High Performance Fortran (HPF)

Source:

Chapter 7 of "Designing and building parallel programs" (Ian Foster, 1995)
HPF

• Designed by a forum from industry, government, universities
• Extends Fortran 90
• Used to parallelize "legacy programs"
  – Many scientific programs are written in Fortran
• Closest attempt to "automatic parallelization"
• Full parallelization (by a compiler) is too hard
## Data parallelism

- **Data parallelism**: same operation applied to different data elements in parallel
- **Data parallel program**: sequence of data parallel operations
- **Overall approach**:
  - Programmer does domain decomposition
  - Compiler partitions operations automatically
- **HPF supports only regular data (arrays), not irregular data (tree, sparse matrix)**

![Diagram of data parallelism](image)
Data parallelism - Concurrency

**Explicit parallel operations**

\[ A = B + C \]  

! Array operator: A, B, C are arrays

**Implicit parallelism**

\[
\begin{align*}
    & \text{do } i = 1, m \\
    & \quad \text{do } j = 1, n \\
    & \quad \quad A(i,j) = B(i,j) + C(i,j) \\
    & \quad \text{enddo} \\
    & \text{enddo}
\end{align*}
\]
Compiling data parallel programs

- Programs are translated automatically into parallel SPMD (Single Program Multiple Data) programs
- Each processor executes same program on subset of the data
- Owner computes rule:
  - Each processor owns subset of the data structures
  - Operations required for an element are executed by the owner
  - Each processor may read (but not modify) other elements
Example

real s, X(100), Y(100) ! s is scalar, X and Y are arrays

X = X * 3.0 ! Multiply each X(i) by 3.0

do i = 2,99
    Y(i) = (X(i-1) + X(i+1))/2 ! Communication required
endo

s = SUM(X) ! Communication required

X and Y are distributed (partitioned)
s is replicated on each machine
• X*3.0: no communication
• Loop: neighbor communication
• SUM: reduction operator
HPF primitives for data distribution

- Directives:

  PROCESSORS: shape & size of abstract processors
  ALIGN: align elements of different arrays
  DISTRIBUTE: distribute (partition) an array

- Directives affect performance of the program, not its result
Processors directive

!HPF$ PROCESSORS P(32)
!HPF$ PROCESSORS Q(4,8)

• Mapping of abstract to physical processors not specified in HPF (implementation-dependent)
Alignment directive

• Aligns an array with another array

• Specifies that specific elements should be mapped to the same processor

real A(50), B(50)
!HPF$ ALIGN A(I) WITH B(I)
    ! A(1) on same cpu as B(1), etc
!HPF$ ALIGN A(I) WITH B(I+2)
    ! A(1) on same cpu as B(3), etc
Distribution directive

• Specifies how elements should be partitioned among the local memories

• Each dimension can be distributed as follows:

  * no distribution
  BLOCK \(n\) block distribution
  CYCLIC \(n\) cyclic distribution
Figure 7.7 from Foster's book

CPU 0

(BLOCK, *)  (*, BLOCK)  (BLOCK, BLOCK)

(CYCLIC, *)  (CYCLIC, CYCLIC)  (CYCLIC, BLOCK)
Example: Successive Over relaxation (SOR)

Recall algorithm discussed in Introduction:

```c
float G[1:N, 1:M], Gnew[1:N, 1:M];
for (step = 0; step < NSTEPS; step++)
    for (i = 2; i < N; i++) /* update grid */
        for (j = 2; j < M; j++)
            Gnew[i,j] = f(G[i,j], G[i-1,j], G[i+1,j], G[i,j-1], G[i,j+1]);
    G = Gnew;
```
Parallel SOR with message passing

float G[lb-1:ub+1, 1:M], Gnew[lb-1:ub+1, 1:M];
for (step = 0; step < NSTEPS; step++)
    SEND(cpuid-1, G[lb]); /* send 1st row left */
    SEND(cpuid+1, G[ub]); /* send last row right */
    RECEIVE(cpuid-1, G[lb-1]); /* receive from left */
    RECEIVE(cpuid+1, G[ub+1]); /* receive from right */
for (i = lb; i <= ub; i++) /* update my rows */
    for (j = 2; j < M; j++)
        Gnew[i,j] = f(G[i,j], G[i-1,j], G[i+1,j], G[i,j-1], G[i,j+1]);
G = Gnew;
Finite differencing (~ SOR) in HPF

See Ian Foster, Program 7.2; uses convergence criterion instead of fixed number of steps

program hpf_finite_difference
!HPF$ PROCESSORS pr(4) ! use 4 CPUs
real X(100, 100), New(100, 100) ! data arrays
!HPF$ ALIGN New(:, :) WITH X(:, :) ! row-wise
!HPF$ DISTRIBUTE X(BLOCK, :) ONTO pr ! row-wise


diffmax = MAXVAL (ABS (New - X))
end
Changing the distribution

Use block distribution instead of row distribution

```fortran
program hpf_finite_difference
!HPF$ PROCESSORS pr(2,2) ! use 2x2 grid
real X(100, 100), New(100, 100) ! data arrays
!HPF$ ALIGN New(:,:,) WITH X(:,:,)
!HPF$ DISTRIBUTE X(BLOCK, BLOCK) ONTO pr ! block-wise


diffmax = MAXVAL (ABS (New-X))
end
```
Performance

Distribution affects

- Load balance
- Amount of communication

Example (communication costs):

```plaintext
!HPF$ PROCESSORS pr(3)
    integer A(8), B(8), C(8)
!HPF$ ALIGN B(:) WITH A(:)
!HPF$ DISTRIBUTE A(BLOCK) ONTO pr
!HPF$ DISTRIBUTE C(CYCLIC) ONTO pr
```
Figure 7.9 from Foster's book

\[
\text{FORALL}(i=1,7) \ A(i) = B(i) \quad (0 \ \text{communications})
\]

\[
\begin{array}{c|c|c}
A & B & C \\
\hline
1 & 2 & 3 \\
\end{array}
\]

\[
\text{FORALL}(i=1,7) \ A(i) = B(i+1) \quad (2 \ \text{communications})
\]

\[
\begin{array}{c|c|c}
A & B & C \\
\hline
1 & 2 & 3 \\
\end{array}
\]

\[
\text{FORALL}(i=1,7) \ A(i) = B((i+1)/2) \quad (5 \ \text{communications})
\]

\[
\begin{array}{c|c|c}
A & B & C \\
\hline
1 & 2 & 3 \\
\end{array}
\]

\[
\text{FORALL}(i=1,7) \ A(i) = C(i+1) \quad (5 \ \text{communications})
\]

\[
\begin{array}{c|c|c}
A & B & C \\
\hline
1 & 2 & 3 \\
\end{array}
\]
Historical Evaluation

Evaluation of HPF

• Problems:
  – Immature compiler technology
  – HPC community was impatient and started using MPI
  – No support for sparse array or irregular data structures
  – Obtaining *portable* performance was difficult
  – Performance tuning was difficult

• Large impact on parallel language design:
  – OpenMP (shared-memory standard)
  – High Productivity Computing Systems (HPCS) languages: Chapel (Cray), Fortress (Sun), X10 (IBM)
Summary

- High-level model
- User specifies data distribution
- Compiler generates parallel program + communication
- More restrictive than general message passing model (only data parallelism)
- Restricted to array-based data structures
- HPF programs will be easy to modify, enhances portability
- Changing data distribution only requires changing directives