University
of Southampton

High Performance Fortran Translator

Dr Bryan Carpenter

Southampton University Computing Services
University of Southampton
Southampton SO9 5NH
United Kingdom

dbc@uk.ac.soton.ecs

Contents

- Background
  - Why HPF?
  - HPFF
- HPF overview
  - Parallel computation
  - Distributed data
- Southampton work
  - PUMA: ADAPT and ADLIB
  - JISC and SERC projects
  - Mapping of computation
  - Run-time library
- Conclusion

1.1 Why HPF?

Some extant HPC architectures:

- Single processor vector
- Single instruction stream, distributed data (SIMD)
- Multiple processor (MIMD)
  - Message-passing
  - Remote memory access
  - Shared memory

Standardisation efforts within some subclasses (eg MPI), but still great diversity.

Parallelisation of sequential languages difficult/intractable.

HPF generalises SIMD dialects of Fortran—less baggage than message-passing or general shared memory. "Bulk-synchronous, parallel random access" model. Adds some of the flexibility of MIMD model.

Can be compiled to any of the above architectures ;-).

1.2 The High Performance Fortran Forum

Started meeting March 1992. Participation of 40+ organisations, including designers of Fortran D, Vienna Fortran, and manufacturers of parallel computers (Cray, DEC, Fujitsu, HP, IBM, Intel, Maspar, Meiko, nCube, Sun, Thinking Machines. . .).

Language is a development of SIMD Fortran dialects such as Connection Machine Fortran, and distributed memory dialects such as Fortran D and Vienna Fortran.

Standard distributed in May '93, published in September.

Still no full Subset implementations available :-(.

2. HPF Overview

HPF extends Fortran in two ways:

- It adds constructs which specify parallel execution of computations.
- It adds directives which specify how program data is "aligned" and distributed across multiple memories.

2.1 Parallel Computation

HPF includes all the "array syntax" and "array intrinsics" of Fortran 90. It adds a **FORALL** construct.

Array Syntax

Examples of Fortran 90 syntax:

```
REAL X (10), Y (10), LEN (10)

LEN = SQRT(X**2 + Y**2)
```

All 10 lengths can be computed in parallel.

```
REAL A (10, 10), B (10, 10), C (10, 10)

DO I = 1, 10
  DO J = 1, 10
    A (I, J) = DOT_PRODUCT (A (I, :), B (:, J))
  END DO
END DO
```

The intrinsic **DOT_PRODUCT** can be computed by a parallel algorithm (but **DO** loops remain sequential). Note syntax for "array sections".

FORALL

By combining the F90 array intrinsics with section subscripts (and “masking”), possible to express many parallel algorithms.

But, array syntax rapidly becomes clumsy for writing practical parallel programs. Can use **FORALL** instead

```
FORALL (I = 1 : N, J = 1 : N)
  U (I, J) = (U (I, J - 1) + U (I, J + 1) +
                U (I - 1, J) + U (I + 1, J)) / 4
END FORALL
```

(a parallel iterative solver).

```
FORALL (I = 1 : N, J = 1 : N)
  A (I, J) = DOT_PRODUCT (A (I, :), B (:, J))
END FORALL
```

(fully parallel version of earlier example).

Example: LU Decomposition

```
REAL A (N, N), B (N, N)
REAL L_COL (N), U_ROW (N)

DO R = 1, N - 1
  R1 = R + 1
  FORALL (I = R : N)
    L_COL (I) = A (I, R)
  END FORALL
  FORALL (J = R1 : N)
    U_ROW (J) = A (R, J) / L_COL (R)
    A (R, J)  = U_ROW (J)
  END FORALL
  FORALL (I = R1 : N, J = R1 : N)
    A (I, J) = A (I, J) - L_COL (I) * U_ROW (J)
  END FORALL
END DO
```

2.2 Distributed Data

In practise programmer has to take some responsibility for how the program is divided across the parallel computer.

Done in HPF by specifying how program data is distributed across local memories.

From these clues compiler can (hopefully) choose sensible places to perform individual computations.

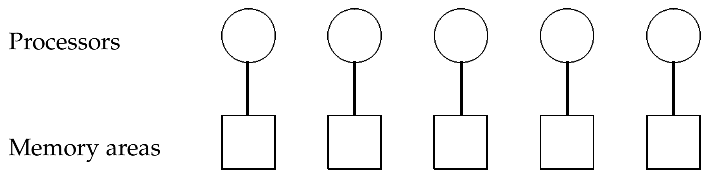

Figure 1: Idealised picture of a distributed memory para llel computer

Bad Placements

```
FORALL (I = 1 : 3)  C (I) = A (I) + B (I)
```

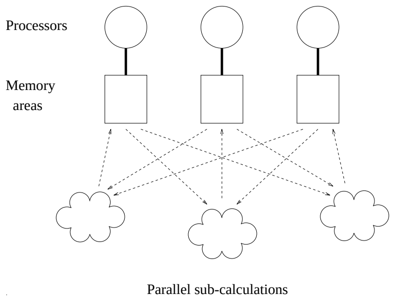

Figure 2: A data distribution leading to excessive communication

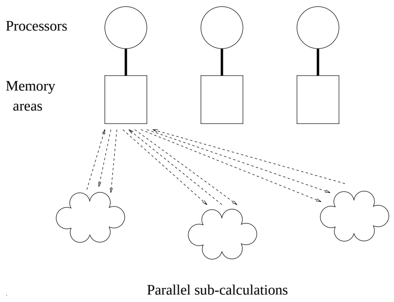

Figure 3: A data distribution leading to poor load balancing

University
of Southampton

Good Placement

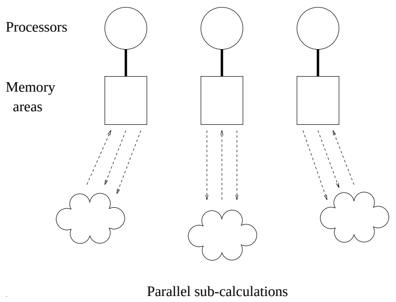

Figure 4: An ideal data distribution

Could be achieved by following the declarations of **A**, **B**, **C** with the directives

```
!HPF PROCESSORS P (3)
!HPF DISTRIBUTE C (BLOCK) ONTO P
!HPF ALIGN A (I) WITH C (I)
!HPF ALIGN B (I) WITH C (I)
```

More Complex Distributions

```
PROCESSORS P (3, 3)
REAL A (6, 6)
!HPF DISTRIBUTE A (BLOCK, BLOCK) ONTO P
```

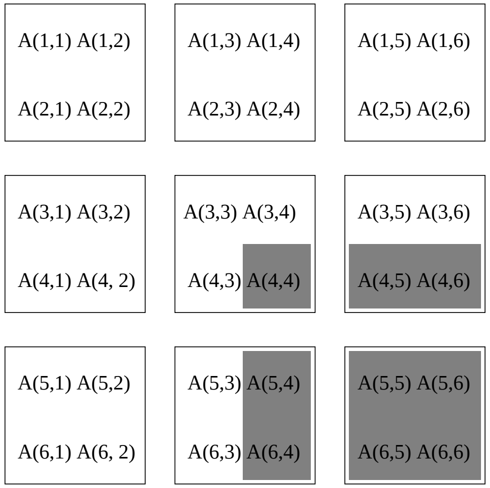

Figure 5: Block distribution

Cyclic distribution

```
!HPF DISTRIBUTE A (CYCLIC, CYCLIC) ONTO P
```

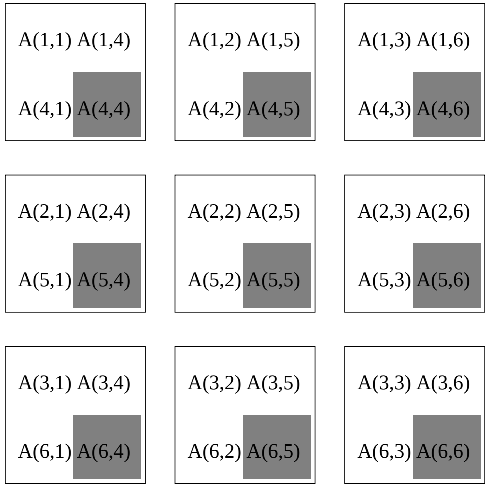

Figure 6: Cyclic distribution

More Complex Alignments

```
REAL A (4, 4)
REAL L_COL (4), U_ROW (4)

!HPF ALIGN L_COL (I) WITH A (I, *)
!HPF ALIGN U_ROW (J) WITH A (*, J)
```

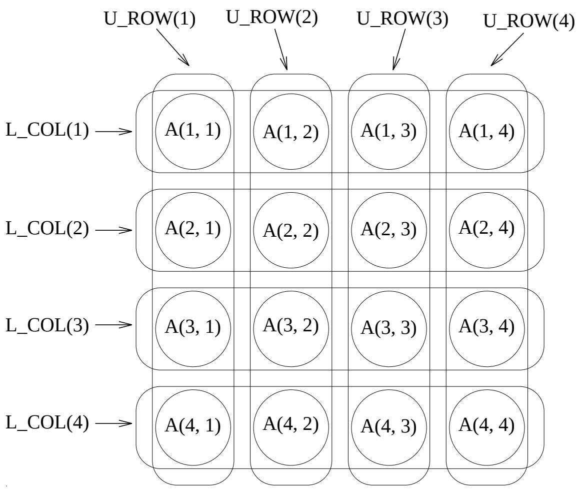

Figure 7: Alignment of the arrays for LU decomposition example

University
of Southampton

Templates

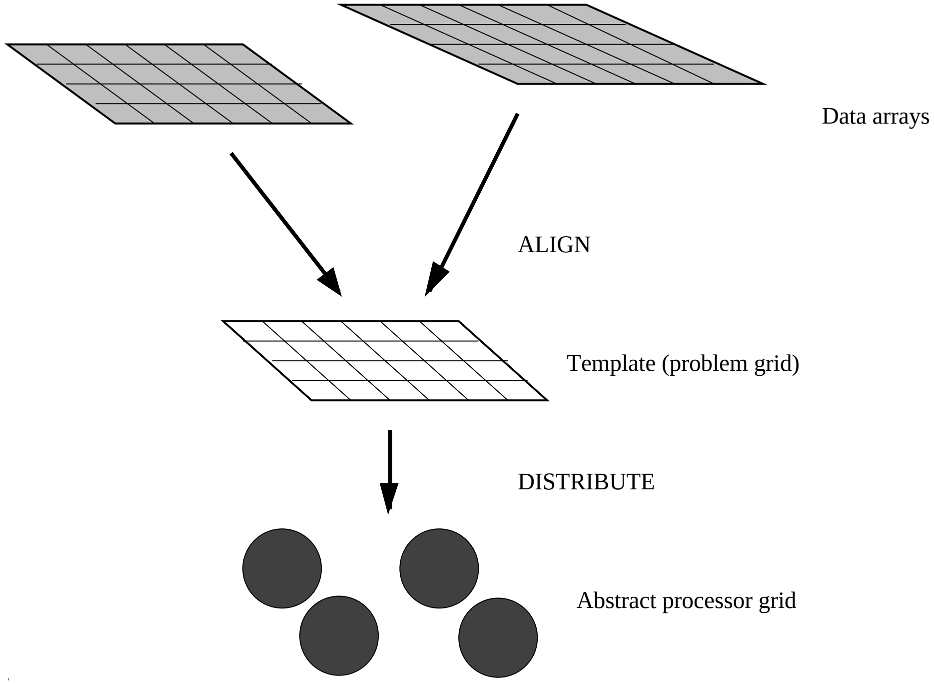

Figure 8: The stages of data mapping in HPF.

3. Southampton Work

3.1 PUMA: ADAPT and ADLIB

Since 1989, ECS has been involved in research and development on *ADAPT* ("Array Distribution Automatic Parallelisation Tool").

Tool for automatically transforming Fortran 90 + data-distribution-declarations to code for message-passing computers.

Work started in Esprit PUMA project. Initiated by John Merlin. Predates HPF by several years.

F90 converted to F77 + calls to run-time library: *ADLIB*.

3.2 JISC and SERC projects

This year Computing Services received 1MY JISC funding, to generalise ADAPT/ADLIB system to Subset HPF. Output will be a public domain translation system.

Subsequently ECS granted 2MY SERC funding to work on optimisation issues. NAG are “heavy uncle”.

3.3 Mapping of Computation

A central problem in HPF translation is mapping of intermediate results, ie, subexpressions. Equivalent to mapping computations to processors. Especially difficult in **FORALL**.

Are developing schemes applicable to rather general **FORALL** statements. Attempt to distribute any index *whose range doesn't depend on other indices*.

Intermediates mapped to templates derived from templates of named program variables.

No reliance on "Owner-Computes" type rules—assignment not treated specially.

Mapping scheme specified in Prolog.

Example Mapping of Intermediate Results

Input code. . .

```
DIMENSION B (n), A (n, n)

FORALL (I = 1 : N, J = 1 : N)
  A (I, J) = FUN1 (I) + (B (J) + FUN2 (J))
ENDFORALL
```

"Translation". . .

```
ALIGN TEMP1 (I) WITH A (I, *)
ALIGN TEMP2 (J) WITH B (J)
ALIGN TEMP3 (J) WITH A (*, J)

FORALL (I = 1 : N)  TEMP1 (I) = FUN1 (I)
FORALL (J = 1 : N)  TEMP2 (J) = B (J) + FUN2 (J)
TEMP3 = TEMP2                              ! remapping
A     = TEMP1 + TEMP3
```

Run-time Library

Schematic form of principal calls. . .

Remapping can occur at either variable reference or function reference nodes.

```
remap(result_data, target_mapping, input_data).
```

target_mapping is any legitimate HPF mapping for the input data. Optimised entry points for shifts, transposes etc.

At variable reference nodes also have

```
gather(result_data, hpf_array,
       subscript_1, subscript_2, ...)
```

Optimised entry points for taking array sections, etc. Simiarly, **scatter**.

4. Conclusion

Plan to make prototype version of HPF translation system available by the end of this year, under the JISC HPF translation project (B. Carpenter).

Results of concurrent and subsequent research on optimisation strategies, funded by SERC project (J. Merlin) will be incorporated later.