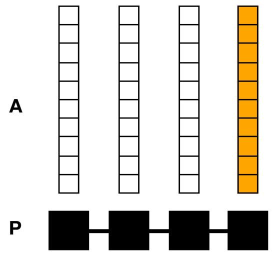
HPF Compilation Issues

- Application (simple parallelism)
- Language (mappings, size, static knowledge)
- Target (architecture, comm., progr. models)

Strange mappings (2)

```
      real A(n)
chpf$ processors P(4)
chpf$ template T(4*n)
chpf$ align A(i)
chpf$.    with T(4*i)
chpf$ distribute T(cyclic)
chpf$.    onto P
```



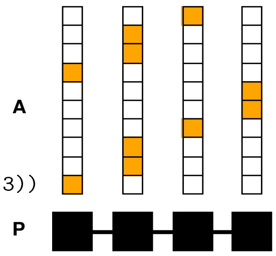
Compilation Course Motivation

the user must understand
the compiler technology
to write *efficient* HPF codes

- HPF *young* language
- user help required (directives, style)
- compilation is still a *research* area
- implementation dependent style and optimizations

Strange mappings (1)

```
      real A(n)
chpf$ processors P(4)
chpf$ template T(2*n)
chpf$ align A(i)
chpf$.    with T(2*i)
chpf$ distribute T(cyclic(3))
chpf$.    onto P
```



Outline

- Introduction
- Compilation Overview and Formalization
- Regular and Irregular Enumeration
- Allocation and Addressing
- Conclusion

HPF Language

- Fortran 77 *old features*
  - commons
  - sequence association
  - assumed size arguments
- Fortran 90
  - structured and modular
  - (too) large
- HPF additions
  - variety of mappings
    (`align`, `cyclic(n)`, replication)
  - dynamic management
    (`re...`, `inherit`, intrinsics)

State of the Art in Compiling HPF

PRS Spring School, March 1996

Fabien COELHO (CRI, ENSMP)
Cécile GERMAIN (LRI, U. Paris-Sud)
Jean-Louis PAZAT (IRISA)

Application

- available data-parallelism
- regularity and locality of accesses
- suitability to Fortran and HPF

Dynamic distributions

```
chpf$ distribute A(cyclic)
      if (A(1).lt.0.) then
chpf$   redistribute A(block)
      endif
! now the compiler
!   cannot know whether
! A is block or cyclic
      B(:) = A(:)
```

```
      subroutine foo(A)
      real A(10)
chpf$ inherit A
chpf$ distribute A * onto *
! the compiler cannot know
!   about A mapping.
      A(:) = ...
      ...
```


HPF Data Mapping

- `align distribute template processors`
- Ownership: $\mathcal{O}_{\mathtt{A}}(p) = \{a | \mathtt{A}(a) \text{ is mapped on } p\}$

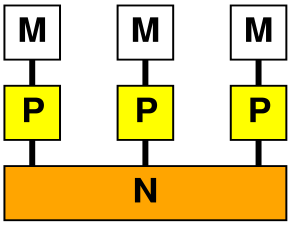
HPF Parallelism

- `forall independent sum prod max...`
- work sharing between processors:

  $\mathcal{C}(p) = \{i | i \in \mathcal{I} \text{ and computed on } p\}$

Target Architecture: MIMD-DM

independent processors and distributed memory



Target Communication Model

- Software, Operating System and Hardware Support
- Message-Passing versus Get, Put, Synchronize

```
! x is mapped on P1
! y is mapped on P2
x = f(y)
y = g(x)
```

- *Owner Computes Rule*

Get, Put, Synchronize

- direct access in the remote memory (address?)
- one-sided communications
- dependences $\Rightarrow$ synchronizations
- need hardware support: Cray T3D

```
! P1                        ! P2
t = get y from P2
x = f(t)
put x as s on P2
synchronize                 synchronize
                            y = f(s)
```


Message-Passing

- packing/unpacking of a message
- necessary hand-shaking
- standards: MPI, PVM
- workstation clusters: IBM SP2, DEC alpha farm

```
! P1                        ! P2
t = receive from P2         send y to P1
x = f(t)
send x to P2                s = receive from P1
                            y = g(s)
```


Hand-Shaking Implications

- aggreement about message
  - existence!
  - identification and content!
  - layout of data!
- different source and target storage and addressing
- *meta* data in the message?

SPMD Programming Model

Single Program Multiple Data

- generation of one (or few) program(s)
- parametrized by the processor identity
- simple, scalable, but less optimizations?

```
// 0 <= myself < n
if (myself>=1) {
    send data to myself-1;
}
if (myself<n-1) {
    receive data from myself+1;
}
```

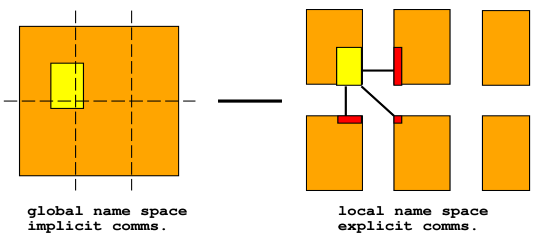
Compilation Constraints

- specified ownership of data and parallelism
- distributed-memory parallel architecture
- message-passing communication model
- SPMD programming model

Issues

- Allocation and Addressing of
  - local data
  - temporary copies of remote data
- Computation distribution
- Communications
- Sequential semantics: I/O

Compilation problem



Principle of HPF

- data-parallel programming language
  - global name space
  - implicit communications
- compiler target programming model
  - local name space . . .
  - explicit communications

Generic Example

independent forall$(i \in \mathcal{I})$
A$(f(i))$ = B$(g_1(i))$ + B$(g_2(i))$

- iteration space $\mathcal{I}$
- access functions f, g_1, g_2
- parallel loop: f must be injective on $\mathcal{I}$

Compilation Overview

- simple parallel loop
- work assignment to processors: *Owner Computes Rule*
- needed elements $\Rightarrow$ communications
- possible SPMD codes

Runtime Resolution Example

```
! Compilation of x = SQR(y)
Cp = Computer(x, y);
Ox = Owner(x);
Oy = Owner(y);

if (myself==Cp) { // Collect
  t1 = (myself==Oy) ? y : receive from Oy;

  t2 = SQR(t1); // Computation!

  if (myself==Ox) // Update
    x = t2;
  else
    send t2 to Ox;
} else {
  if (myself==Oy) // Collect
    send y to Cp;
  if (myself==Ox) // Update
    x = receive from Cp;
}
```


Runtime Resolution of Accesses

- basic default compilation scheme
- managed at runtime by all the processors
- statement per statement basis
  - chose the *computer*
  - compute *owners*
  - elementary communications
- very inefficient

Communications

- message per message or all at once.

  delayed sending but simpler enumeration

- optimizations to reduce the number of messages

  (high latency assumed)

  - *vectorization*: fusion of small messages
  - *coalescing*: fusion of elements ...
  - *aggregation*: several logical messages in one

SPMD codes

send $\mathcal{S}(p,*)$
receive $\mathcal{S}(*,p)$
for $i \in \mathcal{C}(p)$ ...

Simple SPMD code

send $\mathcal{S}(p,*)$
for $i \in \mathcal{C}_{\text{local}}(p)$ ...
receive $\mathcal{S}(*,p)$
for $i \in \mathcal{C}_{\text{remote}}(p)$ ...

Overlap SPMD code

Viewed array elements:

$$\mathcal{V}_{\text{B}}(p) = \{a | \exists i \in \mathcal{C}(p), a = g_1(i) \vee a = g_2(i)\}$$

Communications:

$$\mathcal{S}_{\text{B}}(p,p') = \mathcal{O}_{\text{B}}(p) \cap \mathcal{V}_{\text{B}}(p')$$

- array element a from p to p'

Owner Computes Rule

The owner of the assigned reference (lhs)
performs the computations (rhs)

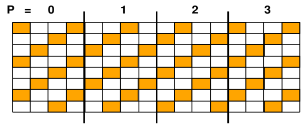
$$\mathcal{C}(p) = \{i \in \mathcal{I} | f(i) \in \mathcal{O}_{\text{A}}(p)\}$$

- simpler code: no remote update

Formalization example

from Chatterjee *et al.*

```
      real A(0:42)
chpf$ template T(0:127)
chpf$ processors P(0:3)
chpf$ align A(i) with T(3*i)
chpf$ distribute T(cyclic(4))
chpf$.     onto P
      A(0:42:3) = ...
```



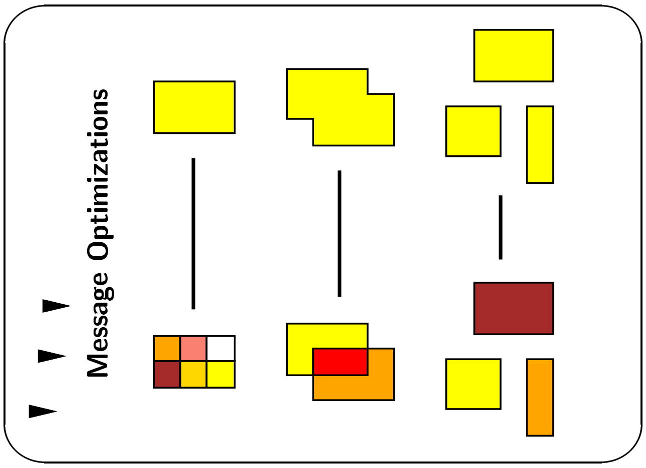
Affine Formalization

- affine parametric equations from:
  - HPF mapping directives
  - loop nest
  - access functions
- $\Rightarrow$ well-known mathematical framework

  linear algebra, polyhedron theory, integer lattices ...

Issues

- high level sets
- efficient enumeration needed
- practical management: many questions
  - data structures, operators?
  - program analysis, parametric?
  - algorithms, approximations?
- constraints on $\mathcal{I}$, f, g_1, g_2?



Affine equations

`A(0:42)`	$0 \leq \alpha \leq 42$
`T(0:127)`	$0 \leq \theta \leq 127$
`P(0:3)`	$0 \leq \psi \leq 3$
`align A(i) with T(3*i)`	$\theta = 3\alpha$
`cyclic(4)`	$0 \leq \delta \leq 3$
`distribute T(cyclic(4)) onto P`	$\theta = 16\gamma + 4\psi + \delta$
`0:42:3`	$0 \leq i \leq 14$
`A(0:42:3)`	$\alpha = 3i$

Regular enumeration

- description of sets ($\mathcal{C}$, $\mathcal{S}$) and associated enumeration
- different assumptions:
  - alignment stride
  - block, cyclic, general cyclic
  - array section, full loop nest
  - local elements ($\mathcal{C}$), communications
  - simple (shift) or general communications

3 Different Approaches

- Closed Forms
- Finite State Machines
- Polyhedron theory

Closed Forms

- formula to be instanciated
- array section focus: lower, upper, stride triplet

  $(1:22:3) =$
- array sections are closed under
  - affine transformations:
    $a(l:u:s)+b = (al+b:au+b:as)$
  - intersections:
    $(l_1:u_1:s_1) \cap (l_2:u_2:s_2) = (l_3:u_3:s_3)$

Array Sections of Block/Cyclic Distributions

`A(`$\mathcal{D}_\mathtt{A}$`) = B(`$\mathcal{D}_\mathtt{B}$`)`

- access: $\mathcal{D}_\mathtt{A}$ and $\mathcal{D}_\mathtt{B}$ are sections; $\mathcal{D}_\mathtt{B} = a\mathcal{D}_\mathtt{A} + b$
- block/cyclic distribution: $\mathcal{O}(p)$ is a section
- *Owner Computes Rule*: $\mathcal{C}(p) = \mathcal{O}_\mathtt{A}(p) \cap \mathcal{D}_\mathtt{A}$
- viewed elements: $\mathcal{V}_\mathtt{B}(p) = a\mathcal{C}(p) + b$
- communications: $\mathcal{S}_\mathtt{B}(p, a, p') = \mathcal{V}_\mathtt{B}(p') \cap \mathcal{O}_\mathtt{B}(p)$

Extensions

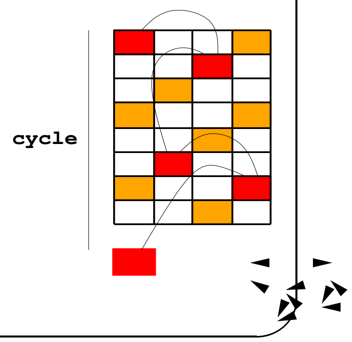
- general cyclic distributions: `cyclic(n)`
  - block distribution on many *virtual* processors
  - cyclic assignement to the HPF processors
  - dual view also possible (cyclic/block)
- alignment strides:
  - resolution at the template level
  - final packing of the section (÷)

Closed Forms Summary

- simple and parametric
- limited to array section accesses
- runtime instanciation of the sections
- many sections for `cyclic(n)`

Finite State Machines

- repeated patterns
- runtime extraction
- automaton
- local addresses



Generated code

```
// Delta holds the pattern
// computed at runtime
Delta[4] = { 2, 5, 2, 3 } ;

// index in the pattern
i = 0 ;
a = initial ;

// loop to scan the pattern
while (a <= final) {
  // computation with address a
  ... ;

  // next location
  a += Delta[i] ;
  i = (i+1) % 4 ;
}
```


Pattern Extraction Algorithm

- local elements of `cyclic(n)` and section access
- initial:
  - for each column (δ)
  - find the first element in the section
  - sort $\Rightarrow$ pattern $\Rightarrow$ transition table
- improvements:
  - based on lattice theory
  - avoid the sort phase

Extensions

- alignment stride: apply the algorithm twice
- affine subscript: combine several transition tables
- simple communication sets: shift
- multi-dimensions: per dimension ...

Discussion

- fast, on local addresses
- runtime management of tables
- simple sets (local elements)

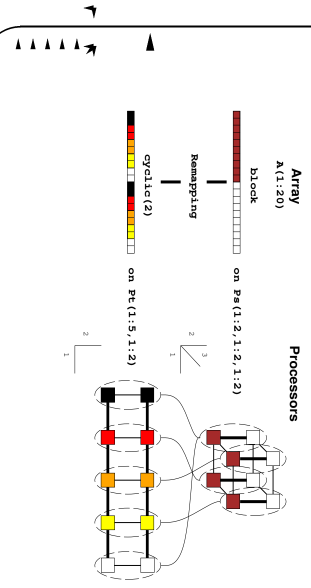
Polyhedrons: General Assumptions

- affine parametric loop nest ($\mathcal{I}$)

```
do i=n, n+m, 2
  do j=i+1, m+7
    A(2*i+j-m,n-i) = ...
```

- affine access functions (f, g_1, g_2)
- full HPF mappings (`align`, `distribute`)

General Communication Example



HPF Modelization

- declarations

  `A(20)` $1 \leq \alpha \leq 20$

  `Ps(2,2,2)` $1 \leq \psi_1 \leq 2 \ldots$

  `Pt(5,1)` $1 \leq \psi'_1 \leq 5 \ldots$

- HPF directives

  `block(10)` $0 \leq \delta \leq 9$

  `distribute()` $\alpha = 10\psi_2 + \delta - 9$

  `cyclic(2)` $0 \leq \delta' \leq 1$

  `distribute()` $\alpha = 10\gamma' + 2\psi'_1 + \delta' - 2$

- local declarations and addressing

  `As(10)` $1 \leq \alpha_s \leq 10,\ \alpha_s = \delta + 1$

  `At(4)` $1 \leq \alpha_t \leq 4,\ \alpha_t = 2\gamma' + \delta' + 1$

- work sharing: $\psi'_1 = 4\lambda + 2\psi_3 + \psi_1 - 2$

Interpretation

- solution to the system *exactly* links
  - array element $\alpha, \alpha_s, \alpha_t$
  - source processor(s) ψ
  - target processor(s) ψ'
- Dual enumeration code (send/receive)
  - guards
  - remote processors enumeration
  - extraction and storage
  - special case: data locally available


```
PSI1 = MYPOS(1,2)
PSI2 = MYPOS(2,2)
PSI3 = MYPOS(3,2)
DO TMP3 = 0, (7-PSI1-2*PSI3)/4
  PSIp1 = -2+PSI1+2*PSI3+4*TMP3
  BUFINDEX = 0
  DO LALPHAp1 = -9+10*PSI2, 10*PSI2
    LALPHA1 = 45+5*LALPHAp1-50*PSI2+PSIp1
    BUFINDEX = BUFINDEX+1
    INTEGER4_BUFF(BUFINDEX) = A_1(LALPHA1)
  ENDDO
  CALL BROADCAST(...)
ENDDO
PSIp1 = MYPOS(1,1)
PSIp2 = MYPOS(2,1)
DO PSI1 = 1, 2
  DO PSI2 = 1, 2
    DO PSI3 = 1, 2
      DO TMP3 = (5-PSI1-2*PSI3+PSIp1)/4, 2-PSI1-2*PSI3+PSIp1/4
        IF (...) THEN
          CALL INTEGER4_HPFC_BUFUPK(...)
          BUFINDEX = 0
          DO LALPHAp1 = -9+10*PSI2, 10*PSI2
            LALPHA1 = 45+5*LALPHAp1-50*PSI2+PSIp1
            BUFINDEX = BUFINDEX+1
            A_2(LALPHAp1) = INTEGER4_BUFF(BUFINDEX)
          ENDDO
        ELSE
          DO LALPHAp1 = -9+10*PSI2, 10*PSI2
            LALPHA1 = 45+5*LALPHAp1-50*PSI2+PSIp1
            A_2(LALPHAp1) = A_1(LALPHA1)
          ENDDO
        ENDIF
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

Irregular Enumeration

- irregular applications (sparsity, indirections ...)
  many (most)? real applications
- runtime management of sets
  - **distributor**: computation sets ($\mathcal{C}$)
  - **inspector**: reusable communication schedules ($\mathcal{S}$)
  - **executor**: actual use

Mixing Temporary and Local Data

- local references in temporary also: copies
  `forall` $(i \in \mathcal{C}(p))$ `A(`$f'(i)$`) = T1(`$t_1(i)$`) + T2(`$t_2(i)$`)`
- temporary along with locals: management?
  `forall` $(i \in \mathcal{C}(p))$ `A(`$f'(i)$`) = B(`$g_1'(i)$`) + B(`$g_2'(i)$`)`

Irregular y = A.x

```
c MSR (Modified Sparse Row) format
SUBROUTINE matvect (n,taille,a,ja,x,y)

integer          n, taille, ja(taille)
double precision  x(n), y(n), a(taille)

DO i = 1, n
  y(i) = a(i)*x(i) ! diagonal
ENDDO
DO i = 1,n
  DO k = ja(i), ja(i+1)-1
    y(i) = y(i) + a(k) *x(ja(k))
  ENDDO
ENDDO
```


Cost of Separate Temporary References

- individual guards: too expensive
  `forall` $(i \in \mathcal{C}(p))$
  `A(`$f'(i)$`) = (`$g_1(i) \in \mathcal{O}_{\mathtt{B}}(p)$ `? B(`$g_1'(i)$`) : T1(`$t_1(i)$`))`
  `+ (`$g_2(i) \in \mathcal{O}_{\mathtt{B}}(p)$ `? B(`$g_2'(i)$`) : T2(`$t_2(i)$`))`
- compute set splitting: exponential, not obvious
  `forall` $(i \in \mathcal{C}_{\mathtt{ll}}(p))$ `A(`$f'(i)$`) = B(`$g_1'(i)$`) + B(`$g_2'(i)$`)`
  `forall` $(i \in \mathcal{C}_{\mathtt{rl}}(p))$ `A(`$f'(i)$`) = T1(`$t_1(i)$`) + B(`$g_2'(i)$`)`
  `forall` $(i \in \mathcal{C}_{\mathtt{lr}}(p))$ `A(`$f'(i)$`) = B(`$g_1'(i)$`) + T2(`$t_2(i)$`)`
  `forall` $(i \in \mathcal{C}_{\mathtt{rr}}(p))$ `A(`$f'(i)$`) = T1(`$t_1(i)$`) + T2(`$t_2(i)$`)`

Polyhedron Approach

- compile-time handling
- general assumptions about the input program
- parametric integer sets
- well-known mathematical formulation
- enumeration code efficiency?
- exponential worstcase behavior?

Allocation and Addressing

- trade-off:
  - reduce the amount of memory
  - allow fast local addresses
- data to be managed:
  - local part distributed arrays
  - temporary space for remote data

Sets and code generation

- sets: compile-time parametric integer polyhedra
- integers solutions = communications or computations
- enumeration code: polyhedron scanning algorithm
  - Fourier elimination
  - Parametric Simplex
- rely on these algorithms for efficiency ...

Results

- emphasize runtime libraries
- `PARTI` routines
- compiler seeks reuse of schedules
- average efficiency

Packed Form

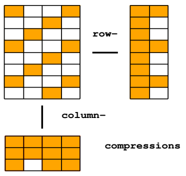
- full compression of holes
- memory optimal!
- no direct addressing formula
- multidimensions?
- suits only FSM

Regular Compression

- compatible with overlaps
- efficient incremental addresses
- possible locality of enumeration

Regular Compression

- dimension per dimension
- remove holes from cycles and aligns
- along rows or columns

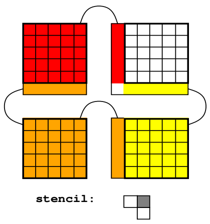


2-D view:
$\gamma = 3\alpha \bmod 4$; $\delta = 3\alpha \div 16$
row compression (r, c):
$r = \delta \div 3$; $c = \gamma$
column compression (r, c):
$r = \delta$; $c = \gamma \div 3$

Allocation of Overlaps

- extend allocation of borders
- suits stencil computations

```
      real A(10,10), B(10,10)
chpf$ processors P(2,2)
chpf$ align A with B
chpf$ distribute B(block,block) onto P
chpf$ independent...
      DO j, i...
        B(i,j) =
     .       A(i,j)+A(i-1,j)+A(i,j-1)
      END DO
```



Conclusion

- compilation of HPF: active *research* area
- all techniques benefit from static knowledge
- slow development of compilers
- all applications do not suit HPF
- port to HPF: change
  - algorithm
  - data structures
  - programming habits

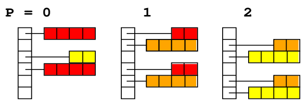
Allocation strategies

- design allocation and enumeration for cache
- subsequent optimizations for addressing

Discussion

- includes remote references
- uniform addressing
- runtime management
- high overhead for small cyclic

```
      B(0:23)
chpf$ align B with A
      B(2:23) = A(0:21)+A(2:23)
```



Software pagination

- paging system for distributed arrays
- address $\Rightarrow$ page number and offset
- allocation of non-empty pages

```
      real A(0:23)
chpf$ processors P(0:2)
chpf$ distribute A(cyclic(4)) onto P
```