Pipelined, Parallel Gaussian Elimination

Thap Panitanarak
CS6260 Spring09

Western Michigan University

April 2, 2009
Consider: Summation of $n$ numbers on $p$-node hypercube ($p = 2^k$)

Data Distribution:
- Each $p$ takes $n/p$ elements

Step 1: Each $p$ (locally) computes its summation of $n/p$ numbers
- $n/p - 1$ additions

Step 2: Summation-Reduction of $p$ processors
- $2 \log p$ for communications and additions

Total Time:

$$T_p = \frac{n}{p} - 1 + 2 \log p$$

$$\approx \frac{n}{p} + 2 \log p$$
Scalability: Speedup & Efficiency

- Sequential Time: $T_s = n - 1 \approx n$
- Speedup:

$$S = \frac{T_s}{T_p} = \frac{n}{\frac{n}{p} + 2 \log p}$$

$$= \frac{np}{n + 2p \log p}$$

- Efficiency:

$$E = \frac{S}{p} = \frac{n}{n + 2p \log p}$$
Scalability: Speedup & Efficiency

Figure: Speedup vs. Number of Processors

- Linear
- n = 64
- n = 192
- n = 320
- n = 512
Table: Efficiency of various $p$ and $n$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$p = 1$</th>
<th>$p = 4$</th>
<th>$p = 8$</th>
<th>$p = 16$</th>
<th>$p = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1.0</td>
<td>0.80</td>
<td>0.57</td>
<td>0.33</td>
<td>0.17</td>
</tr>
<tr>
<td>192</td>
<td>1.0</td>
<td>0.92</td>
<td>0.80</td>
<td>0.60</td>
<td>0.38</td>
</tr>
<tr>
<td>320</td>
<td>1.0</td>
<td>0.95</td>
<td>0.87</td>
<td>0.71</td>
<td>0.50</td>
</tr>
<tr>
<td>512</td>
<td>1.0</td>
<td>0.97</td>
<td>0.91</td>
<td>0.80</td>
<td>0.62</td>
</tr>
</tbody>
</table>
Consider the previous efficiency,

\[
E = \frac{n}{n + 2p \log p} = \frac{1}{1 + \frac{2p \log p}{n}}
\]

\[
\frac{2p \log p}{n} \text{ fixed } \Rightarrow E \text{ fixed}
\]

\[
k = \frac{2p \log p}{n}
\]

\[
n = Kp \log p, \quad K = \frac{2}{k}
\]

Find \(K(E = .8): n = 64, p = 4 \rightarrow K = 8 \rightarrow n = 8p \log p\)

\[
p = 16 \rightarrow n = 8 \times 16 \log 16 = 512
\]

\[
p = 32 \rightarrow n = 8 \times 32 \log 32 = 1280\]
Consider speedup,

\[ S = \frac{T_s}{T_p} = \frac{R_s(n) + R_p(n)}{R_s(n) + R_p(n)/p + H(n, p)} \]

\[ = \frac{p(R_s + R_p)}{pR_s + R_p + pH} \]

\[ = \frac{p(R_s + R_p)}{R_s + R_p + (p - 1)R_s + pH} \]

\[ = \frac{pT_s}{T_s + T_o}, \quad T_o = (p - 1)R_s + pH \]

Substitute \( S \) in \( E \), we get,

\[ E = \frac{S}{p} = \frac{T_s}{T_s + T_o} = \frac{1}{1 + T_o/T_s} \]

\[ T_s = \frac{E}{1 - E} T_o \]
Scalability: Isoefficiency & Scalability Function

- To maintain constant efficiency, let $C = \frac{E}{1-E}$, then,

  $$T_s = \frac{E}{1-E} T_o = CT_o$$

- Isoefficiency Function: $T_s = CT_o$
- (Definition) $T_o$ is the total amount of time spent by all processors doing work not done by the sequential algorithm.
- Another way to find $T_o$, consider,

  $$T_o = (p - 1)R_s + pH + T_s - T_s$$
  $$= (p - 1)R_s + pH + (R_s + R_p) - T_s$$
  $$= pR_s + R_p + pH - T_s$$
  $$= p(R_s + R_p/p + pH) - T_s$$
  $$T_o = pT_p - T_s$$
Isoefficiency Function: \( T_s = C T_o \)

General Form: \( n = f(p) \)

Not sufficient? Different problems of the same problem size might require different size of memory

Example: Problems size \( n \), Linear Array(\( n \)) vs. Matrix(\( n^2 \))

\( M(n) \) - a memory size required of a problem size \( n \)

\( M(f(p))/p \) - a memory size increased per processor to keep the same level of efficiency

Scalability Function: \( M(f(p))/p \)
Example: Summation of $n$ numbers on $p$-node hypercube ($p = 2^k$)

- **Sequential:** $T_s = n$
- **Parallel:** $T_p = \frac{n}{p} + 2 \log p$

$$T_o = pT_p - T_s = p\left(\frac{n}{p} + 2 \log p\right) - n = 2p \log p$$

- **Isoefficiency:** $T_s = CT_o \Rightarrow n = 2Cp \log p$
- **Memory:** $M(n) = n$
- **Scalability:** $\frac{M(f(p))}{p} = \frac{2Cp \log p}{p} = 2C \log p$
Gaussian Elimination: Sequential vs. Parallel (Row Pivoting)

- Gaussian Elimination with Row Pivoting
  - Sequential Algorithm

```plaintext
for i = 0 to n - 1 do
    // Finding pivot row
    pmax = 0
    for j = i to n - 1 do
        if |a[j, i]| > pmax then
            pmax = |a[j, i]|
            prow = j
        end
    end
    Swap(i, prow)
    // Elimination
    for j = i + 1 to n - 1 do
        mul = a[j, i] / a[i, i]
        for k = i + 1 to n + 1 do
            a[j, k] = a[j, k] - a[i, k] * mul
        end
    end
end
```

- Total Time: $O(n^3)$
Gaussian Elimination: Sequential vs. Parallel (Row Pivoting)

- Gaussian Elimination with Row Pivoting
  - Parallel Algorithm (Hypercube)

```plaintext
for i = 0 to n - 1 do
  // Finding pivot row
  foreach PE do
    Find its pmax and prow of Col(i)
  end
  All-Reduce to find global pmax
  Swap(i,prow) between 2 PE's
  // Elimination
  PE that has Row(i) broadcasts Row(i)
  foreach PE do
    Compute mul
    Zero off their corresponding rows
  end
end
```

- Parallel Time in (Each Iteration):
  - Finding Pivot Row: Local $\rightarrow n/p$, Global $\rightarrow \log p$
  - Swap Row: $n \log p$ (can avoid by swapping only indexes)
  - Elimination: Broadcast $\rightarrow n \log p$, Zero off $\rightarrow n^2/p$
  - Total Time: $\frac{n^2}{p} + n \log p + n^2 \log p + \frac{n^3}{p} \approx n^2 \log p + \frac{n^3}{p}$
Gaussian Elimination: Sequential vs. Parallel (Column Pivoting)

- Gaussian Elimination with Column Pivoting
  - Sequential Algorithm

  ```
  for i = 0 to n - 1 do
    // Finding pivot row
    pmax = 0
    for j = i to n - 1 do
      if |a[i, j]| > pmax then
        pmax = |a[i, j]|
        pcol = j
      end
    end
    Swap(i, pcol)
  end
  
  // Elimination
  for j = i + 1 to n - 1 do
    mul = a[j, i] / a[i, i]
    for k = i + 1 to n + 1 do
      a[j, k] = a[j, k] - a[i, k] * mul
    end
  end
  ```

- Total Time: $O(n^3)$
Gaussian Elimination: Sequential vs. Parallel (Row Pivoting)

Gaussian Elimination with Row Pivoting

Parallel Algorithm (Ring)
- Define: Message(i) is the Row(i) and its $pcol$
- Initial Phase: PE containing Row(0) finds its $pcol$, sends Message(0) to the next PE, and also, performs elimination step 0
- Loop Phase: PE receiving Message(i) forwards it to the next PE (if the next PE does not perform elimination step $i$ yet), and performs elimination step $i$. If such PE contains Row($i+1$), it continues finding $pcol$ of Row($i+1$), and sends Message($i+1$) to the next PE before performs elimination step $i+1$.
- Final Phase: PE containing Row($n-1$) receives Message($n-2$), and performs elimination step $n-2$ (no sending/forwarding)

Parallel Time (in Each Iteration):
- Computation: Pivot Finding $\Rightarrow n$, Elimination $\Rightarrow \frac{n^2}{p}$
- Communication: Sending/Forwarding $\Rightarrow n$

All communication time can be overlapped with computation time except in Initial Phase
- Total Time: $\frac{n^3}{p} + n$
Scalability Analysis: Parallel without Pipelining - Row Pivoting

- Algorithm: Parallel Gaussian Elimination with Row Pivoting
- Topology: Hypercube
- Sequential: \( T_s = n^3 \)
- Parallel: \( T_p = \frac{n^3}{p} + n^2 \log p \)
- \( T_o = pT_p - T_s = n^2p \log p \)
- Isoefficiency: \( n^3 = Cn^2p \log p \Rightarrow n = Cp \log p \)
- Memory: \( M(n) = n^2 \)
- Scalability: \( \frac{M(f(p))}{p} = \frac{(Cp \log p)^2}{p} = C^2p \log^2 p \)
Scalability Analysis: Parallel with Pipelining - Column Pivoting

- Algorithm: Pipelined, Parallel Gaussian Elimination with Column Pivoting
- Topology: Ring
- Sequential: \( T_s = n^3 \)
- Parallel: \( T_p = \frac{n^3}{p} + n \)
- \( T_o = pT_p - T_s = np \)
- Isoefficiency: \( n^3 = Cnp \Rightarrow n = \sqrt{Cp} \)
- Memory: \( M(n) = n^2 \)
- Scalability: \( \frac{M(f(p))}{p} = \frac{(Cp\sqrt{Cp})^2}{p} = C \)
## Table: Gaussian Elimination Algorithms: Comparison

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Row Pivoting</th>
<th>Column Pivoting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Topology</td>
<td>Hypercube</td>
<td>Ring</td>
</tr>
<tr>
<td>Time</td>
<td>$\frac{n^3}{p} + n^2 \log p$</td>
<td>$\frac{n^3}{p} + n$</td>
</tr>
<tr>
<td>Scalability</td>
<td>$C^2 p \log^2 p$</td>
<td>$C$</td>
</tr>
</tbody>
</table>
Question

Find the execution time $T_p$, speedup $S$, efficiency $E$, isoefficiency fn and scalability fn of the summation of $n$ numbers using $p$-node hypercube

Answer

$T_s = n$, $T_p = \frac{n}{p} + 2 \log p$

$S = \frac{T_s}{T_p} = \frac{np}{n+2p \log p}$

$E = \frac{S}{p} = \frac{n}{n+2p \log p}$

$T_o = pT_p - T_s = 2p \log p$

isoefficiency fn: $T_s = CT_o \Rightarrow n = 2Cp \log p$

scalability fn: $\frac{M(f(p))}{p} = 2C \log p$
References

