RUNTIME ARRAY REDISTRIBUTION IN HPF PROGRAMS

Rajeev Thakur
Alok Choudhary
Geoffrey Fox

Dept. of Electrical and Computer Eng. and
Northeast Parallel Architectures Center
Syracuse University

May 24, 1994
Outline

- Introduction
- Need for Array Redistribution
- Block(m) to Cyclic
- Cyclic to Block(m)
- Cyclic(x) to Cyclic(y)
  - Special Case: \( x = k y \)
  - Special Case: \( y = k x \)
  - General Case
- Redistribution of Multidimensional Arrays
- Conclusions

Rajeev Thakur
Alok Choudhary
Geoffrey Fox
Arrays can be distributed as BLOCK(m) or CYCLIC(m).

In BLOCK(m), contiguous blocks of size m are distributed among the processors. (Note $m \times P \geq N$)

"BLOCK" $\Rightarrow$ BLOCK(⌊N/P⌋)

In CYCLIC(m), blocks of size m are distributed in a round-robin manner.

CYCLIC(m) commonly known as block-cyclic.

"CYCLIC" $\Rightarrow$ CYCLIC(1)
Need for Array Redistribution

- The distribution of an array can be changed anywhere in the program.
- REDISTRIBUTE directive.
- Not practical to write intrinsic and runtime libraries for all distributions.
- Distribution in subroutines may be different than distribution in main program.
- Significant performance improvements in some applications like 2D FFT, ADI etc.
Cyclic(x) to Cyclic(y)

(I) Special Case: \( x = ky \)

\[
\begin{array}{c|c|c}
  \text{p0} & \text{p1} \\
  \hline
  1 & 2 & 3 & 4 & 9 & 10 & 11 & 12 & 5 & 6 & 7 & 8 & 13 & 14 & 15 & 16 \\
\end{array}
\]

Cyclic(4)

\[
\begin{array}{c|c|c}
  \text{p0} & \text{p1} \\
  \hline
  1 & 2 & 5 & 6 & 9 & 10 & 13 & 14 & 3 & 4 & 7 & 8 & 11 & 12 & 15 & 16 \\
\end{array}
\]

Cyclic(2)

Communication Pattern:

- If \( k < P \), all-to-many (to \( k \) or \( k - 1 \) processors).
- If \( k \geq P \), all-to-all.
Cyclic(ky) to Cyclic(y) contd.

Send Phase

- The first $k$ sub-blocks of size $y$ have to be sent to the $k$ processors starting from $MOD(kp, P)$.

- This sequence is repeated for other sets of $k$ sub-blocks.

Receive Phase

- **Case 1**: $(k \leq P)$ and $(MOD(P, k) = 0)$
  The source processors of each set of $k$ blocks of size $y$ are $MOD(p/k + i(P/k), P)$, $(0 \leq i < k)$.

- **Case 2**: $(k > P)$ or $(MOD(P, k) \neq 0)$
  The source processor of each block $i$ $(0 \leq i \leq \lfloor L/y \rfloor - 1)$ of size $y$ is $MOD[(iP + p)/k, P]$. 
Cyclic($k_y$) to Cyclic($y$) contd.

Synchronous Method:

- Receive data from other processors.
- Local array is filled in sequence using data received from appropriate processor.
- Better use of cache as local array is scanned only once.
- Higher memory requirements.
- Higher idle time spent waiting to receive data from other processors.

Asynchronous Method:

- Receive data from one processor at a time and store it in the local array.
- Overlaps computation and communication.
- Lower memory requirements.
- Lower waiting time.
- Local array is scanned several times.
Synchronous v/s Asynchronous Method

Cyclic(4) to Cyclic(2)
Global array size: 1M
Machine: Intel Paragon
(II) Special Case: $y = kx$

\begin{align*}
\begin{array}{cccccccc}
p_0 & | & p_1 \\
1 & 2 & 5 & 6 & 9 & 10 & 13 & 14 \\
3 & 4 & 7 & 8 & 11 & 12 & 15 & 16 \\
\end{array}
\end{align*}

Cyclic(2)

\begin{align*}
\begin{array}{cccccccc}
p_0 & | & p_1 \\
1 & 2 & 3 & 4 & 9 & 10 & 11 & 12 \\
5 & 6 & 7 & 8 & 13 & 14 & 15 & 16 \\
\end{array}
\end{align*}

Cyclic(4)

Communication Pattern:

- If $k < P$, all-to-many (to $k$ or $k - 1$ processors).
- If $k \geq P$, all-to-all.
Cyclic($x$) to Cyclic($kx$) contd.

Send Phase

• **Case 1:** ($k \leq P$) and ($MOD(P, k) = 0$)
  The destination processors of each set of $k$
  blocks of size $x$ are $MOD(p/k + i(P/k), P)$,
  $(0 \leq i < k)$.

• **Case 2:** ($k > P$) or ($MOD(P, k) \neq 0$)
  The destination processor of each block $i$
  ($0 \leq i \leq \lceil L/x \rceil - 1$) of size $x$ is
  $MOD[(i P + p)/k, P]$.

Receive Phase

• The source processors of the first $k$
  sub-blocks of size $x$ are the $k$ processors
  starting from $MOD(k p, P)$.

• This sequence is repeated for other sets of $k$
  sub-blocks.
General Cyclic($x$) to Cyclic($y$)

Send Phase

- Destination processor of each local array element $i$ is
  $MOD[\{MOD(i - 1, x) + (P((i - 1)/x) + p)x + y\}/y - 1, P]$

Receive Phase

- Source processor of each local array element $i$ is
  $MOD[\{MOD(i - 1, y) + (P((i - 1)/y) + p)y + x\}/x - 1, P]$

Rajeev Thakur
Alok Choudhary
Geoffrey Fox
Multidimensional Arrays

1. **Shape Retaining**: Shape of the local array does not change. eg. (block, block) to (cyclic, cyclic)

   ![Diagram](block, block) ![Diagram](cyclic, cyclic)

2. **Shape Changing**: Shape of the local array changes. eg. (block, *) to (*, block)

   ![Diagram](block, *) ![Diagram](*, block)
Shape Retaining Redistribution

**Indirect Method:**

- Array is redistributed separately along each dimension. eg. (block,block) → (block,cyclic) → (cyclic,cyclic)
- Optimizations developed for 1D arrays can be used.

**Direct Method:**

- Data sent directly to destination processor.
- Optimizations developed for 1D arrays cannot be directly used.
- Different algorithms for each case.
Direct v/s Indirect Method

(block,block) to (cyclic,cyclic)
Global Array Size: 1K × 1K
Machine: Intel Paragon

Rajeev Thakur
Alok Choudhary
Geoffrey Fox
Related Work

- Gupta, Kaushik et. al. (1992): Send and receive processor and data sets for block and cyclic; virtual processor approach for block-cyclic.

- Wakatani and Wolfe (PARLE 94): Strip Mining Redistribution.

- Kalns and Ni (IPPS 94): Data mapping method to minimize communication during redistribution.

- Chatterjee et. al. (PPoPP 93); Stichnoth et. al. (JPDC, April 94); Gupta, Kaushik et. al. (ICPP 1993): Local addresses and communication sets for
  \[ A(l_1 : h_1 : s_1) = B(l_2 : h_2 : s_2) \]
Conclusions and Future Work

- Efficient algorithms for array redistribution.
- Practical and can be easily implemented.
- Useful for HPF runtime libraries
- Can be directly used in application programs written using message passing.
- Future work: Analytical model for estimating the cost of redistribution.