University
of Southampton

HPF Translation Project

Dr Bryan Carpenter

High-Performance Computing Centre
University of Southampton
Southampton SO9 5NH
United Kingdom

dbc@uk.ac.soton.ecs

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. . .

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 main 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.

University of Southampton

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 (B (I, :), C (:, 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 (B (I, :), C (:, J))
END FORALL
```

(fully parallel version of earlier example).

University of Southampton

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.

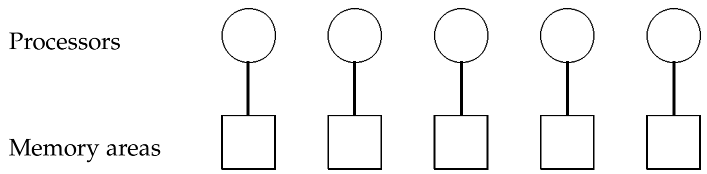


Figure 1: Idealised picture of a distributed memory parallel computer

Bad Placements

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

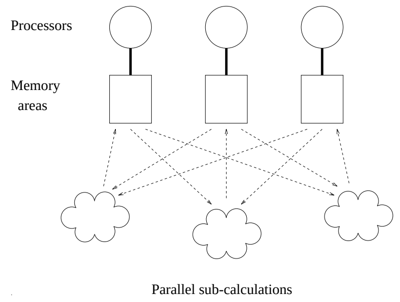


Figure 2: A data distribution leading to excessive communication

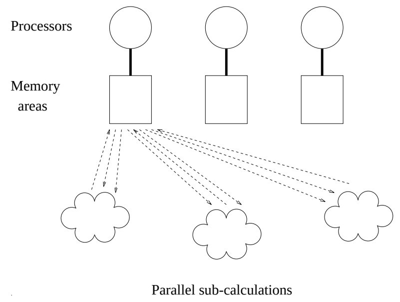


Figure 3: A data distribution leading to poor load balancing

University of Southampton

Good Placement

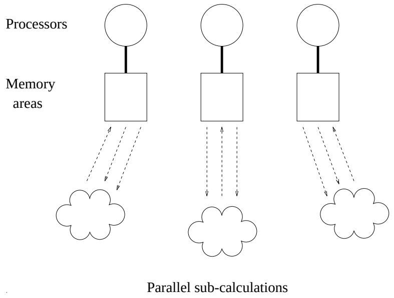


Figure 4: An ideal data distribution

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
```

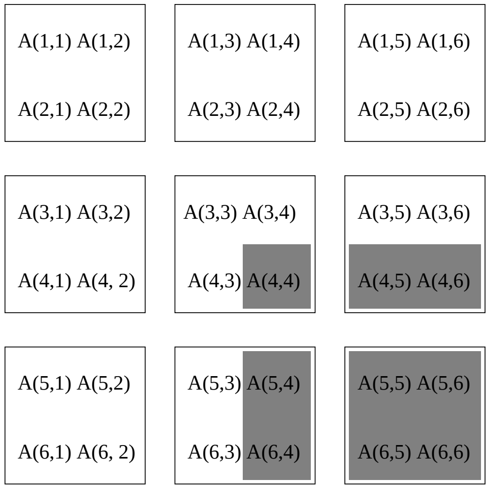


Figure 5: Block distribution

Cyclic distribution

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

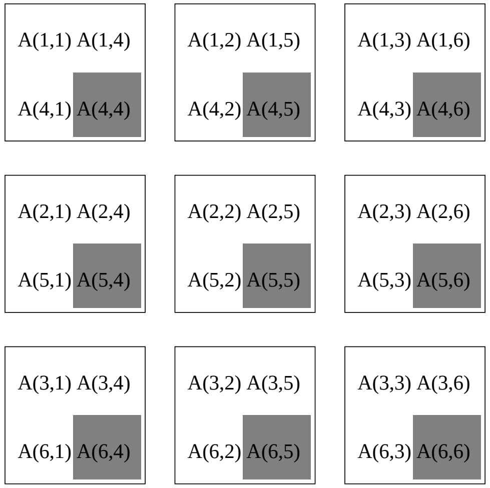
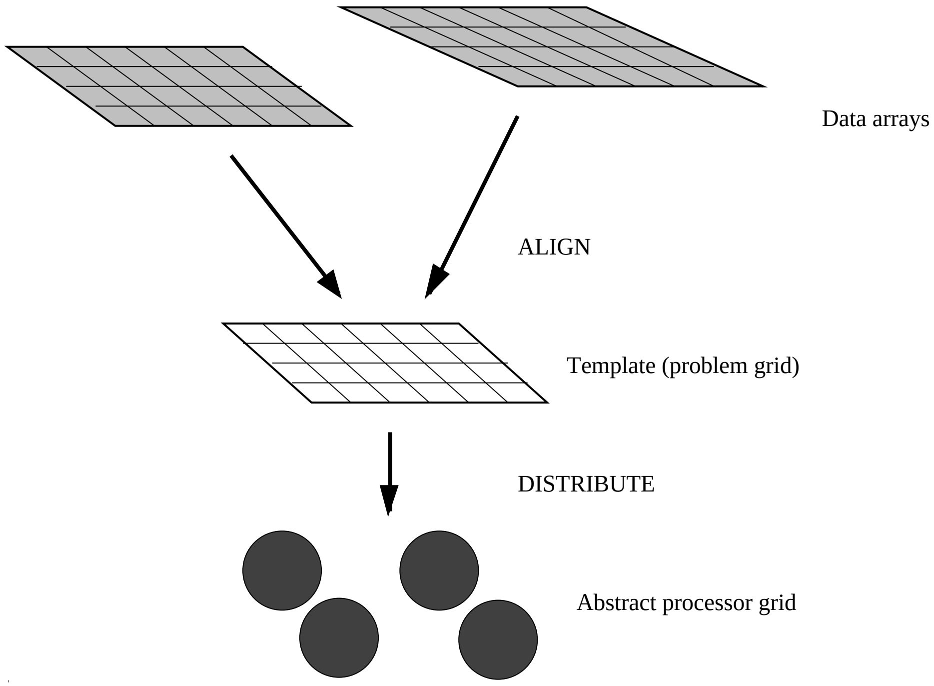


Figure 6: Cyclic distribution

University
of Southampton

Templates



3. Southampton HPF Project

PUMA: ADAPT, ADLIB, ACL

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*.

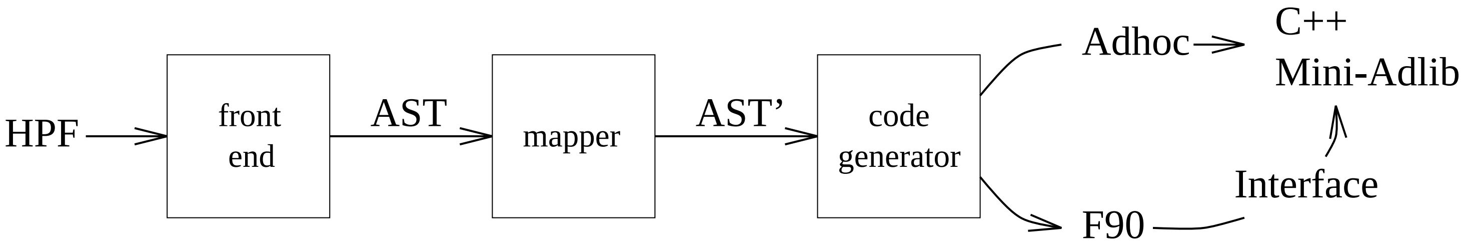
ACL was an occam-based collective communication library, also developed in the PUMA project.

JISC NTI and SERC projects

This year SUCS 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”.

Overview of Translation System



- *front end*—syntactic and static semantic analysis of the input HPF.
- *mapper*—"expression mapping".
- *code generator*—convert AST to Adhoc-based intermediate form with explicit communication and distributed loops, then target language.
- *Adhoc*—a high-level target language, close to C++.
- *Mini-Adlib*—C++ interface to a minimal RTL.

Adhoc

```
      REAL A (10, 10), B (10)
!HPF$ DISTRIBUTE A (BLOCK, BLOCK)
!HPF$ ALIGN B WITH A ( : , *)

      FORALL (I = 1 : 10, J = 1 : 10)
        A (I, J) = B (I) + FUN (J)
      END FORALL
```

could be translated as

```
range i_a(10 #blk p_1), j_a(10 #blk p_2) ;
array float a(i_a, j_a) ;
array float b(i_a) ;

where(i_a)
  where(j_a)
    a = b + fun(j_a) ;
```

Communication in Adhoc

```
      REAL U (12)
!HPF$ DISTRIBUTE U (BLOCK)

      FORALL (I = 2 : 11)
        U (I) = (U (I - 1) + U (I + 1)) / 2
      END FORALL
```

could translate to

```
  range i_u(12 # p_1) ;
  array float u [i_u] ;
  range i_interior(10 : base 1 : i_u) ;
  array float n [i_interior], p [i_interior] ;

  remap(n, u [10 : base 0]) ; remap(p, u [10 : base 2]) ;
  where(i_interior)
    u = (n + p) / 2 ;
```

Adhoc Communication Primitives

remap copies data between two arrays of the same shape—they can have *any* legitimate HPF (or Adhoc) mapping.

Also provides *gather* and *scatter* primitives to handle communications implied by non-linear subscripts.

A couple of communication primitives are needed for IO, and some extra support is needed to implement F90 array intrinsics.

Mini-Adlib

Minimal “Run-time library” for Adhoc.

Mini-Adlib provides classes for “process arrays” and ranges, and template classes for arrays. Range objects have built in “iterator” functionality, implementing the distributed loops of Adhoc.

Free use made of operator overloading (eg, function application “operator” used for array subscripting).

(Less elegant) Fortran 90 interfaces will also be provided.

Run-time support for Fortran IO will be lifted from existing sequential Fortran translators.

Transcription of Adhoc to C++

```
range i_a(10 #blk p_1), j_a(10 #blk p_2) ;
array float a(i_a, j_a) ;
array float b(i_a) ;

where(i_a)
  where(j_a)
    a = b + fun(j_a) ;
```

translates to

```
Range i_a(10, p_1, blk), j_a(10, p_2, blk) ;
Array<2, float> a(i_a, j_a) ;
Array<1, float> b(i_a) ;

for(i_a.first() ; i_a ; i_a++)
  for(j_a.first() ; j_a ; j_a++)
    a(i_a, j_a) = b(i_a) + fun(j_a()) ;
```

The Mapping Problem

A central problem in HPF translation is mapping of intermediate results, ie, subexpressions. Especially difficult in FORALL.

Are developing schemes applicable to general FORALL statements, with no reliance on "Owner-Computes" type rules. Mapping scheme specified in Prolog.

Example:

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

FORALL (I = 1 : N, J = 1 : N)
  a (I, J) = FUN1 (I) + (b (J) + FUN2 (J))
ENDFORALL
```

Mapping of Example

Possible Adhoc translation. . .

```
array float b [i_b] ;
array float a [i_a, j_a] ;

array float temp1 [i_b] ;
where(i_b)
  temp1 = b + fun2(i_b) ;  // temp1 [j] = b [j] + fun2(j)

array float temp2 [j_a] ;
remap(temp2, temp1) ;      // communication !

where(i_a) {
  float temp3 = fun1(i_a) ;
  where(i_b)
    a = temp3 + temp2 ;    // a [i, j] = fun1(i) + temp2 [j]
}
```

JISC work completed January-September 1994

Major tasks have been

- Investigation of "expression mapping" strategies.
- Requirements analysis for HPF run-time library.
- Definition of Adhoc.
- Definition of Mini-Adlib.

Implementation work on Adlib in progress. First MPI version expected by December 1994.

Translator: EPSRC project.

JISC project closely related to EPSRC project "Efficient translation of High Performance Fortran for distributed memory systems", currently supporting John Merlin.

Latter project emphasises *translation*, and will provide the HPF translator—JISC manpower mainly focussed on issues of run-time support (target "abstract machine").

First translator (available end of this year) will adopt a simplified mapping strategy. Next year, JISC work on expression mapping will be further developped and integrated into more advanced translator. Also, code generation will be improved.