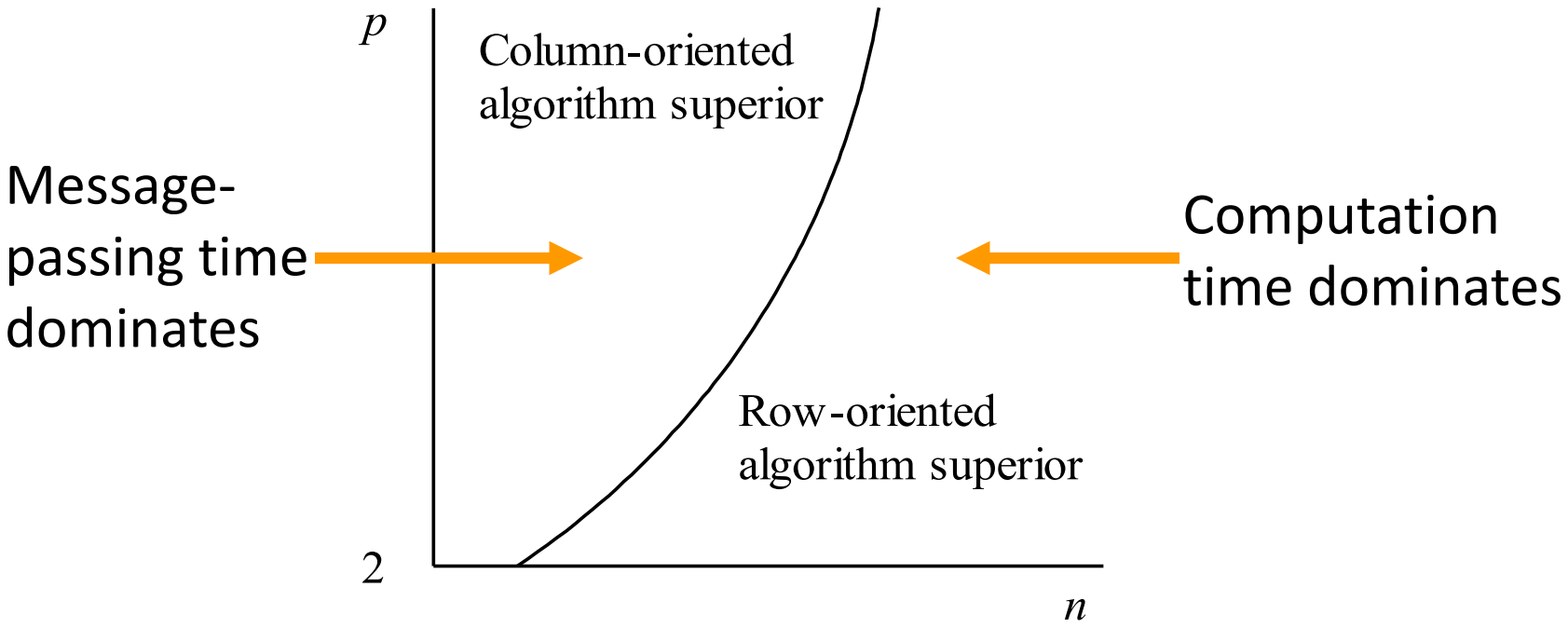
Back Substitution – column oriented algorithm

- An alternative design associates one primitive task per column of A
- Last task starts with vector *b*
- During iteration *i* task *i* computes x_i, updates *b*, and sends *b* to task *i* -1
- In other words, no computational concurrency
- Agglomerate tasks in interleaved fashion

Back Substitution – column oriented algorithm

- Computational complexity
  - Since *b* always updated by a single process, computational complexity same as sequential algorithm: $\Theta(n^2)$
- communication complexity
  - Since elements of *b* passed from one process to another each iteration
    - communication complexity is $n(t_s+t_w n)$
- Total parallel runtime
  - $T_p = n^2 + n(t_s + nt_w)$

Comparison


p
Column-oriented
algorithm superior
Message-
passing time
dominates
Computation
time dominates
Row-oriented
algorithm superior
2
n

Gaussian Elimination

- 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
- We may perform three operations on a system of linear equations without changing the value of the solution
  - Multiply every term of an equation by a nonzero constant
  - Interchange two equations
  - Add a multiple of one equation to another equation
- Hence we can replace any row of a linear system by the sum of that row and a nonzero multiple of any row of the system.

Gaussian Elimination

- Here is a dense system of linear equations
  - Coefficient.$a_{1,0}$ = 2 and $a_{0,0}$ = 4. Dividing 2 by 4 yields 0.5. If we replace row 1 by the sum of row 1 and -0.5 times row 0, the first term of row 1 becomes 0
  - if we replace row 2 by the sum of row 2 and I times row 0, the first term of row 2 becomes 0
  - Replacing row 3 by the sum of row 3 and -2 limes row 0 causes the first term of row 3 to become 0

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

Gaussian Elimination

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

Now that we've driven to 0 all coefficients below $a_{0,0}$, let's focus on coefficient in the column below $a_{1,1}$. We replace row 2 by the sum of row 2 and 1 times row 1. We replace row 3 by the sum of row 3 and 2 times row 1.

Gaussian Elimination

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

- Finally, we need to drive to 0 the coefficient below $a_{2,2}$. We replace row 3 by the sum of row 3 and -2 times row 2

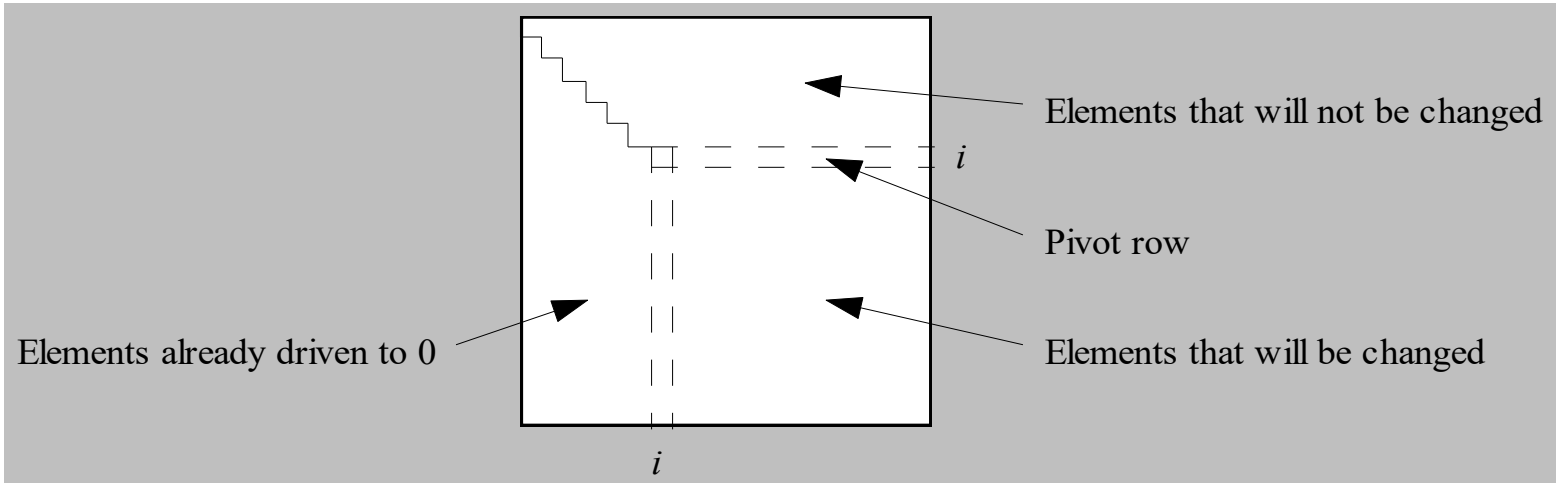
Gaussian Elimination

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

- This completes our transformation of the dense linear system into an upper triangular system. At this point we can use back substitution to transform the system into diagonal form, allowing us to determine the solution vector

Gaussian Elimination

- All nonzero elements below the diagonal and to the left of column i have already been eliminated. In step i the nonzero elements below the diagonal in column i are eliminated by replacing each row j, where i+1<= j < n, with the sum of row j and $-a_{j,i}/a_{i,i}$ times row i. After n - 1 such iterations, the linear system is upper triangular.
  - during iteration i row i is the pivot row, that is, the row used to drive to zero all nonzero elements below the diagonal in column I
  - if the pivot element $a_{i,i}$ is close to zero, dividing by it can result in significant roundoff errors. Hence this approach does not in general exhibit good numerical stability on digital computers.



Gaussian elimination with partial pivoting

- In step i of Gaussian elimination with partial pivoting of rows, rows i through n - 1 are searched for the row whose column i element has the largest absolute value. This row is swapped (pivoted) with row i. Once this has been done, this row will be used as pivot row.

```
for i ← 0 to n − 1

  {Find pivot row picked}
  magnitude ← 0
  for j ← i to n − 1
    if |a[loc[j], i]| > magnitude
      magnitude ← |a[loc[j],i]|
      picked ← j
    endif
  endfor
  tmp ← loc[i]
  loc[i] ← loc[picked]
  loc[picked] ← tmp
```

Requires about $2n^3/3$ floating-point operations

```
{Back substitution}
for i ← n − 1 down to 0
  x[i] ← a[loc[i], n]/a[loc[i], i]
  for j ← 0 to i − 1 do
    a[loc[j], n] ← a[loc[j], n] − x[i] × a[loc[j], i]
  endfor
```

```
{Drive to 0 column i elements in unmarked rows}
for j ← i + 1 to n − 1
  t ← a[loc[j], i]/a[loc[i], i]
  for k ← i + 1 to n + 1
    a[loc[j], k] ← a[loc[j], k] − a[loc[i], k] × t
  endfor
endfor
```

Gaussian elimination with partial pivoting

- A study of the algorithm's data dependences reveals that both the innermost for loop indexed by k and the middle for loop indexed by j can be executed in parallel
- Once the pivot row has been found, the modifications to all unmarked rows may occur simultaneously
  - Row oriented Algorithm
  - Column oriented algorithm

Row oriented Algorithm

- Associate primitive task with each row of A and corresponding elements of x and b
- A kind of reduction needed to find the identity of the pivot row
  - Tournament: want to determine identity of row with largest value, rather than largest value itself
  - Could be done with two all-reductions
  - MPI provides a simpler, faster mechanism
    - Using MPI_Allreduce (&local, &global, 1, MPI_DOUBLE_INT, MPI_MAXLOC, MPI_COMM_WORLD);
- Computation across tasks is not uniform.
  - Block distribution: row wise block-striped decomposition strategy
    - n^3/p
  - Cyclic distribution of n/p rows to p tasks

Row oriented Algorithm

- Communication complexity
  - Complexity of tournament (finding pivot row) assuming hypercube all-reduce: $(t_s+t_w)\log p$
  - Complexity of broadcasting pivot row (after changes): $(t_s+t_w n) \log p$
  - A total of n - 1 iterations
    - Overall communication complexity: $n((t_s+t_w)\log p+ (t_s+t_w n) \log p) \rightarrow (n+\log p) t_s+ ((n+n^2)\log p) t_w$
- Computation complexity
  - There n^3 computations. As it is cyclic distribution, computations are approx. uniformly spread across processes
  - n^3/p
- Total parallel runtime
  - $Tp=n^3/p + (n+\log p) t_s+ ((n+n^2)\log p) t_w$

Gaussian Elimination

- Scalability analysis
  - $W=n^3$
  - $T_p=n^3/p + (n+logp)\ t_s + ((n+n^2)logp)\ t_w$
  - $T_0=pTp-W$
  - ➔ $T_0=(pn+plogp)\ t_s + ((pn+pn^2)logp)\ t_w$
- $W=Kpn^2logpt_w$
- → $n^3=Kpn^2logpt_w$
- $n=kplogptw$
- $n^3=k^3p^3log^3ptw^3$
- Isoefficiency function is p^3log^3p
  - Poor scalability
- Scalability function
  - n >= plogp; Memory per process $=n^2= (plogp)^2/p=plog^2p$

Column-oriented Algorithm

- Associate a primitive task with each column of *A* and another primitive task for *b*
- During iteration *i* task controlling column *i* determines pivot row and broadcasts its identity
- During iteration *i* task controlling column *i* must also broadcast column *i* to other tasks
- Agglomerate tasks in an interleaved fashion to balance workloads
- Runtime and Isoefficiency same as row-oriented algorithm

Comparison of Two Algorithms

- Both algorithms evenly divide workload
- Both algorithms do a broadcast each iteration
- Difference: identification of pivot row
  - Row-oriented algorithm does search in parallel but requires all-reduce step
  - Column-oriented algorithm does search sequentially but requires no communication
- Row-oriented superior when *n* relatively larger and *p* relatively smaller

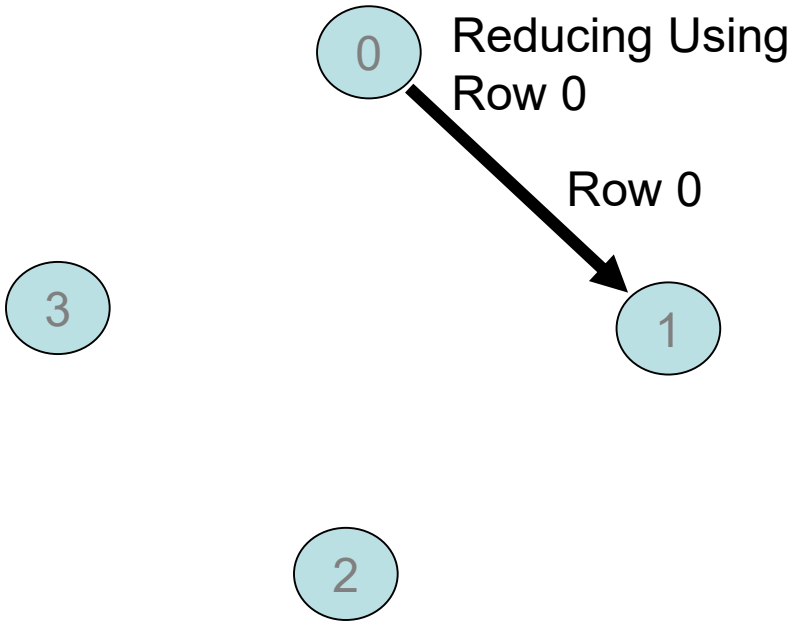
Problems with These Algorithms

- They break parallel execution into computation and communication phases
- Processes not performing computations during the broadcast steps
- Time spent doing broadcasts is large enough to ensure poor scalability

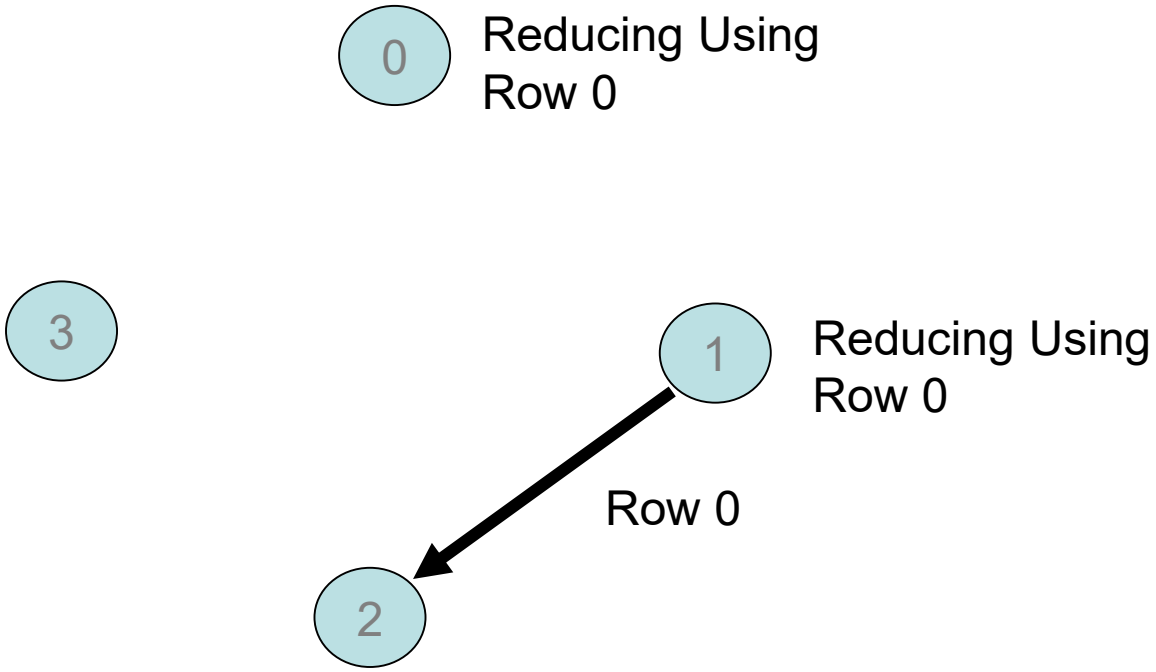
Pipelined, Row-Oriented Algorithm

- Want to overlap communication time with computation time
- We could do this if we knew in advance the row used to reduce all the other rows.
- Let's pivot columns instead of rows!
- In iteration *i* we can use row *i* to reduce the other rows.
  - When the algorithm begins execution, process 0 searches row 0 to determine the column containing the element with the largest magnitude. As soon as it finishes the search, it sends a message to task 1 containing row 0 and the index of the pivot element. While this message is being transmitted, process 0 can reduce the rest of its share of the augmented matrix.

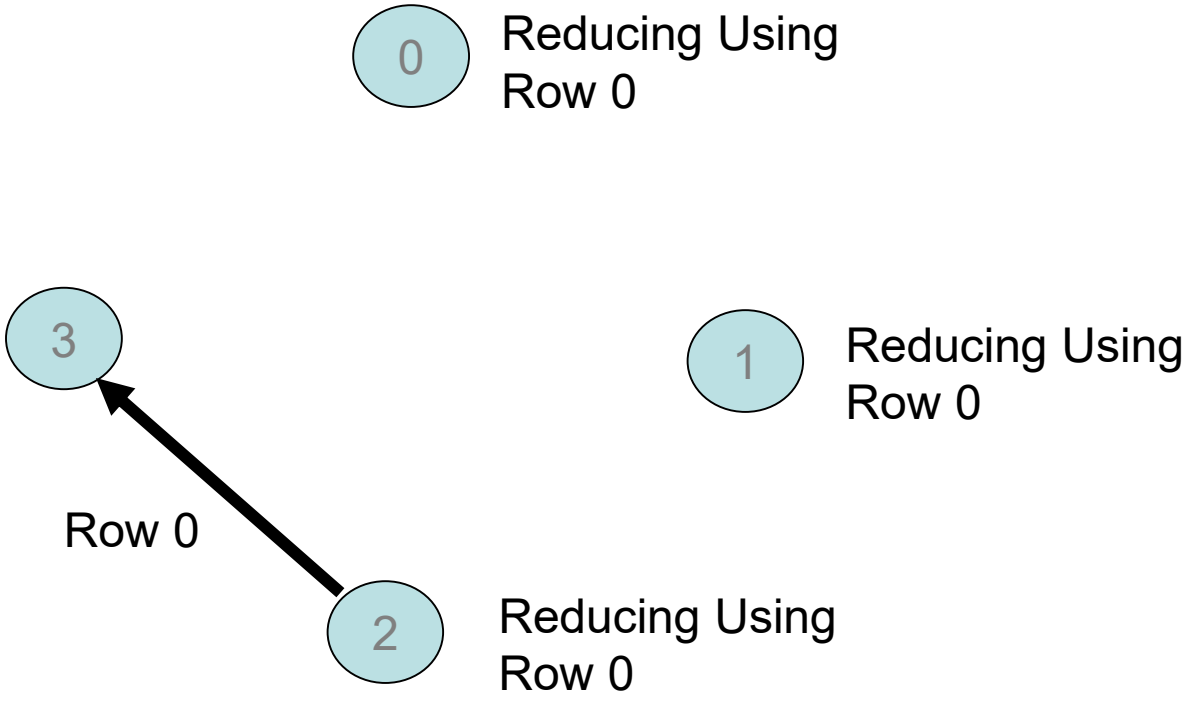
Communication Pattern



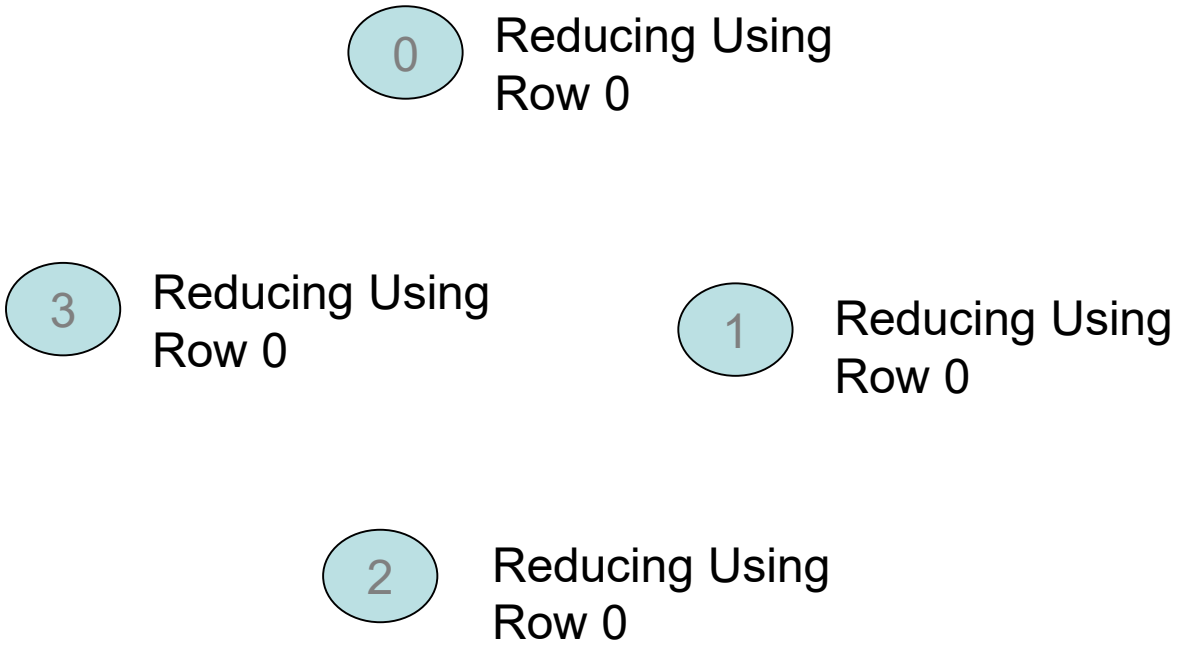
Communication Pattern



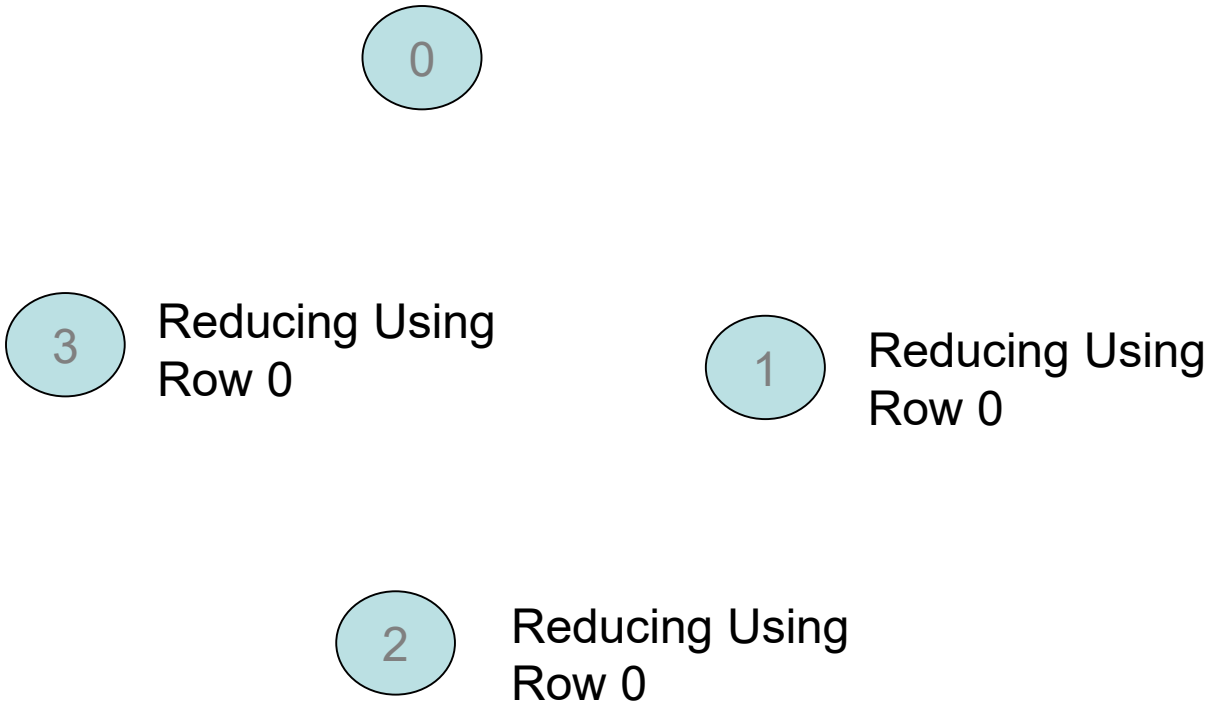
Communication Pattern



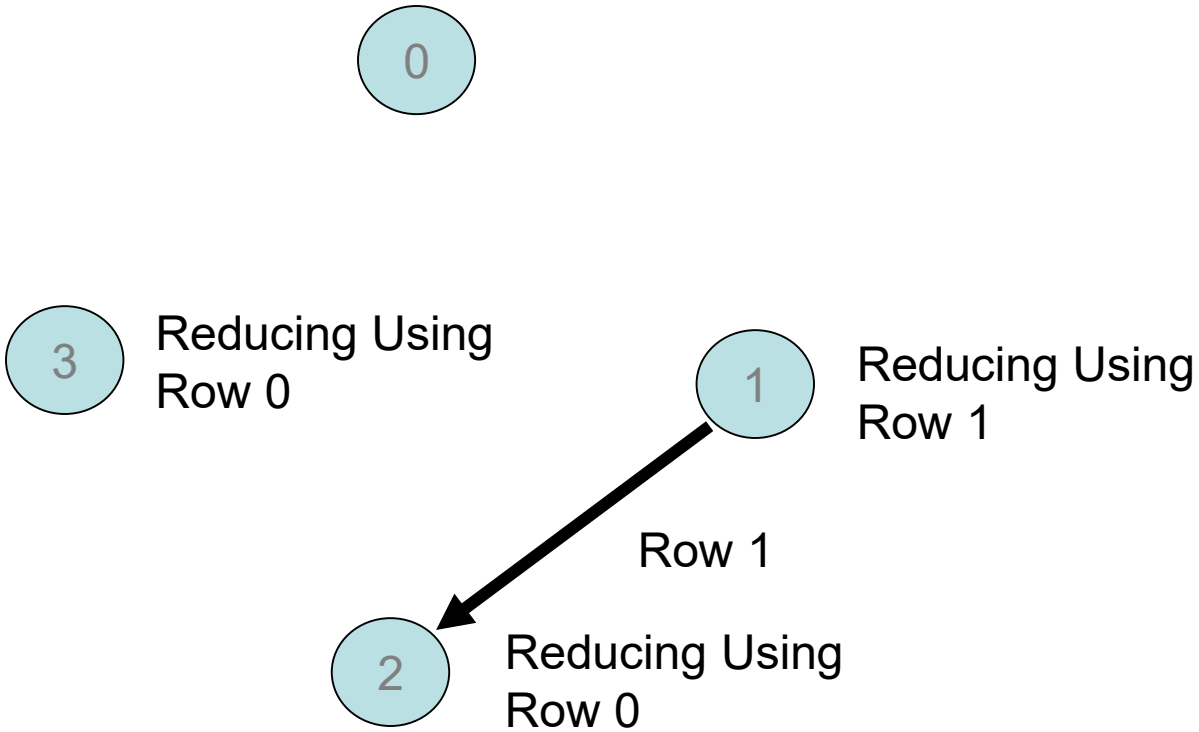
Communication Pattern



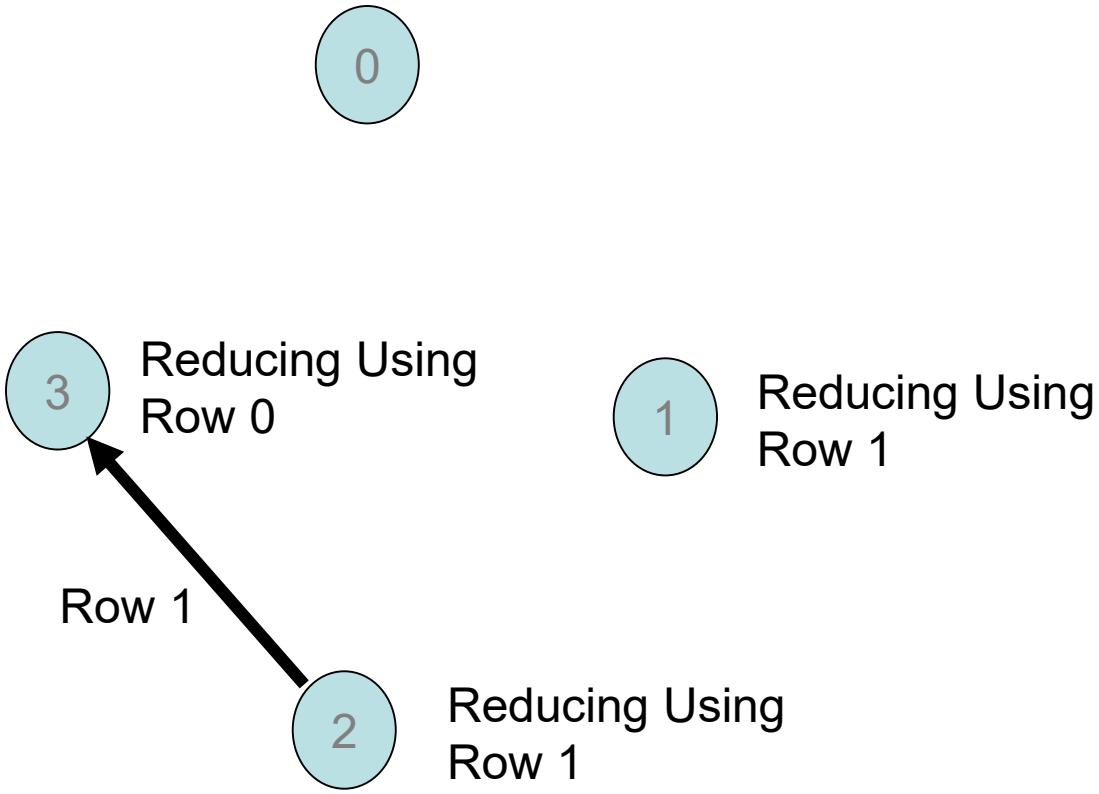
Communication Pattern



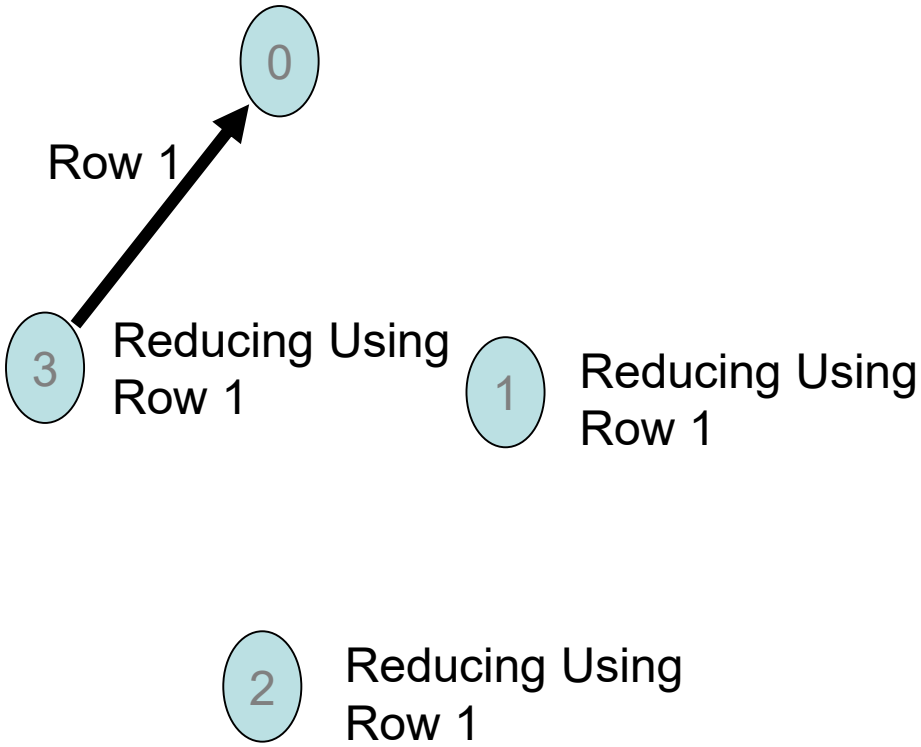
Communication Pattern



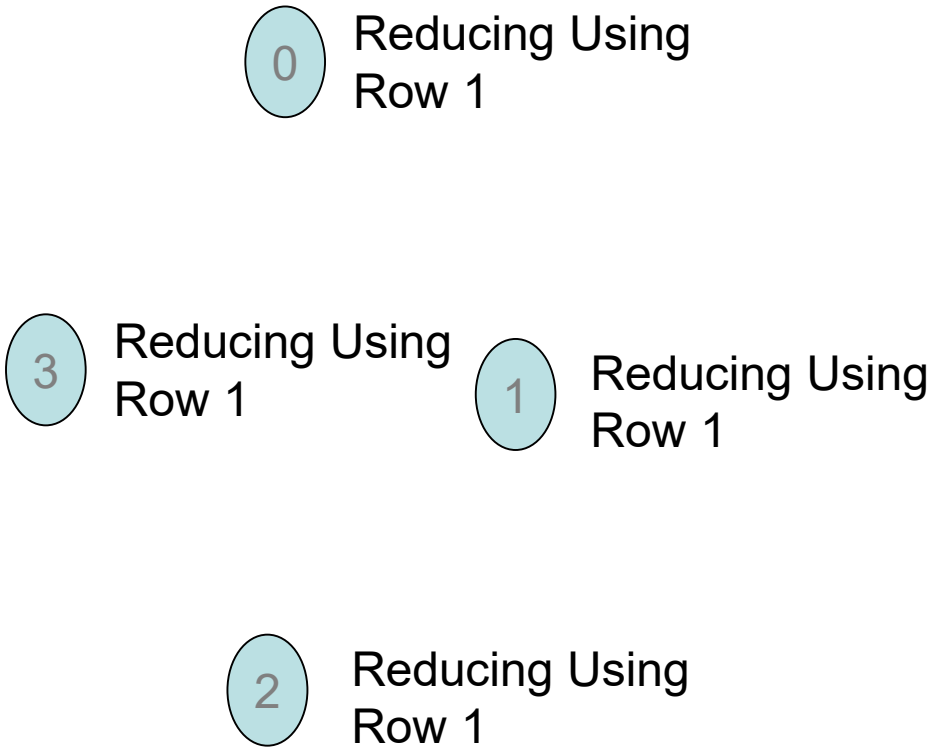
Communication Pattern



Communication Pattern



Communication Pattern



Pipelining

- By pipelining the flow of messages, the parallel algorithm has two advantages
  - First, it facilitates asynchronous execution: processes can reduce their portions of the augmented matrix as soon as the pivot rows are available
  - Second, it allows processes to effectively overlap communication time with computation time

Analysis

- Communication complexity
  - A point to point message in each iteration. Each message contains entire row.
  - $n(t_s+nt_w)$
- Computational complexity
  - $\Theta(n^3/p)$
- Computation overlapped by communication (only transmission not startup time).
- Total parallel time
  - $Tp= n^3/p+n$

Analysis

- Scalability
  - $W=n^3$
  - $T_0=pT_p-W$
  - $T_0=np$
  - $n^3=Knpt_s$
  - $n^2=kpt_s$
  - $n>=\sqrt{Kpt_s}$
  - Since $M(n) = n^2$, Scalability function is
  - $M(\sqrt{Kpt_s})/p=Kpts/p=Kts$
- Assuming n is large enough to ensure that message transmission time overlaps with computation time, this parallel system is perfectly scalable.

Sparse Systems

- Gaussian elimination not well-suited for sparse systems
- Coefficient matrix gradually fills with nonzero elements
  - Changes sparsity pattern -introduces non-zero entries which were originally zero
- Result
  - Increases storage requirements
  - Increases total operation count

Methods To Solve Linear Systems

- Direct solvers
  - Gaussian elimination
  - LU decomposition
- Iterative solvers
  - Stationary iterative solvers
    - Jacobi
    - Gauss-Seidel
    - Successive over-relaxation
  - Non-Stationary iterative methods
    - Generalized minimum residual (GMRES)
    - Conjugate gradient

References

- Ch12 of Quinn's book

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You