

Modelos de Programación Paralela

- Modelos de Programación para Multiprocesadores
- El Modelo de Variables Compartidas
- Expresión del Paralelismo en el Modelo de Variables Compartidas
- Primitivas de Sincronización
- Ejemplo de Programación
- El Modelo de Paso de Mensajes
- Primitivas para Paso de Mensajes
- Ejemplo de Programación
- Paralelización Automática
- High Performance Fortran
- Librerías Numéricas Paralelas

Modelos de Programación para Multiprocesadores

Modelos de Programación Secuencial

- ✓ FORTRAN 77

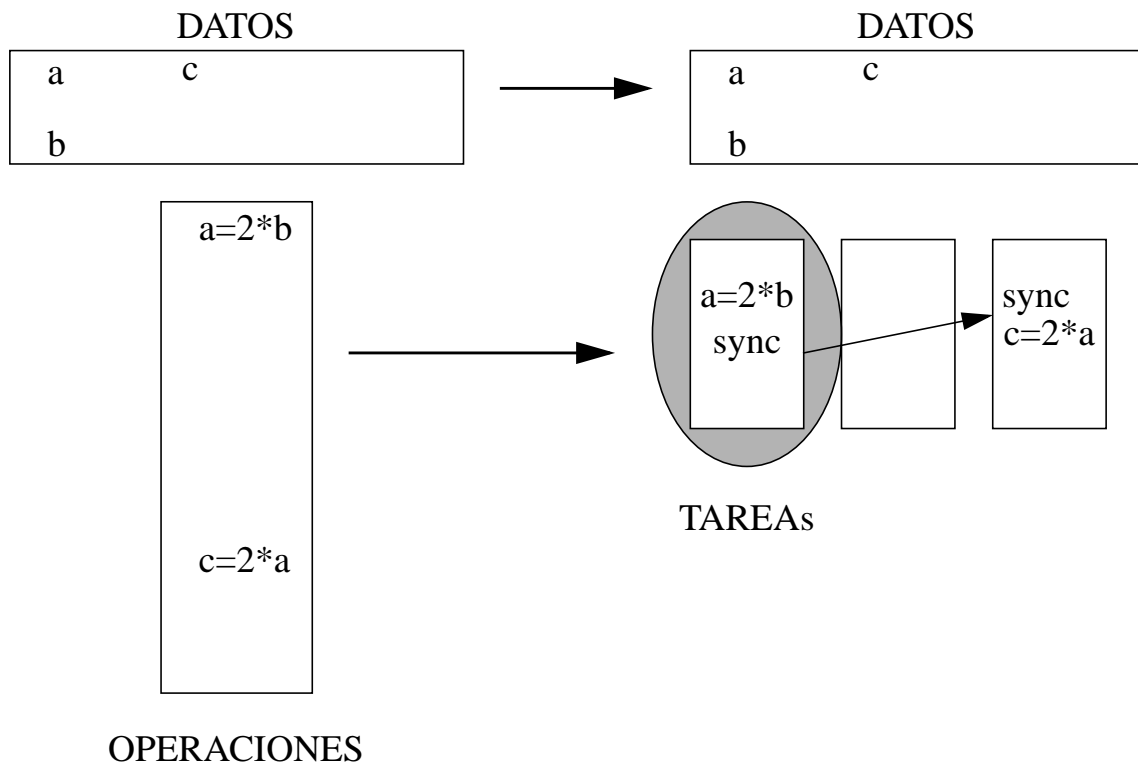
- ✓ FORTRAN 77 + Librerías Paralelas

Modelos de Programación Paralela

- ✓ Variables Compartidas

- ✓ Paso de Mensajes

El Modelo de Variables Compartidas



Las operaciones se descomponen en tareas.

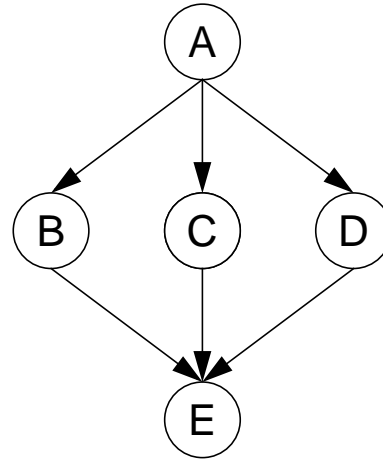
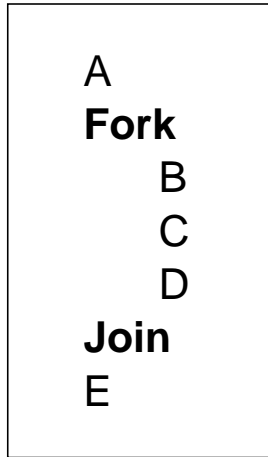
Los datos son compartidos por todas las tareas.

Se requieren primitivas de sincronización para:

- ✓ Señalización
- ✓ Acceso Exclusivo

Expresión del Paralelismo en el Modelo de Variables Compartidas

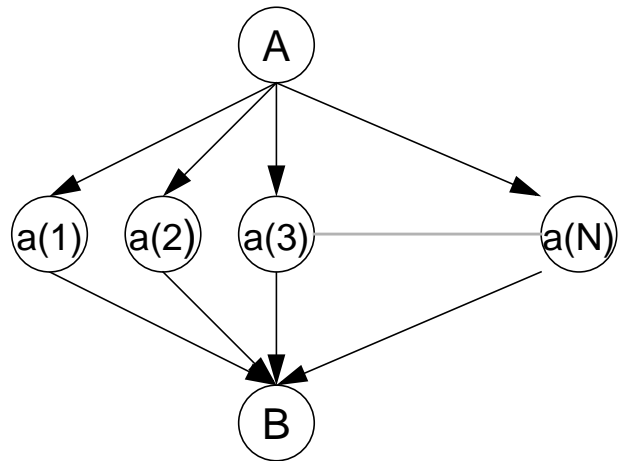
□ Fork & Join



Expresión del Paralelismo en el Modelo de Variables Compartidas

□ DOALL

```
A  
Doall i=1, N  
  a(i) := b(i)*c(i)  
enddo  
B
```



Necesidad de Sincronización

❑ Señalización

Tarea A

·
·
x := f(a)
·
·
·

Tarea B

·
·
·
y := f(x)
·
·

❑ Acceso exclusivo

variable compartida {cont = 0}

Tarea A

·
·
cont := cont + 1
·
·
·

Tarea B

·
·
cont := cont + 1
·
·
·

Primitivas de Sincronización

Sincronización mediante Loads y Stores convencionales

□ Señalización

variable compartida {s= 0}

Tarea A

```
.  
. .  
x := f(a)  
s := 1  
.
```

Tarea B

```
.  
. .  
while s = 0 do  
enddo  
y := f(x)
```

□ Acceso exclusivo

variables compartidas {cont = 0, turno = 0, procA = fuera,
procB = fuera}

Tarea A

```
procA:= dentro  
if procB = dentro then  
  if turno = 1 then  
    procA := fuera  
    repeat until turno = 0  
    procA := dentro  
  endif  
  repeat until procB = fuera  
endif  


cont := cont + 1

  
turno := 1  
procA:= fuera
```

Tarea B

```
procB:= dentro  
if procA = dentro then  
  if turno =0 then  
    procB := fuera  
    repeat until turno = 1  
    procB := dentro  
  endif  
  repeat until procA = fuera  
endif  


cont := cont + 1

  
turno := 0  
procB:= fuera
```

Primitivas de Sincronización

Primitivas de Bajo Nivel (con necesidad de soporte hardware)

Usualmente implementadas en forma de instrucción de lenguaje máquina.

❑ Test & Set

Semántica

Test&Set (lock)

```
tmp := lock
```

```
lock := 1
```

```
return tmp
```

Acceso Atómico

Uso para Acceso Exclusivo

```
{lock = 0}
```

```
.
```

```
While (Test&Set (lock)≠0)
```

```
endwhile
```

```
{acceso exclusivo}
```

```
lock := 0
```



CEPBA



D A C

Primitivas de Sincronización

Primitivas de Alto Nivel

Implementadas en base a primitivas de bajo nivel más una cierta cantidad de software.

□ Semáforos

Semántica

Wait (s)

```
if s = 0 then
  bloquea la tarea en
  una cola
else   s = 0
```

Signal (s)

```
if cola no vacía then
  desbloquear el primero
  de la cola
else   s = 1
```

Uso para Acceso Exclusivo

```
{s = 1}
.
Wait (s)
{acceso exclusivo}
Signal (s)
```

Primitivas de Sincronización

Primitivas de Alto Nivel

□ Barreras

Semántica

Barrier (b)

$b := b+1$

if $b < \text{Numproc}$ **then**

bloquear tarea en la cola

else

desbloquear todas las tareas de la cola

$b := 0$

Uso

{ $b = 0$ }

While cond **do**

leer información compartida

cálculo

actualización de la información compartida

barrier (b)

end

Primitivas de Sincronización

Primitivas de Alto Nivel

□ Monitores

Ejemplo

```
Producer-Consumer: monitor
begin
  buffer está compartido
  is_full es un boolean
  empty and full son semáforos

  procedure produce (data)
  begin
    if (is_full) then
      wait (empty)
    endif
    buffer := data
    is_full := true
    signal (full)
  end

  procedure consume (data)
  begin
    if (not is_full) then
      wait (full)
    endif
    data := buffer
    is_full := false
    signal (empty)
  end

  initialize
    is_full := false
end
```

Ejemplo de Programación

Calcular un histogramas de notas

Algoritmo 1

```
integer notas (N) SHARED (* notas son enteros entre 0 y 10 *)
integer histo (0:10) SHARED
semaphore s (0:10)
integer i

doall i=1,N
  wait (s(notas(i)))
  histo (notas(i)) = histo (notas (i)) + 1
  signal (s(notas (i)))
enddo
```

Ejemplo de Programación

Calcular un histogramas de notas

Algoritmo 2

Se pretende reducir el coste de sincronización

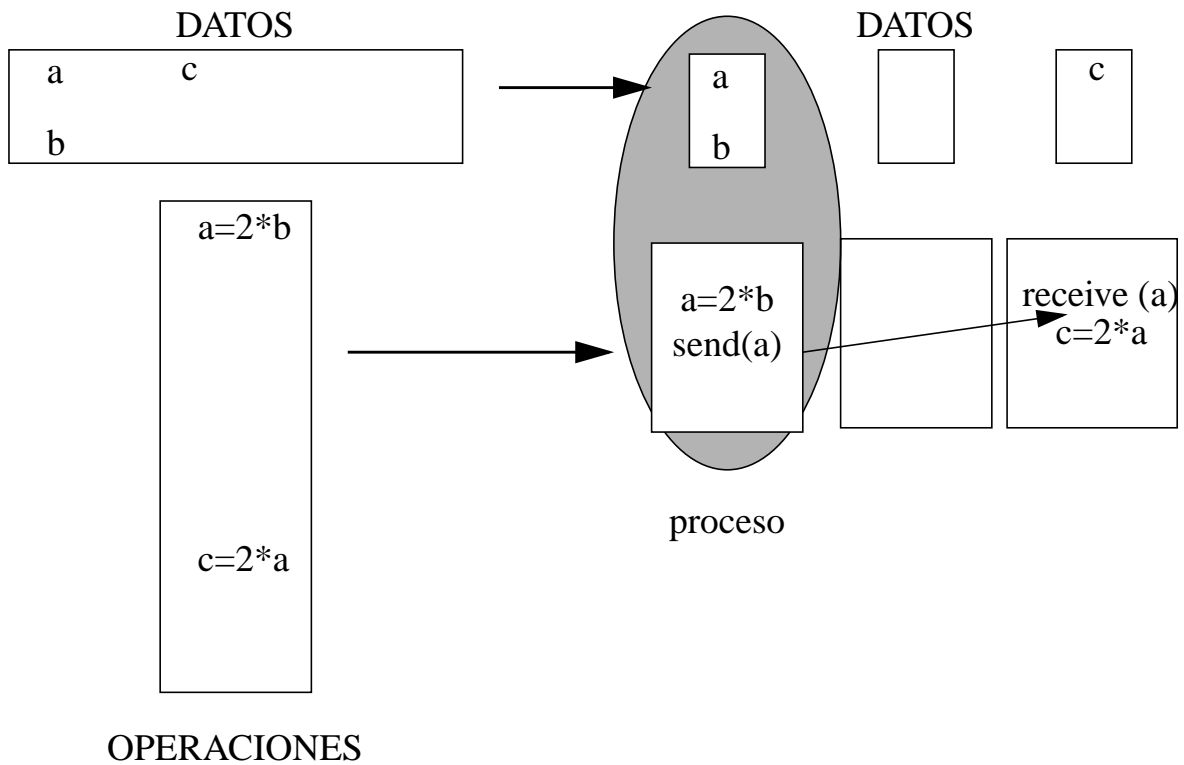
```
integer notas (N) SHARED (* notas son enteros entre 0 y 10 *)
integer histo (0:10) SHARED
semaphore s
integer histo_local (0:10)
integer i, p, Numproc

doall p=0, Numproc-1
  do i=p*N/Numproc + 1, (p+1)*N/Numproc
    histo_local (notas(i))= histo_local (notas(i)) + 1
  enddo

  wait (s)
  do i=0, 10
    histo (i) = histo (i) + histo_local (i)
  enddo
  signal (s)

enddo
```

El Modelo de Paso de Mensajes

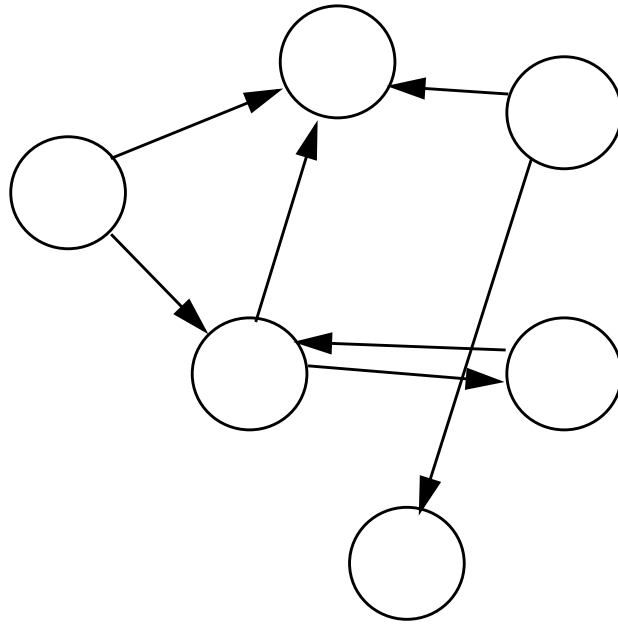


Las operaciones Y LOS DATOS se descomponen en procesos.

Los procesos sólo tienen acceso directo a los datos privados (locales).

Los datos no locales se acceden mediante intercambio de mensajes entre los procesos.

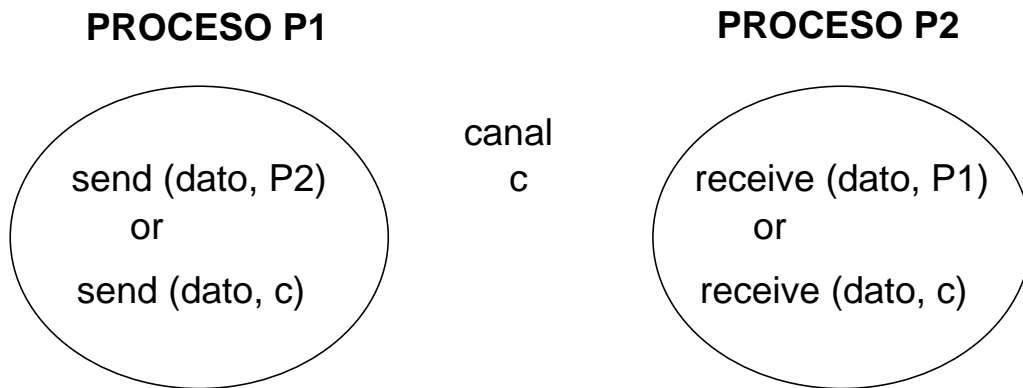
Procesos y Canales



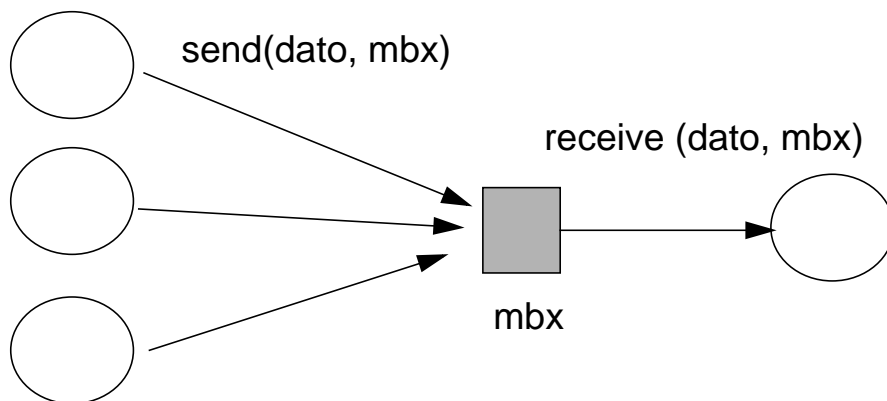
En el modelo de paso de mensajes, un programa paralelo se ve como un conjunto de procesos que se intercambian información a través de canales.

Primitivas para Paso de Mensajes

Canales Punto a Punto



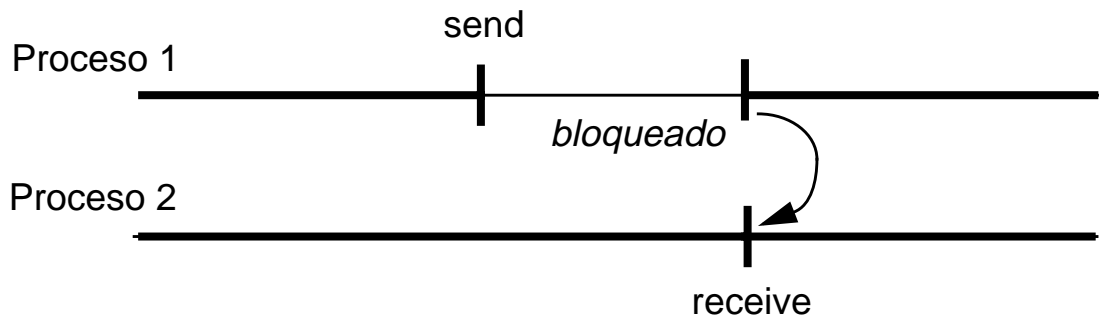
Mailboxes



Primitivas para Paso de Mensajes

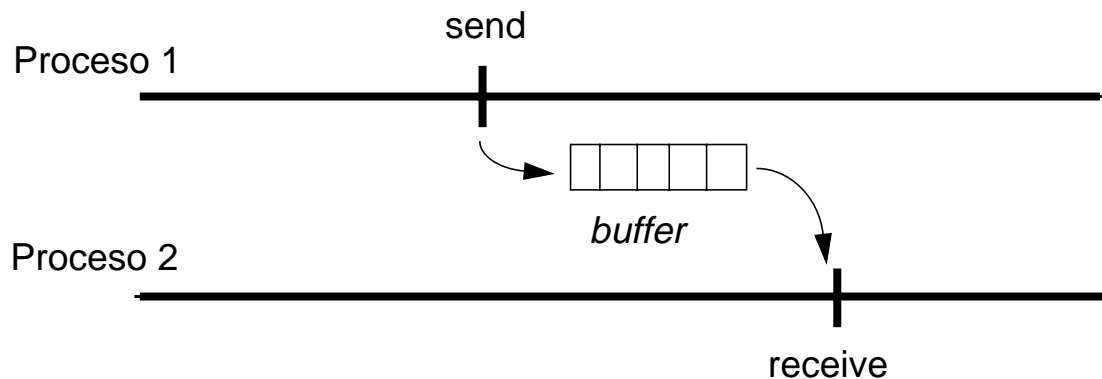
Sincronización en la comunicación

❑ Comunicación Síncrona



Comunicación más rápida porque no necesita buffering temporal.

❑ Comunicación Asíncrona



Comunicación más lenta.
Permite solapamiento cálculo/comunicación.

Ejemplo de Programación

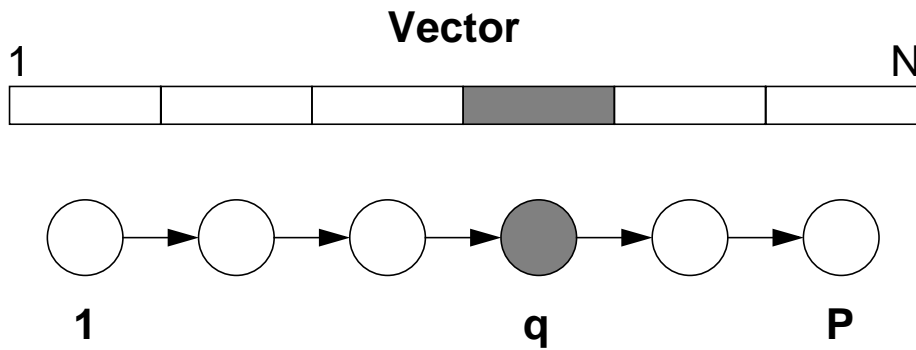
Calcular el máximo de un vector de reales.

Algoritmo 1

Usa una línea de P procesos.

Inicialmente, el proceso q almacena los elementos del $(q-1)*N/P + 1$ al $q*N/P$.

Finalmente, el máximo es escrito por el proceso P.



Ejemplo de Programación

Algoritmo 1

Código para el proceso q

```
real local_data (N/P), maxloc, maxloc2
```

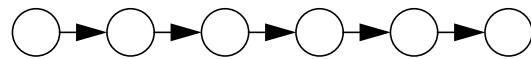
```
maxloc = -∞
```

```
do j=1, N/P
```

```
  if local_data [j] > maxloc
```

```
    then maxloc = local_data [j]
```

```
  endif
```



```
enddo
```

```
if q=1 then send (maxloc, q+1)
```

```
else if q < P
```

```
  then receive (maxloc2, q-1)
```

```
    if maxloc2 > maxloc
```

```
      then maxloc = maxloc2
```

```
    endif
```

```
    send (maxloc, q+1)
```

```
  else receive (maxloc2, q-1)
```

```
    if maxloc2 > maxloc
```

```
      then maxloc = maxloc2
```

```
    endif
```

```
    write (maxloc)
```

```
  endif
```

```
endif
```



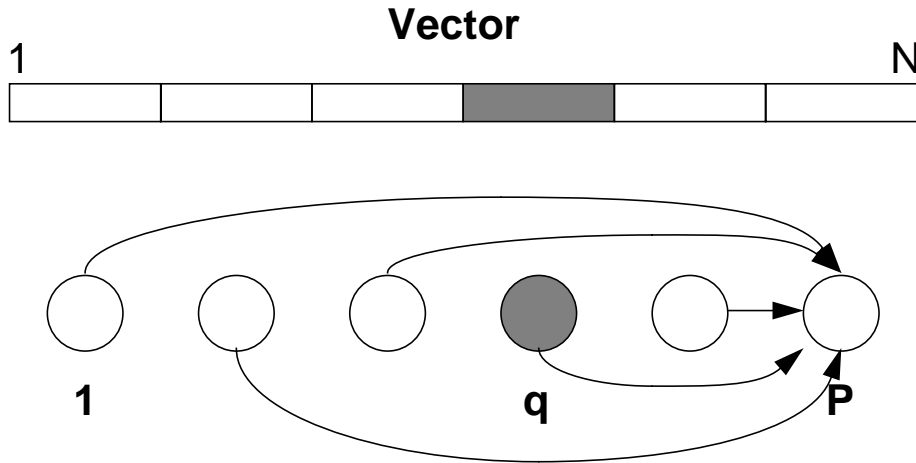
CEPBA



D A C

Ejemplo de Programación

Algoritmo 2



El máximo es escrito por el proceso P.

Ejemplo de Programación

Algoritmo 2

Código para el proceso q

```
real local_data (N/P), maxloc, maxloc2
```

```
maxloc = -∞
```

```
do j=1, N/P
```

```
  if local_data [j] > maxloc
```

```
    then maxloc = local_data [j]
```

```
  endif
```

```
enddo
```

```
if q ≠ P then send (maxloc, P)
```

```
else do i=1, P-1
```

```
  receive (maxloc2, i)
```

```
  if maxloc2 > maxloc
```

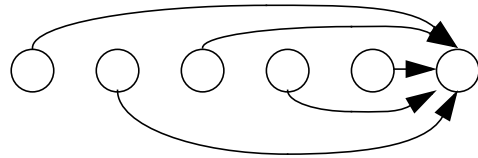
```
    then maxloc = maxloc2
```

```
  endif
```

```
enddo
```

```
write (maxloc)
```

```
endif
```



El código es más sencillo pero probablemente la topología de comunicación no coincide con la topología de interconexión (sobrecarga debida al encaminamiento de las comunicaciones no locales).

¿Es más fácil programar en el modelo de Variables Compartidas que en el de Paso de Mensajes?

Una Versión para el Modelo de Variables Compartidas

real data (N) **SHARED**

integer i,N,B

doall i=1,N,B

do j=i, i+B-1

if data [i] < data [j]

then data [i] = data [j]

endif

enddo

enddo

do i=1,N,B

if data [1] < data [i]

then data [1] = data [i]

endif

enddo

write (data [1])



CEPBA



D A C

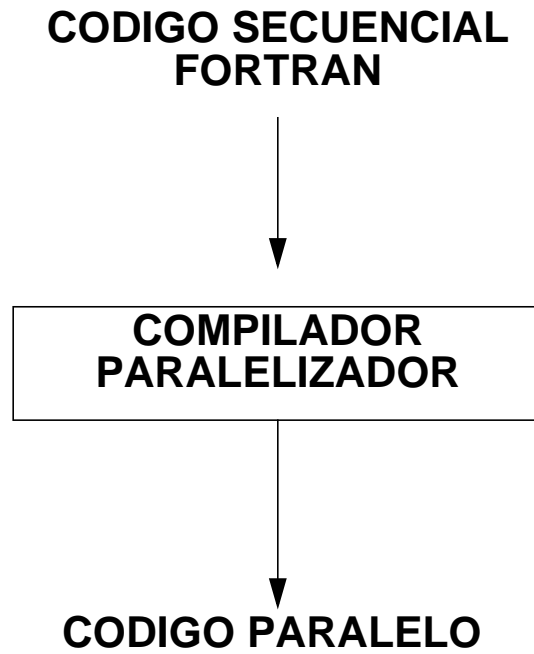
Relación Organización-Modelo de Programación

MODELO DE PROGRAMACIÓN

		Variables Compartidas	Paso de Mensajes
ORGANIZACIÓN	Memoria Compartida	Combinación natural Poco escalable Fácil de programar	Poco escalable Programación difícil
	Memoria Distribuida	Programación fácil Escalable Implementación difícil	Combinación natural Programación difícil Escalable

Paralelización Automática

Objetivos

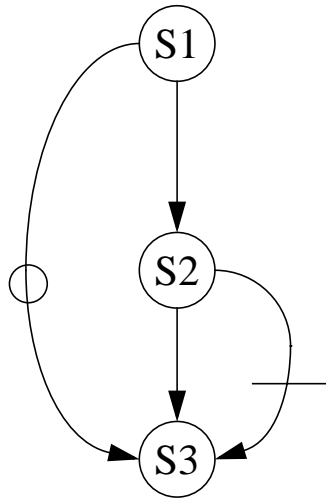


Paralelización Automática

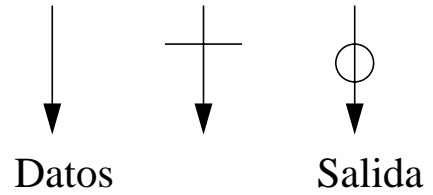
Paralelización para un modelo de Variables Compartidas. Técnicas Básicas

Identificación de dependencias entre sentencias

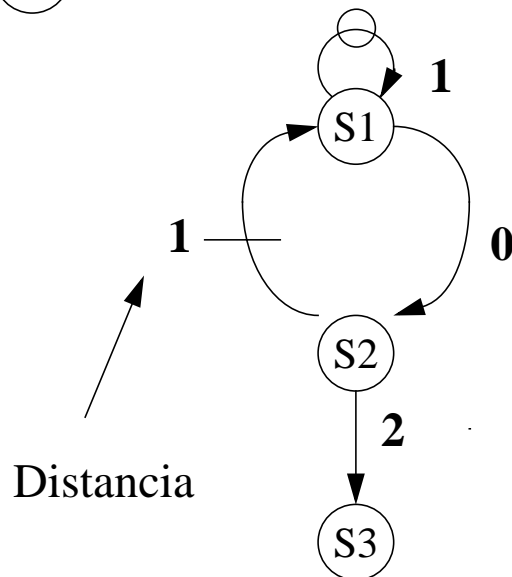
S1: $A = B + C$
S2: $D = A + E$
S3: $A = F + D$



Tipos de Dependencias



DO $i=1,N$
S1: $X = B(i)$
S2: $C(i) = X + 1$
S3: $A(i) = 2 * C(i-2)$
ENDDO



Paralelización Automática

Técnicas Básicas

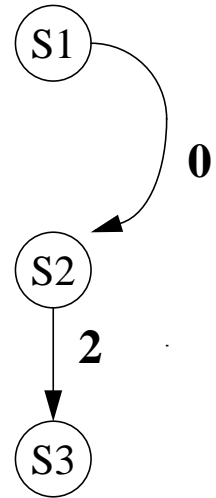
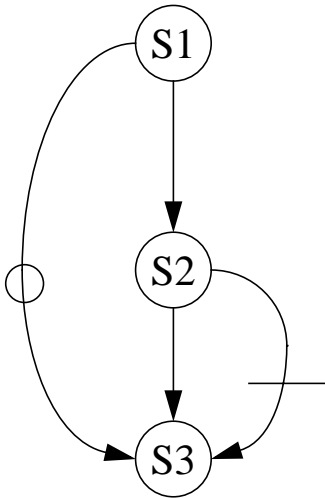
Eliminación de antidependencias y dependencias de salida

S1: $A = B + C$
S2: $D = A + E$
S3: $A = F + D$



S1: $A = B + C$
S2: $D = A + E$
S3: $AA = F + D$

DO $i=1,N$
S1: $\mathbf{X}(i) = B(i)$
S2: $C(i) = \mathbf{X}(i) + 1$
S3: $A(i) = 2 * C(i-2)$
ENDDO



Paralelización Automática

Técnicas Básicas

Distribución de Bucles

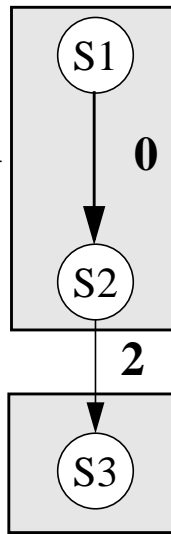
```
DO i=1,N  
S1: X(i) = B(i)  
S2: C(i) = X (i) +1  
S3: A(i) = 2*C(i-2)  
ENDDO
```

```
DO i=1,N  
S1: X(i) = B(i)  
S2: C(i) = X (i) +1  
ENDDO
```

```
DO i=1, N  
S3: A(i) = 2*C(i-2)  
ENDDO
```

```
DOALL i=1,N  
S1: X(i) = B(i)  
S2: C(i) = X (i) +1  
ENDDO
```

```
DOALL i=1, N  
S3: A(i) = 2*C(i-2)  
ENDDO
```



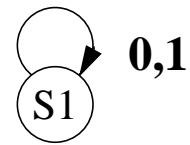
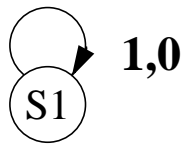
Paralelización Automática

Técnicas Básicas

Intercambio de Bucles

```
DO i=1,N  
  DO j=1,M  
    E(i,j) = E(i-1,j)*B(i,j)  
  ENDDO  
ENDDO
```

```
DOALL j=1,M  
  DO i=1,N  
    E(i,j) = E(i-1,j)*B(i,j)  
  ENDDO  
ENDDO
```



Paralelización Automática

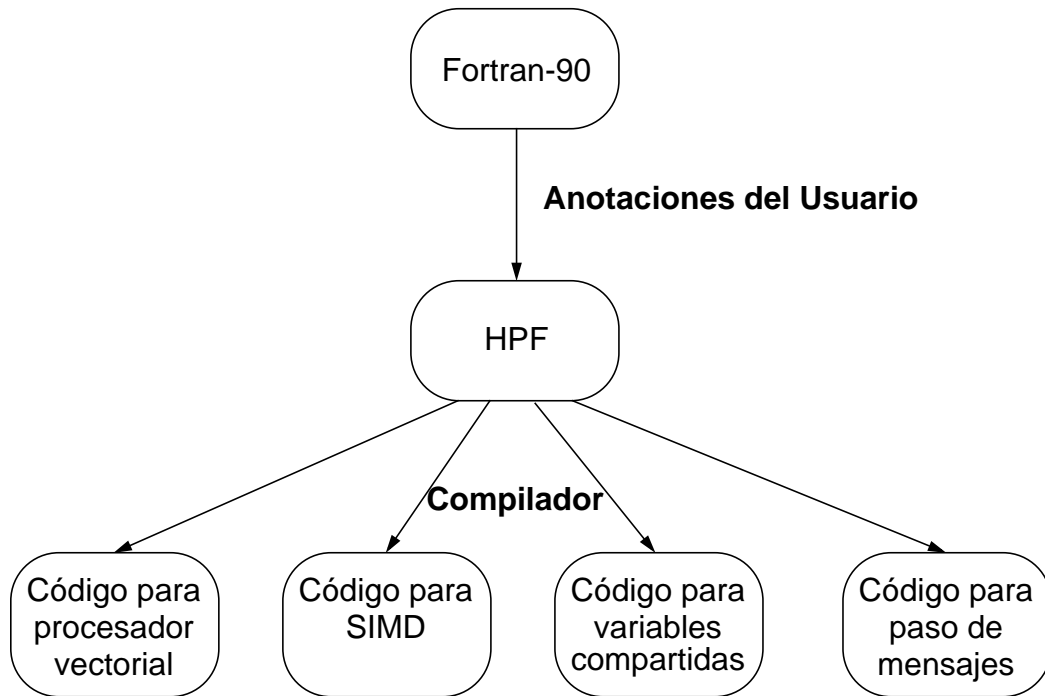
Paralelización para un modelo de Paso de Mensajes

El problema es mucho más complejo porque no basta con identificar sentencias que puedan ejecutarse en paralelo. También hay que tomar decisiones sobre la ubicación de datos.

De momento no hay compiladores suficientemente satisfactorios.

Actualmente se está trabajando en la línea de pedir ayuda al programador. El lenguaje ofrece unas directivas que permiten al programador especificar operaciones independientes y recomendar distribuciones de datos adecuadas para la aplicación. En esta dirección se han propuesto lenguajes tales como Fortran D, Viena Fortran o High Performance Fortran (HPF).

High Performance Fortran (HPF)



High Performance Fortran (HPF)

□ Declaración de Arrays

1) REAL A(10) ;A(1), A(2), A(3), ..., A(10)

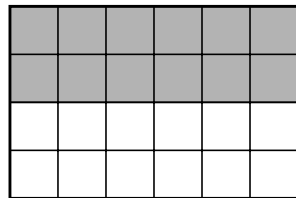
A(primero : último : salto)

A(1 : 5 : 2) → A(1), A(3), A(5)

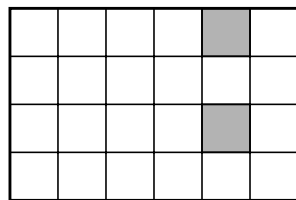
A(10 : 1 : -3) → A(10), A(7), A(4), A(1)

2) REAL M(4, 6)

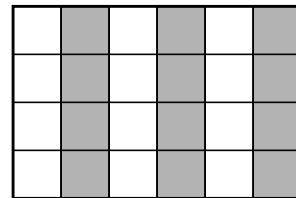
M(1 : 2 , :)



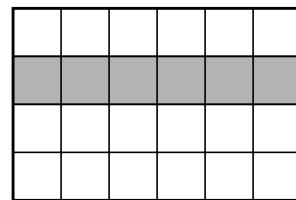
M(1 : 4 : 2 , 5)



M(:, 2 : 6 : 2)



M(2 , :)



High Performance Fortran (HPF)

❑ Operaciones con Arrays

REAL A(5), B(5), V(10)

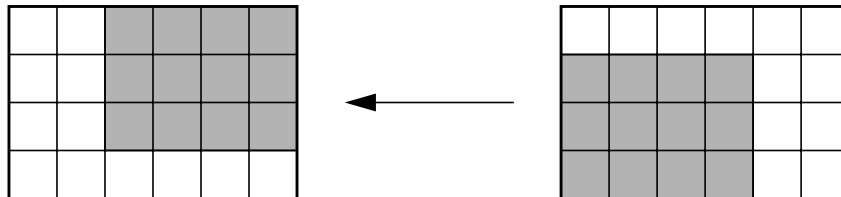
V(1 : 5) = A ** 2 + B ** 2

B = A (5 : 1 : -1)

WHERE (A(1:3) .NE. 0) V(1 : 3) = B(1 : 3) / A(1 : 3)

V(1 : 9) = V (2 : 10)

M (1 : 3, 3 : 6) = M (2 : 4 , 1 : 4)



High Performance Fortran (HPF)

□ Funciones Intrínsecas

- ✓ DOT_PRODUCT (Vector1, Vector2)

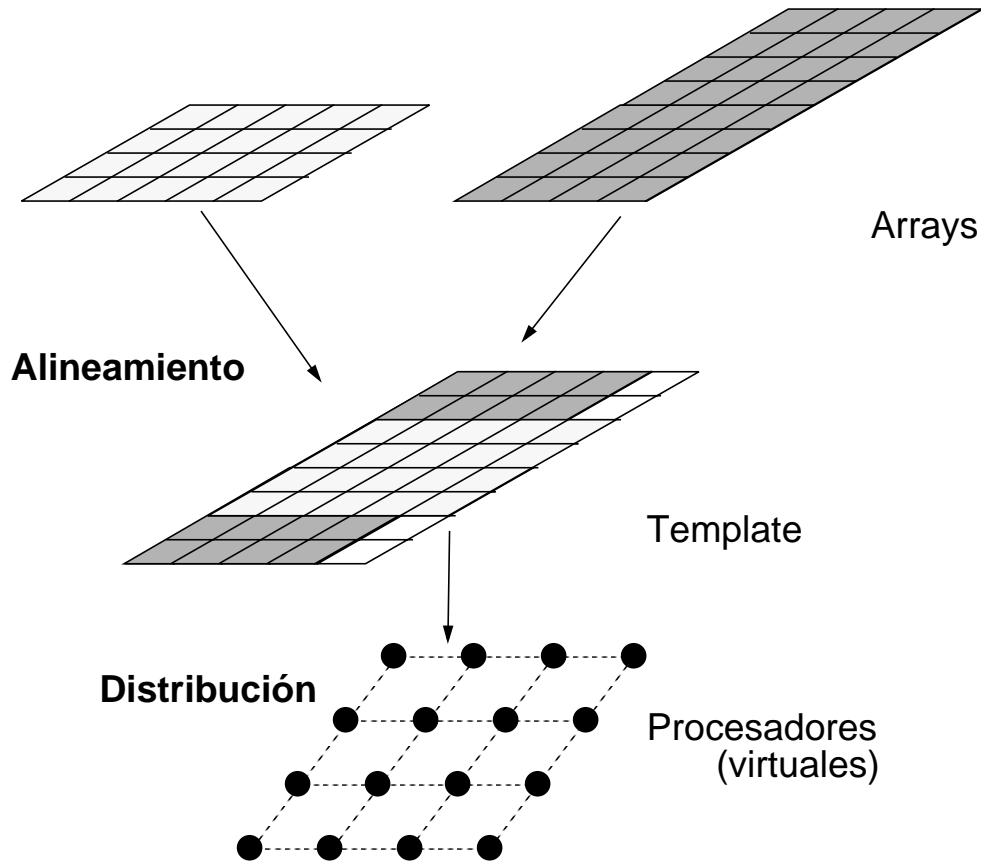
$$a = \text{DOT_PRODUCT} (B, C)$$

$$B = [1 \ 2 \ 3] \quad C = [2 \ 3 \ 4] \quad a = 20$$

- ✓ MATMUL (Matrix1, Matrix2)

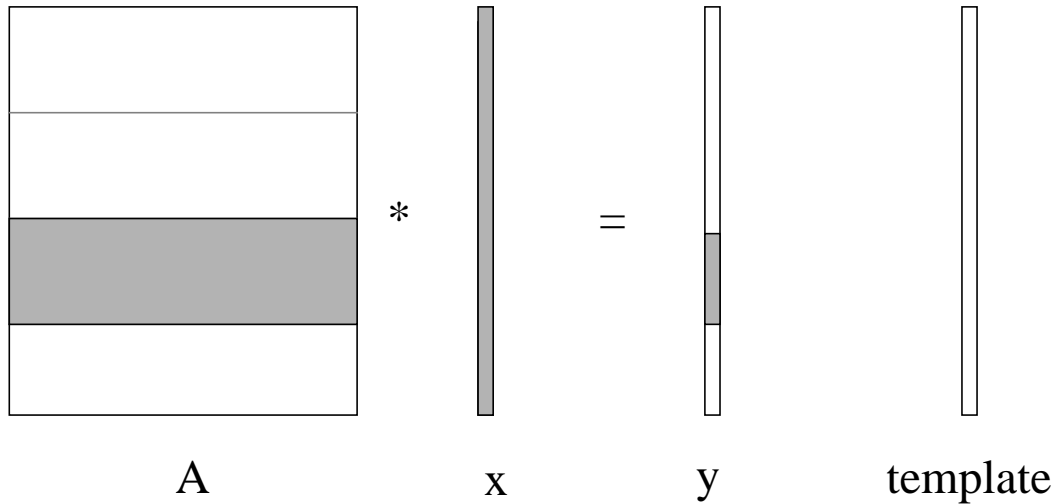
High Performance Fortran (HPF)

□ Directivas para la distribución de datos



High Performance Fortran (HPF)

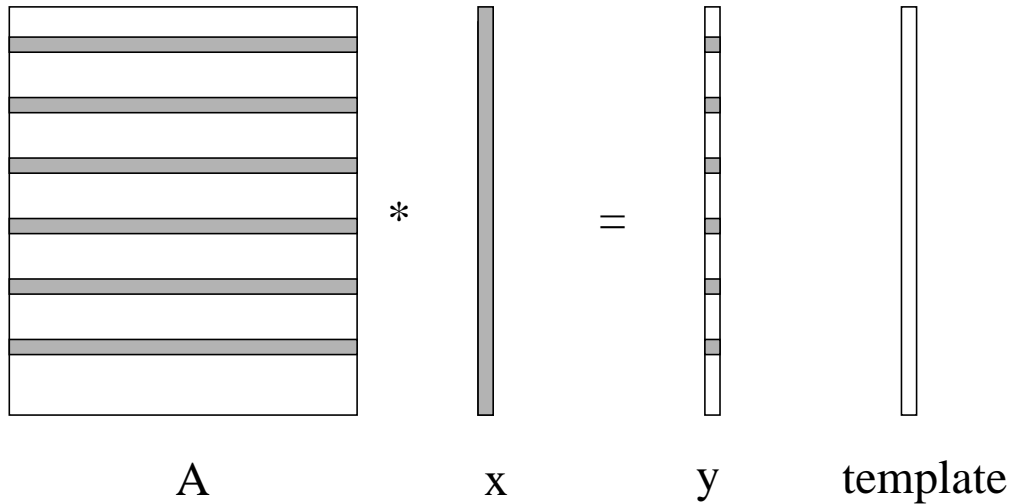
□ Directivas para la distribución de datos



A	x	y	template
Real*8	$A(N,N), x(N), y(N)$		
TEMPLATE	$t(N)$		
ALIGN	$A(i,j), t(i)$		
ALIGN	$y(i), t(i)$		
ALIGN	$x(*), t(i)$		
DISTRIBUTE (BLOCK)	$t(N)$		

High Performance Fortran (HPF)

□ Directivas para la distribución de datos



Real*8

TEMPLATE

ALIGN

ALIGN

ALIGN

DISTRIBUTE (CYCLIC)

$A(N,N)$, $x(N)$, $y(N)$

$t(N)$

$A(i,j)$, $t(i)$

$y(i)$, $t(i)$

$x(*)$, $t(i)$

$t(N)$



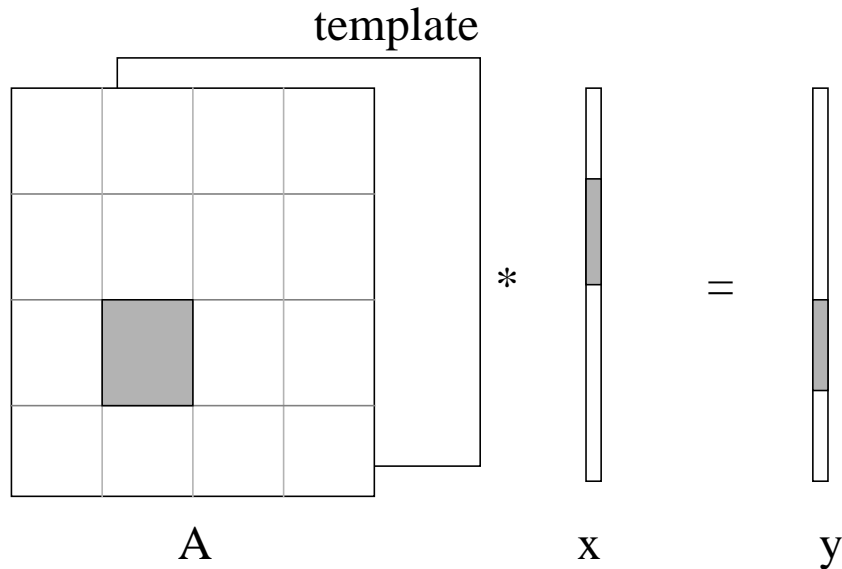
CEPBA



D A C

High Performance Fortran (HPF)

□ Directivas para la distribución de datos



Real*8

TEMPLATE

ALIGN

ALIGN

ALIGN

DISTRIBUTE (BLOCK,BLOCK)

A(N,N), x(N), y(N)

t(N,N)

A(i,j), t(i,j)

y(i), t(i,*)

x(j), t(*,j)

t(N,N)

High Performance Fortran (HPF)

□ Primitivas para la expresión del paralelismo

```
DO J=1, N
  DOALL I=1,N
    Y(I) = Y(I)+A(I,J)*X(J)
  ENDDO
ENDDO
```

Librerías Numéricas Paralelas

□ Librerías Numéricas

- ✓ Las librerías numéricas son colecciones de rutinas que implementan operaciones habituales en aplicaciones científicas y de ingeniería.

- ✓ Las librerías pueden optimizarse para cualquier computador. Los fabricantes de supercomputadores suelen ofrecer una implementación eficiente de las librerías más populares en su computador.

- ✓ Las ventajas del uso de librerías son:
 - Facilitan la construcción de aplicaciones numéricas.

 - Facilitan la portabilidad de la aplicación sin sacrificar la eficiencia (si existe una implementación eficiente de la librería en el computador al que se transporta la aplicación).

Librerías Numéricas Paralelas

□ Algunas Librerías Numéricas

✓ BLAS (Basic Linear Algebra Subprograms)

Es una colección de rutinas que implementan operaciones básicas de álgebra lineal. Se organizan en tres niveles:

BLAS 1: Son operaciones **vector-vector**.

Producto Interno

AXPY

BLAS 2: Son operaciones **matriz-vector**

Producto de matrix por vector

Actualizacion Rango-1

BLAS 3: Son operaciones **matriz-matriz**

Producto de matrices

Sistema triangular matricial de ecuaciones



CEPBA



D A C

Librerías Numéricas Paralelas

□ Algunas Librerías Numéricas

✓ LINPACK

Operaciones de álgebra lineal de mayor complejidad que BLAS. De hecho, las rutinas de LINPACK usan rutinas BLAS1 y BLAS2.

Descomposición LU

Descomposición QR

✓ EISPACK

Operaciones para el cálculo de valores y vectores propios

Librerías Numéricas Paralelas

□ Algunas Librerías Numéricas Paralelas

✓ LAPACK

Operaciones contenidas en LINPACK y EISPACK pero optimizadas para multiprocesadores con memoria compartida, capacidad vectorial y jerarquía de memoria. Las rutinas LAPACK usan rutinas BLAS3.

✓ ScaLAPACK

Operaciones incluidas en LAPACK pero optimizadas para multiprocesadores con memoria distribuida.