Data Parallel Programming with HPF

Dr Alan D Simpson
Technical Director, EPCC
a.simpson@epcc.ed.ac.uk
+44 131 650 5120

Contents

Data Parallel Paradigm

- Data Parallel programming involves the processing and manipulation of large arrays
- Processors should (simultaneously) perform similar operations on different array elements
- DP programming languages characterised by:
  - single-threaded control
  - global address space
  - loosely synchronous processes
  - parallelism implied by operations applied to data
  - compiler directives
- Transfer low level details from programmer to compiler
  - Encourages wider use of parallelism

DP: Pros and Cons

- Advantages: Ease of Use
  - simple to write (no explicit message passing),
  - debug and maintain (single thread of control) and
  - port: both old codes and new (standard languages + set of intrinsic functions)
  - global address space allows easy access to all data
- Disadvantages: Flexibility and Control
  - forced to write in DP style, only suitable for certain applications
  - restricted control: distribution of data and work
  - harder to get top performance
  - *reliant on good compilers*

Applications

- Requirements
  - data must be in large arrays
  - similar operations on each element
  - independent operations
  - problem should be naturally load-balanced

- Examples
  - lattice physics, e.g. QCD,
  - atmospheric modelling,
  - cellular automata,
  - image processing

Data Parallel Languages

- Many data parallel languages implemented:
  - Fortran-Plus, MP Fortran, CM Fortran, C*, CRAFT, Fortran D, Vienna Fortran
- Languages expressed data parallel operations differently

- Problems:
  - too many languages and dialects
  - languages were machine specific

- Result:
  - lack of portability (users tied to one manufacturer and prospective users unwilling to learn)
  - need portable standard ... HPF

HPF: High Performance Fortran

- Need for a Data Parallel standard well recognised
- High Performance Fortran Forum formed in 1991
- HPF Specification V1.0 published in 1993
- HPF Forum aimed to define a language that offered:
  - data parallel programming (single threaded, global name space, loosely synchronous parallel computation)
  - top performance on all machines (with code tuning)
- These goals are addressed in the form: Fortran plus
  - compiler directives: data alignment and distribution
  - new intrinsics, library functions and language constructs
- Subsidiary goals:
  - simplicity and portability (existing code and of new code)
  - ease of compiler implementation

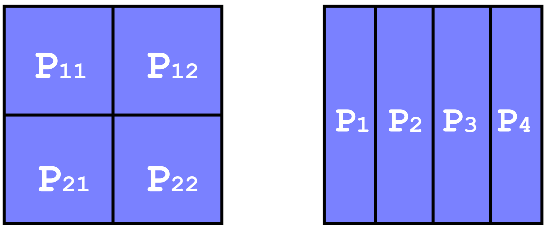
DP Building Blocks

- Data distribution
- Array syntax
- Conditional operations
- Intrinsic functions
  - Shifts, reductions, scans, …
- Libraries
- Many of these are already part of standard Fortran
  - …and so are inherited by HPF

Data Distribution

epcc

- Data distribution is the major new feature in Data Parallel languages like HPF
- Data distributions controlled by language constructs and directives, e.g., HPF `DISTRIBUTE` directive



- *More on this topic later in the lecture*

Elemental Operations

epcc

- Whole arrays used in expressions

$$\begin{bmatrix} 2 & 12 \\ 8 & 25 \\ 18 & 42 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} * \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix}$$

`c = a * b`

- Mathematical functions on array-valued arguments
- Makes code more compact and readable
- Array operations in parallel

Fortran Array Syntax

- Arrays syntax allows arrays to become basic units

FORTRAN 77

```
REAL a(100), b(100)
DO 1 i = 1,100
a(i) = 2.0
1 b(i) = b(i)*a(i)
```

HPF/Fortran 2008

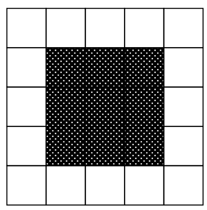
```
REAL,DIMENSION(100):: a,b

a = 2.0
b = b * a
```

- This tends to make HPF code quite compact

Array Sections

- Features to allow some operations to act on only a subset of array elements
  - regular patterns



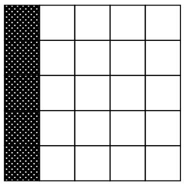
- Array sections and array constants can be defined through triplet notation in Fortran/HPF:

```
start index : end index : stride
```

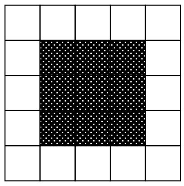
- For example:

```
REAL, DIMENSION(1024) :: a
a(1:1024) = 1.0                  ! same as a = 1.0
a(:1024:2) = 2.0                 ! a(1)=a(3)=...= 2.0
a = (/ (i,i=1,1024) /)           ! a(1)=1,a(2)=2,...
a = (/ (i,i=1,4096,4) /)         ! a(1)=1,a(2)=5,...
```

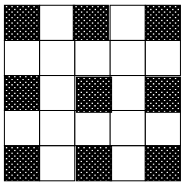
HPF/Fortran Array Sections



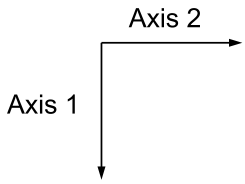
`a(:,1)`



`a(2:4,2:4)`



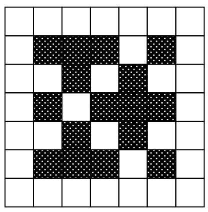
`a(1::2,1::2)`



Conditional Operations

- Operations on subset of array, subject to some conditional mask
  - arbitrary patterns specified
- In Fortran and HPF, **WHERE** uses a logical mask
  - equivalent to `IF' on each element
  - there is also an **ELSE WHERE** clause

```
REAL, DIMENSION(7,7) :: a, b
...
WHERE (a != 0) b = 1/a
   ...
WHERE (a > 2.0)
   b = a * 9.0
ELSE WHERE
   b = a / 3.0
END WHERE
```



Shift Operations

- Move whole array in particular direction
- For example, a cyclic shift one place to the right would produce the following result

1	2	3	4
5	6	7	8

Source

→

4	1	2	3
8	5	6	7

Result

- Also "end-off" shift; provide boundary conditions
- Many efficient parallel algorithms can be implemented in terms of shifts
- Useful for, e.g., image processing, cellular automata

CSHIFT in Fortran/HPF

- Regular movement of data

Axis 2

Axis 1

src

1	2	3
4	5	6
7	8	9

→

res

3	1	2
6	4	5
9	7	8

```
res = CSHIFT(src, SHIFT=-1, DIM=2)
```

which is roughly equivalent to

```
res(i,j) = src(i,j-1)
```

Reduction Operations

- Produce one result from many array elements, some global property of the array
- Provided to enable the programmer to, e.g.
  - find the largest/smallest value in an array
  - calculate the sum/product of all the elements of an array
  - logical AND, OR, EOR
  - count number of true elements in a logical array
- Very important in data parallel programming and often implemented efficiently

Reduction Operations

- Typically involve every element → global operation

```
REAL :: a(1024), b(4,1024)
scalar = SUM(a)           !sum of all elements
scalar = COUNT(a==0)      !number of zero elements
a = PRODUCT(b, DIM=1)     !product in first dim
scalar = MAXVAL(a, MASK=a.LT.0)
```

- Logical Reductions: **ANY, ALL**

```
LOGICAL a(n)
REAL, DIMENSION(n) :: b, c
IF (ALL(a))               ! Global AND
IF (ALL(b == c))          ! true if all elements equal
IF (ANY(a))               ! Global OR
IF (ANY(b < 0.0))         ! true if any element < 0.0
```

Other Operations

Intrinsic Functions and Libraries

- High level parallel constructs
- Useful for constructing parallel algorithms; saves reinventing the wheel.
- Efficient implementations exist: provided by manufacturer to make best use of host machine
- Examples
  - scan operations:
    - scan through arrays and produce some cumulative result
  - generalised communications:
    - user defined mapping and combination of elements to form a result

Data Parallel Summary

- Data parallel programming
  - single-threaded control
  - global address space
  - loosely synchronous processes
  - parallelism implied by operations applied to data
  - compiler directives
- Data parallel building blocks
  - data distribution
  - array syntax and conditional operations
  - intrinsic functions and libraries
- Fortran is an excellent basis for a data parallel programming language, such as HPF
  - widely used by computational scientists
  - contains many key data parallel building blocks

Data Distribution in HPF

- Data Parallel Programming involves the processing and manipulation of large arrays across many processors
- Generally, each processor has its own local memory
  - large arrays should be distributed across all processors
- Arrays distributed in different ways depending on how they are to be used
  - chose distribution which maximises the ratio of local work to communication
- Current compilers are not sophisticated enough to automatically distribute arrays over processors
- HPF has directives to control data mapping and help compiler
  - can reduce communication overhead and produce more efficient code
- Data distribution is the key new feature of HPF

DISTRIBUTE Directive

- Compiler directive to specify type of distribution to use
- Specifies the distribution for each dimension of array
- Formats:

```
!HPF$ DISTRIBUTE a(distribution)
```

or

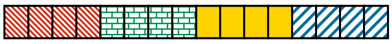
```
!HPF$ DISTRIBUTE (distribution) :: a, b
```

  - **!HPF$** is used for all HPF compiler directives --- this is a comment to non-HPF compilers
  - ***distribution*** is a comma-separated list of the distributions for each array dimension

BLOCK Distribution

```
REAL, DIMENSION(16) :: a
!HPF$ DISTRIBUTE (BLOCK) :: a

   P1     P2     P3     P4
    1      5      9     13
    2      6     10     14
    3      7     11     15
    4      8     12     16
```



- *Example:* regular domain decomposition (Game of Life)

CYCLIC Distribution

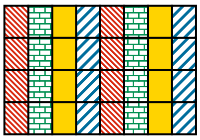
```
REAL, DIMENSION(16) :: a
!HPF$ DISTRIBUTE (CYCLIC) :: a

   P1     P2     P3     P4
    1      2      3      4
    5      6      7      8
    9     10     11     12
   13     14     15     16
```

- *Example:* ray tracing; Mandelbrot

Degenerate Distribution

```
REAL, DIMENSION(4, 8) :: a
!HPF$ DISTRIBUTE (*, CYCLIC) :: a
```



- First dimension can be called *degenerate, serial, collapsed,...*
- *Example*: serial dimension is an array of attributes associated with a point on the grid

General Distributions

- **`CYCLIC(n)`**: array split into contiguous chunks of size **`n`** and then distributed cyclically

```
REAL, DIMENSION(16) :: a
!HPF$ DISTRIBUTE (CYCLIC(2)) :: a
```

- **`BLOCK(n)`**: similar to **`CYCLIC(n)`** except only 1 chunk per (abstract) processor
  - for above example of 16 elements on 4 processors, $n \geq 16 / 4$
  - this means **`BLOCK(4), BLOCK(5)`**, etc., would be OK, while **`BLOCK(3)`** would be *non-conforming*

Example 1: Distributed

```
PROGRAM disfunc
IMPLICIT NONE
INTEGER :: i
INTEGER, PARAMETER :: c1=5, c2=10
INTEGER, DIMENSION(10000) :: x, y

! Distribute data and hence work
!HPF$ DISTRIBUTE (CYCLIC) :: x, y
! CYCLIC for load balance

x = (/ (i, i = 1, 10000) /)

y(1:400)= x(1:400)*x(1:400)*x(1:400)+ c1
y(401:1000)= x(401:1000)*x(401:1000)+ c2
y(1001:10000)= x(1001:10000)
END
```


Data Distribution: Summary

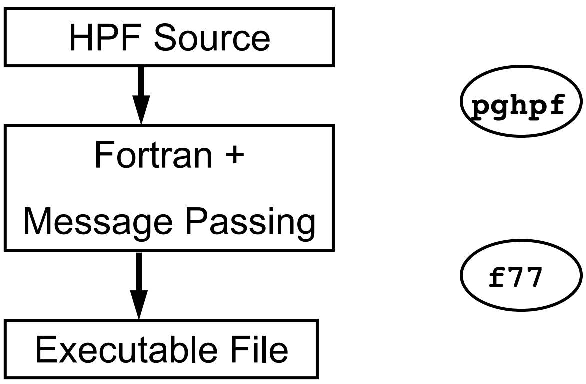
- Data distribution is one of the major HPF extensions to Fortran
- **`DISTRIBUTE`** directive indicates type of distribution appropriate for each array dimension
- Data distribution helps compiler with work sharing
- Choosing the appropriate distribution can make a significant difference to performance

The Portland Group Compiler

pghpf

- Parallelising translator
- Compiles to SPMD FORTRAN + message passing
- Works on a variety of systems:
  - CRAY, Fujitsu, HP, IBM, SGI, Sun, RISC Workstations, …
- Supports various communication protocols: MPI, PVM, RPM (optimised PVM), SHMEM, ...
- Full HPF 1.1 functionality with some HPF 2.0 + approved extensions

Compilation Stages



Executable file for the target system

Summary

- High Performance Fortran is a standardised data parallel Fortran
  - defined by HPF Forum (HPFF)
- HPF uses key features of Fortan
  - with extensions such as: data distribution, additional intrinsics and library functions,…
- The Portland Group compiler: pghpf
  - good functionality and portability

HPF in CP Lab

- Using the PGI HPF compiler, **`pghpf`**
- Compile command:

```
pghpf -Mautopar -lrt -o hello hello.f90
```

- Interactive run command:

```
./hello -pghpf -np 4
```

- *Note:* **`pghpf`** recognises **`.f90`** as free format Fortran
- Do ***NOT*** use **`.hpf`** as **`pghpf`** then assumes fixed format

Batch Jobs

- Running batch jobs via Sun Grid Engine:

  `qsub -cwd -pe mpi 4 ./hello.sge`

  – where the script file **`hello.sge`** should contain the following line:

  `./hello -pghpf -np $NSLOTS`

  – the environment variable **`NSLOTS`** is set equal to the number of processors requested by qsub

  – output will appear in a file called **`hello.sge.oxxxx`**

- There is a template file **`hpfbatch.sge`** which you can rename

  `cp hpfbatch.sge hello.sge`