Scalable Modeling System

SMS cosè?

- Creato dalla National Oceanic & Atmospheric Administration.
- Un *tool* basato su direttive al compilatore.
- È usato per parallellizzare codici Fortran (ANSI Fortran 77 più qualche istruzione Fortran 90) su macchine a memoria condivisa ed a memoria distribuita (SGI,Altix,Sun UltraSparc,IBM SP, HP, Linux Clusters).
- Può essere usato per applicazioni su griglia strutturata regolare.

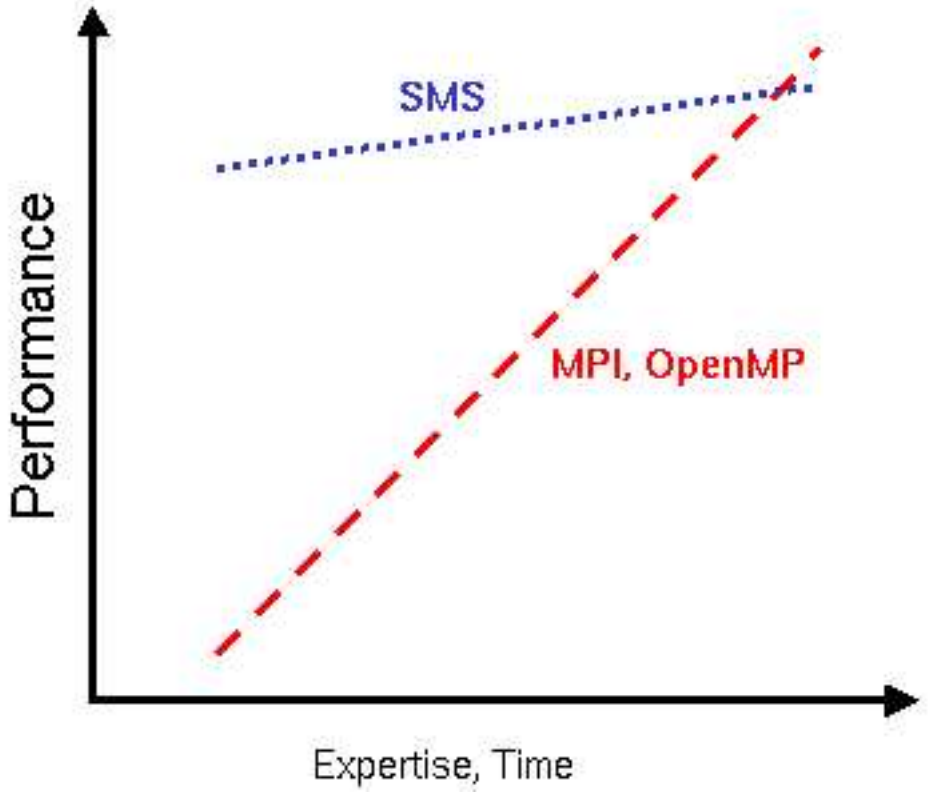
Dal seriale al parallelo

- **Partizionamento dei dati in Memoria**
  - definire le regioni *locali* associate alla singola CPU
- **Gestione dei Do–Loops e degli indici dei vettori**
  - eseguire *do–loops* sulle regioni *locali*
- **Analisi delle Dipendenze**
  - identificare dove devono avvenire le comunicazioni
- **I/O:**
  - mantenere l'ordinamento sequenziale dei dati
- **Debugging:**
  - confrontare i risultati con quelli del codice seriale
- **Ottimizzazione delle Prestazioni:**
  - scalabilità col numero di CPU
- **Portabilità:**
  - *portabilità* delle prestazioni
- **Mantenimento:**
  - mantenere una sola versione seriale e parallela del codice

Obiettivi di SMS

- **Facilità di utilizzo:**
  - minimizzare i cambiamenti nel codice
  - effettuare parellizzazione incrementale
  - supporto per il *debugging*
- **Alte Prestazioni:**
  - non degradare le (alte) prestazioni del codice seriale
- **Correttezza:**
  - avere gli stessi risultati del codice seriale per un numero qualsiasi di CPU
- **Portabilità:**
  - I/O
  - portabilità delle prestazioni
  - rendere trasparenti eventuali ottimizzazioni specifiche del *vendor*
- **Flessibilità:**
  - codice non bloccato ad un numero fissato di CPU

Benefici dell'utilizzo di SMS



Caratteristiche delle direttive

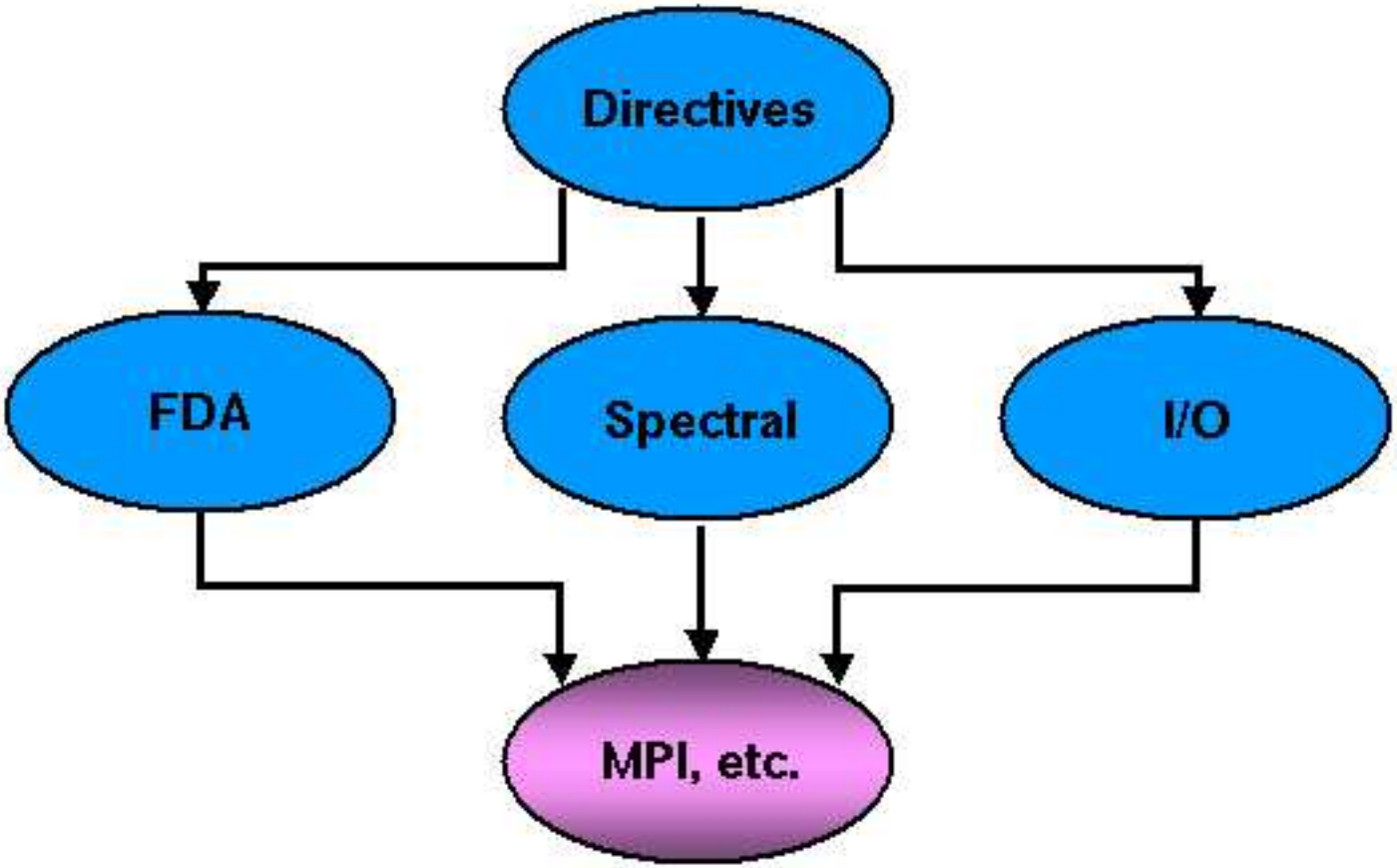
- Iniziano con $CSMS$\$ (!SMS\$ per il free format).
- Non sono CASE SENSITIVE.
- Non possono avere più di 72 caratteri per linea.
- Possono continuare ad un'altra linea (($CSMS$\$ >).
- Possono essere incluse in include/module file.
- Possono essere applicate a multipili di linee con la sintassi begin/end.

Esempio di codice

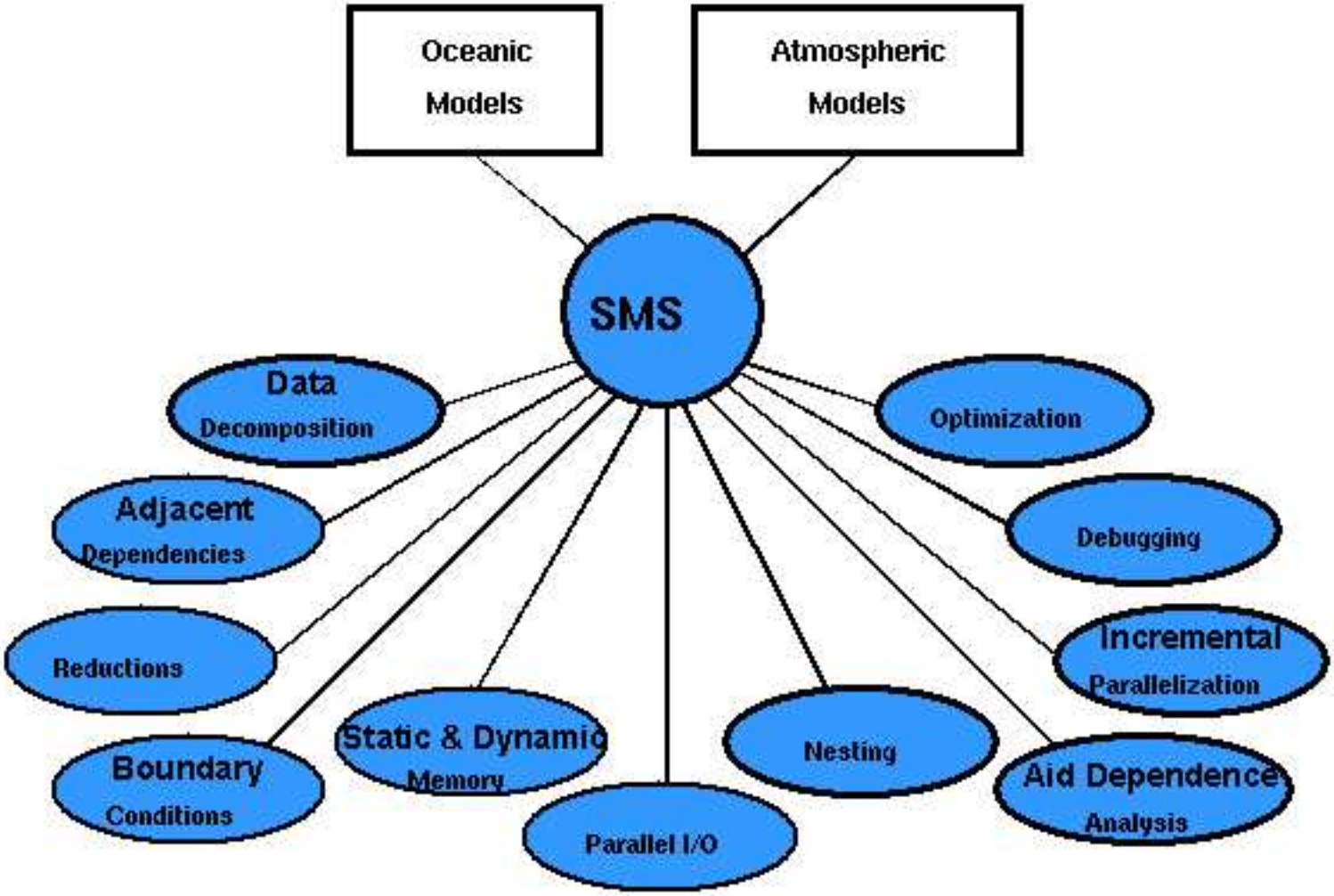
```
      program prova
      parameter(IM=15)

CSMS$DECLARE_DECOMP(my_dh,1)
CSMS$DISTRIBUTE(my_dh,<IM>) BEGIN

      real X(IM),Y(IM)

CSMS$DISTRIBUTE END
CSMS$CREATE_DECOMP(my_dh,<IM>,2)

open(10,file='x_in.dat',form='unformatted')
         read(10) X

CSMS$PARALLEL(my_dh,<i>) BEGIN

         do i=3,13
          y(i)=x(i)-x(i-1)-x(i+1)-x(i-2)-x(i+2)
         end do

CSMS$CHECK_HALO(Y,'prima del loop 1')
CSMS$EXCHANGE(y)

          do i=3,13
           x(i)=y(i)-y(i-1)-y(i+1)-y(i-2)-y(i+2)
```

```
   end do

CSMS$COMPARE_VAR(x,'dopo il loop')

           xsum=0.0
           do i=1,15
            xsum=xsum+x(i)
           end do

CSMS$REDUCE(xsum,SUM)
CSMS$PARALLEL END

          print*,'xsum=',xsum
           end
```

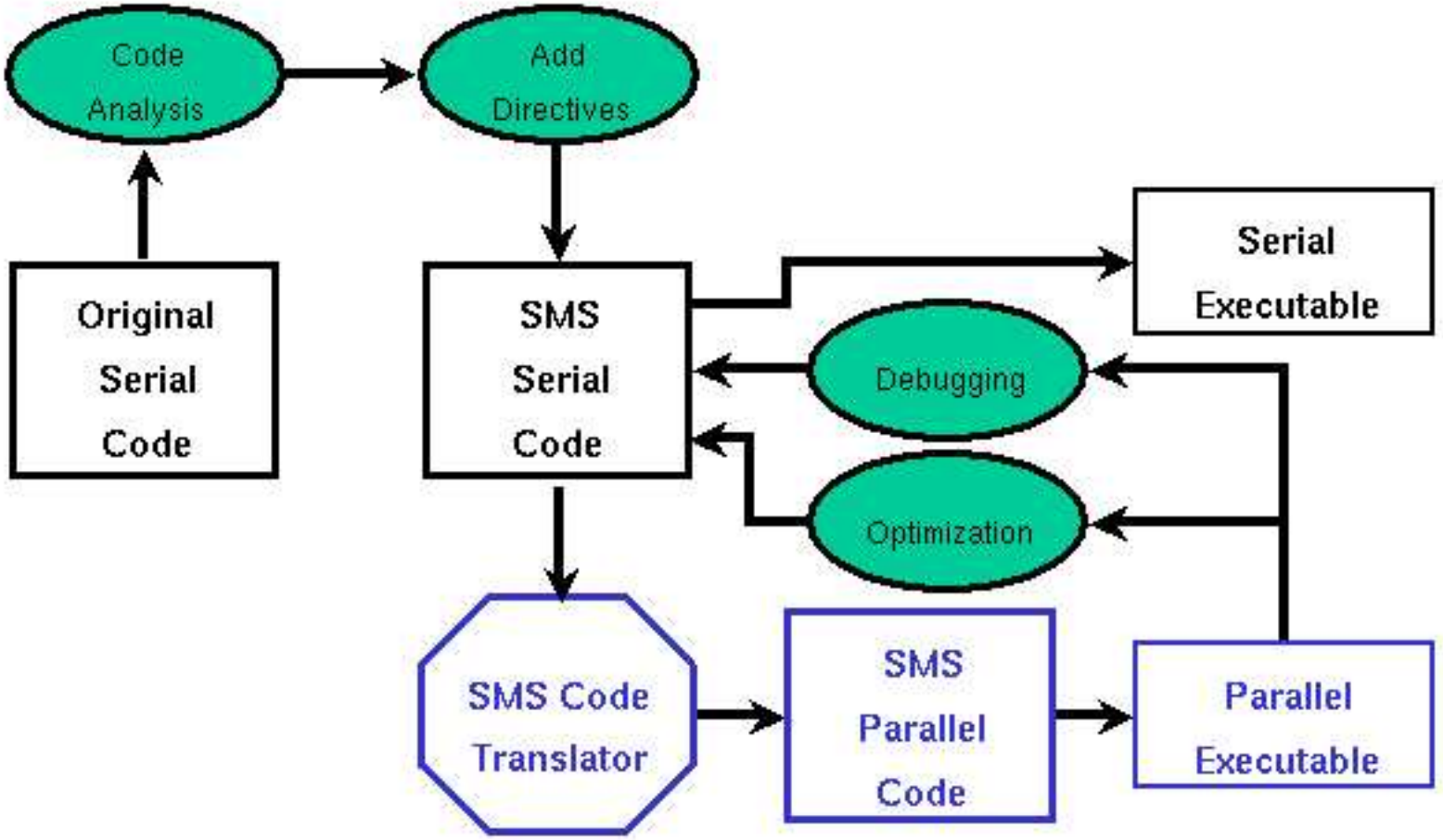
Struttura di SMS



Caratteristiche di SMS

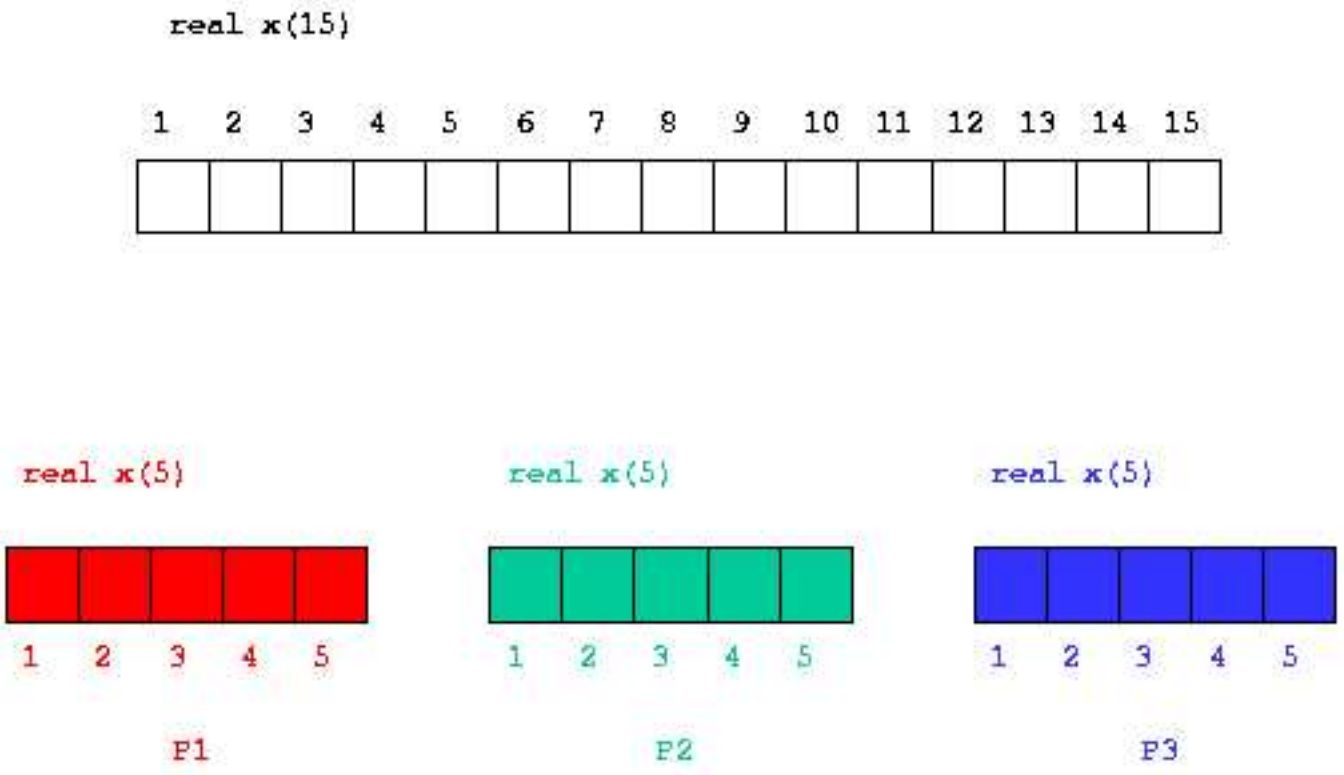


La *Roadmap* di SMS

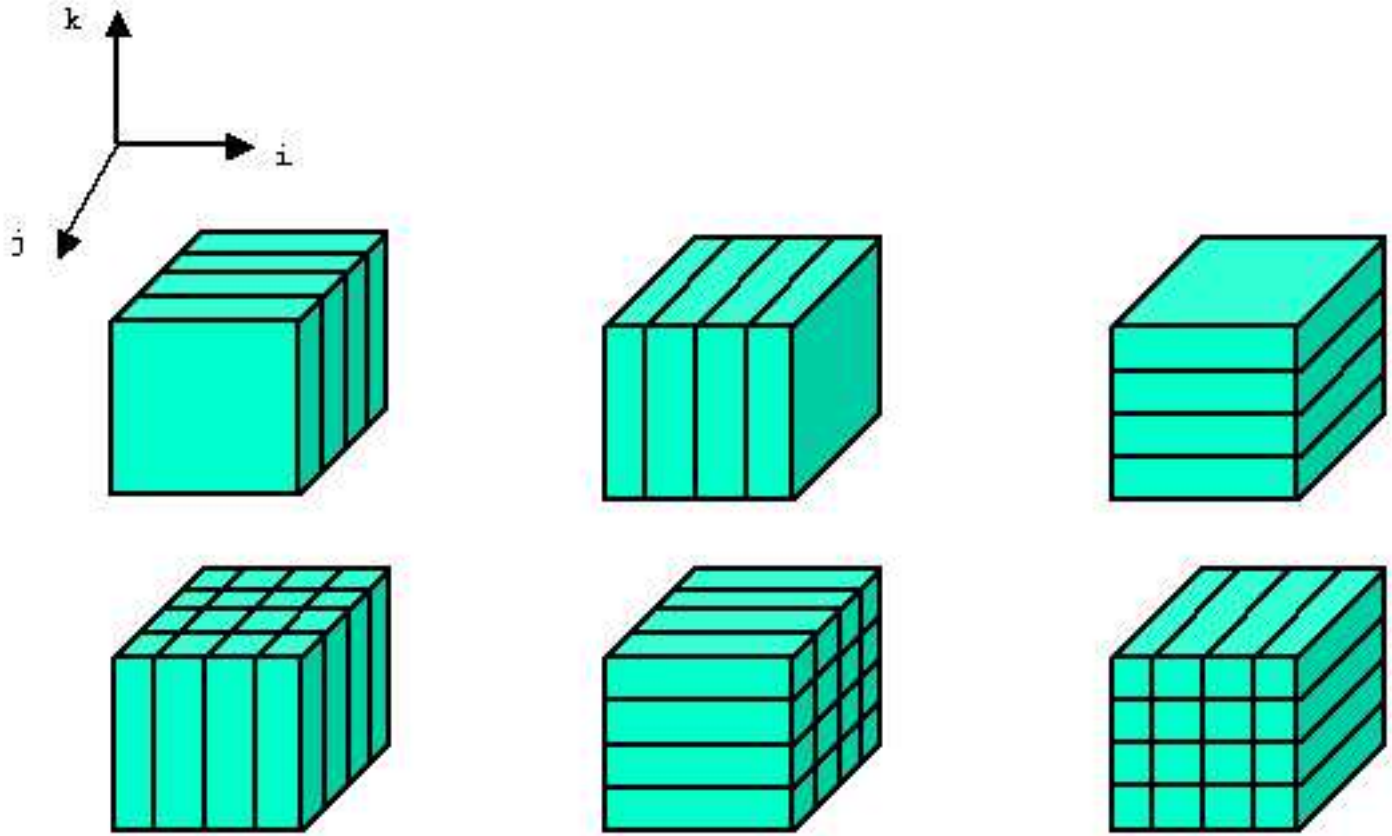


Decomposizione dei dati

Semplice decomposizione $1D$: memoria statica



Tipi di decomposizione dei dati



Direttive SMS: CSMS$DECLARE_DECOMP, CSMS$CREATE_DECOMP

Direttive di decomposizione

Nel caso di memoria statica
CSMS$DECLARE_DECOMP(nome_decomposizione,
<taglia_locale_dim1_dec,taglia_locale_dim2_dec>:<lower1,lower2>)

Nel caso di memoria dinamica
CSMS$DECLARE_DECOMP(nome_decomposizione,numero_dimensioni_decomposte)

- Direttiva dichiarativa (non eseguibile).
- Rende il nome della decomposizione visibile alle altre direttive.
- Dichiara strutture interne ad SMS.
- Comunica il numero delle dimensioni decomposte (al massimo 2).
- Deve essere usata una sola per una specifica decomposizione.

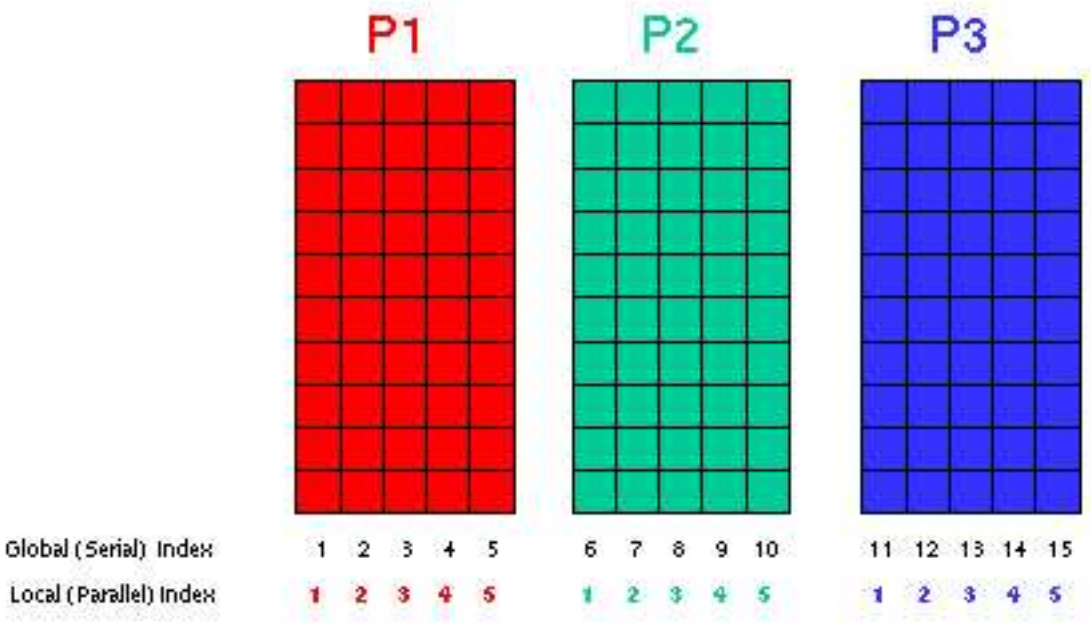
Direttive di decomposizione

CSMS$CREATE_DECOMP(nome_decomposizione,
dim1_taglia_globale,dim2_taglia_globale,
<dim1_larghezza_halo>,<dim2_larghezza_halo>)

dim1(2)_taglia_globale: taglia globale della prima (seconda) dimensione decomposta
dim1(2)_larghezza_halo: deve essere uguale alla massima taglia dello stencil sull'intero modello per la prima(seconda) dimensione.

- Una direttiva eseguibile che deve essere piazzata all'inizio del codice.
- Comunica ad SMS i valori tra parentesi.
- Inizializza strutture interne di dati SMS.
- Genera i limiti della memoria locale nei processori.

Divisione statica bilanciata in memoria

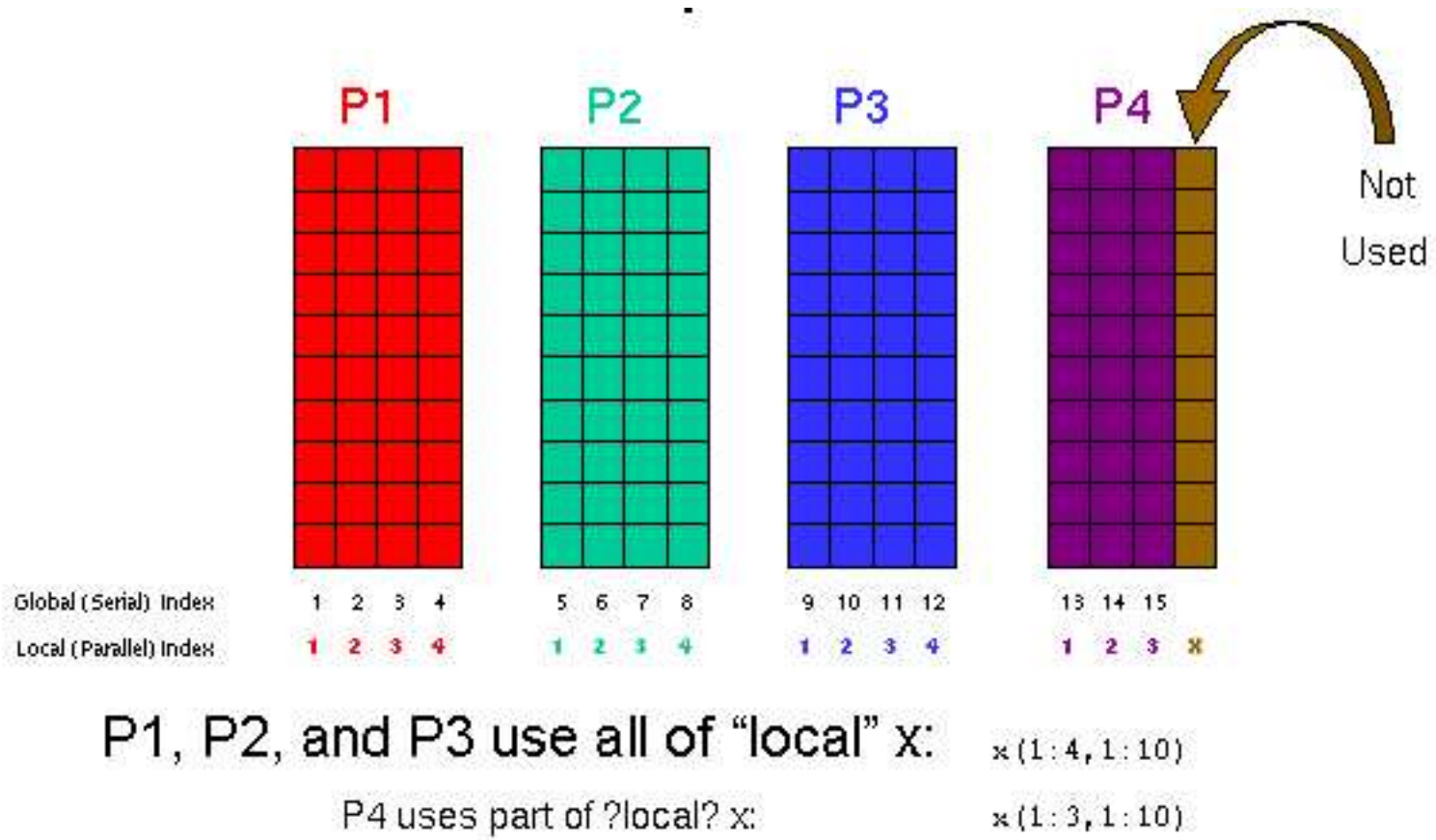


Serial Declaration: `real x(15,10)`

Parallel Declaration: `real x( 5,10)`

Direttiva SMS: CSMS$DISTRIBUTE

Caso sbilanciato



Direttiva SMS: CSMS$DISTRIBUTE

Direttive di decomposizione

Nel caso di memoria statica
CSMS$DISTRIBUTE(nome_decomposizione,<tag_dimensione1>,<tag_dimensione2>)
$CSMS\$ >$ BEGIN
 codice seriale
CSMS$DISTRIBUTE END

Nel caso di memoria dinamica
CSMS$DISTRIBUTE(nome_decomposizione,indice1_decomposizone, indice2_decomposizone)
$CSMS\$ >$ BEGIN
 codice seriale
CSMS$DISTRIBUTE END

- Una direttiva dichiarativa (non eseguibile).
- Specifica come gli array vengono decomposti.
- Modifica la dichiarazione e l'allocazione degli array con processi locali equivalenti.
- Associa gli array con le decomposizioni:
  - come HALO_UDATE,SERIAL,CHECCK_HALO,COMPARE_VAR
  - usata per il Fortran I/O.
- Se ne possono dichiarare più di una per decomposizione

Direttive di decomposizione:esempio

La prima e la seconda dimensione di X1 decomposte usando la prima e la seconda dimensione di my_dh:

```
CSMS$DECLARE_DECOMP(my_dh,2)
CSMS$DISTRIBUTE(my_dh,<IM>,<JM>) begin
 real :: X1(IM,JM,KM)
CSMS$DISTRIBUTE end
```

```
real X1(start1:end1,start2:end2,KM)
```

Trasformazione dei Do–Loops

```
C original serial loop
      do i=1,15


C parallel loops over the sub-domain of each process
C P1 -->
      do i=1,4
C P2 -->
      do i=1,4
C P3 -->
      do i=1,4
C P4 -->
      do i=1,3
```

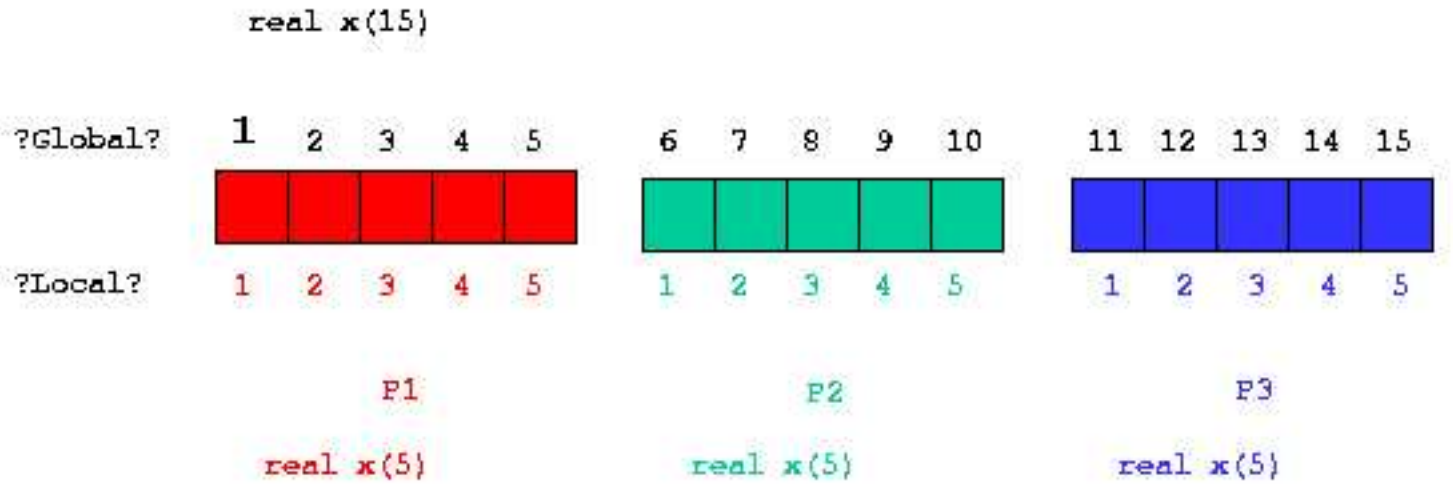
Direttiva SMS: CSMS$PARALLEL

Trasformazione dei Do–Loops

CSMS$PARALLEL(nome_decomposizione,<var_dim1_loop>,CSMS$><var_dim2_loop>)
CSMS$> begin
 codice seriale
SMS$PARALLEL end

- Comunica ad SMS come e dove i loop devono essere tradotti nel loro equivalente parallelo.
- Definisce una "regione parallela".
- L'utente deve specificare gli indici dei loop da trasformare.

Indici locali e globali: caso statico



Direttive SMS:

CSMS$TO_LOCAL

CSMS$TO_GLOBAL

CSMS$GLOBAL_INDEX (usato nelle condizioni al bordo)

Indici locali e globali: caso statico

CSMS$TO_GLOBAL(indice1_decomp,var1,indice2_decomp,var2) begin

codice seriale

CSMS$TO_GLOBAL end

Converte la variabile(var), usata come indice di un array, dal processore locale al loro corrispondente valore globale.

CSMS$TO_LOCAL(indice1_decomp,var1,indice2_decomp,var2:size) begin

codice seriale

CSMS$TO_LOCAL end

Converte il valore di una variabile dal suo valore globale al suo valore locale al processore.

Se viene usata size la variabile viene convertita nella taglia dichiarata localmente della dimensione decomposta specificata.

CSMS$TO_GLOBAL: esempio

```
 program tran_index1
      implicit none
      integer i, j
      integer, parameter :: im = 5
      integer, parameter :: jm = 3

CSMS$DECLARE_DECOMP(DECOMP_IJ, <im, jm>)
CSMS$DISTRIBUTE(DECOMP_IJ, <im>, <jm>) BEGIN

      integer x(im,jm)

CSMS$DISTRIBUTE END
CSMS$CREATE_DECOMP(DECOMP_IJ, <im, jm>, <0,0>)
CSMS$PARALLEL(DECOMP_IJ,<i>,<j>) BEGIN

      do j=1,jm
        do i=1,im

CSMS$TO_GLOBAL(<1,i>, <2,j>) BEGIN

        x(i,j) = (100 * i) + j

CSMS$TO_GLOBAL END

        end do
```

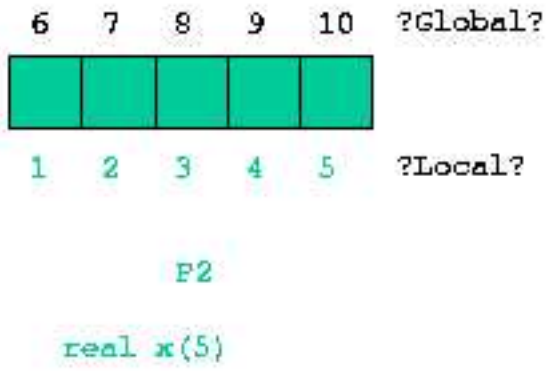
```
      end do

CSMS$SERIAL BEGIN

      do j = 1, jm
        write(*,fmt='("SMS_REG: ", (16i5))') (x(i,j),i=1,im)
      end do

CSMS$SERIAL END
CSMS$PARALLEL END

      end
```

CSMS$TO_LOCAL: esempio

```
CSMS$DISTRIBUTE(decomp,nx,ny) BEGIN

      real a(nx,ny,kk)
      real b(nx,ny,kk)

CSMS$DISTRIBUTE END
CSMS$PARALLEL(decomp) BEGIN
CSMS$TO_LOCAL(<1,nx>,<2,ny>:size) BEGIN

       call compute(a,b,nx,ny)

CSMS$TO_LOCAL END
CSMS$PARALLEL END

subroutine compute(a,b,m,n)
    integer m,n
    real a(m,n),b(m,n)

    do i=1,m
     do j=1,n
      calcoli locali
     end do
    end do
```

Condizioni al bordo

```
C Serial code
      x(8) = 0.0


C Parallel code
C P1 -->
      do NOTHING
C P2 -->
      x(3) = 0.0
C P3 -->
      do NOTHING
```



Condizioni al bordo

Tipiche per codici in ambito scientifico (PDE).

```
CSMS$GLOBAL_INDEX(dim1,dim2) BEGIN
 codice seriale
CSMS$GLOBAL_INDEX END
```

- Le direttive comprese tra BEGIN e END vengono eseguite solo da un determinato processore.
- Deve essere usata all'interno di una regione parallela da cui prende le caratteristiche della decomposizione.

CSMS$GLOBAL_INDEX:esempio

```
program tran_index5
      include 'tran1.inc'
      im = 5
      jm = 3

CSMS$CREATE_DECOMP(DECOMP_IJ, <im,jm>, <0,0>)

      call compute
      end

      subroutine compute
      include 'tran1.inc'

CSMS$DISTRIBUTE(DECOMP_IJ, <im>, <jm>) BEGIN

      integer x(im,jm)

CSMS$DISTRIBUTE END

      integer i, j

CSMS$PARALLEL(DECOMP_IJ,<i>,<j>) BEGIN

      do 100 j=1,jm
      do 100 i=1,im
```

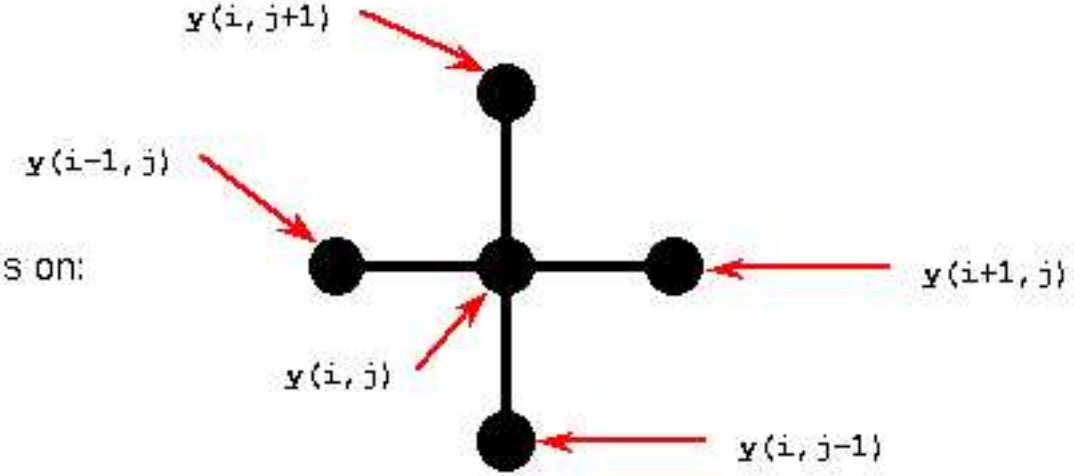
```
CSMS$TO_GLOBAL(<1,i>, <2,j>) BEGIN

        x(i,j) = (100 * i) + j

CSMS$TO_GLOBAL END

  100 continue
      do 110 j=2,jm-1

CSMS$GLOBAL_INDEX(1) BEGIN

        x( 1,j) = 0
        x(im,j) = 0

CSMS$GLOBAL_INDEX END

  110 continue

CSMS$GLOBAL_INDEX(1,2) BEGIN

      x( 1, 1) = 0
      x(im, 1) = 0
      x( 1,jm) = 0
      x(im,jm) = 0
```

```
CSMS$GLOBAL_INDEX END
CSMS$PARALLEL END
CSMS$SERIAL BEGIN

      print *,'SMS_REG: ARRAY x:'
      print *, 'SMS_REG: ', x

CSMS$SERIAL END

      return
      end

[tran1.inc]
integer im, jm
      common /sizes_com/ im, jm

CSMS$DECLARE_DECOMP(DECOMP_IJ, 2)
```

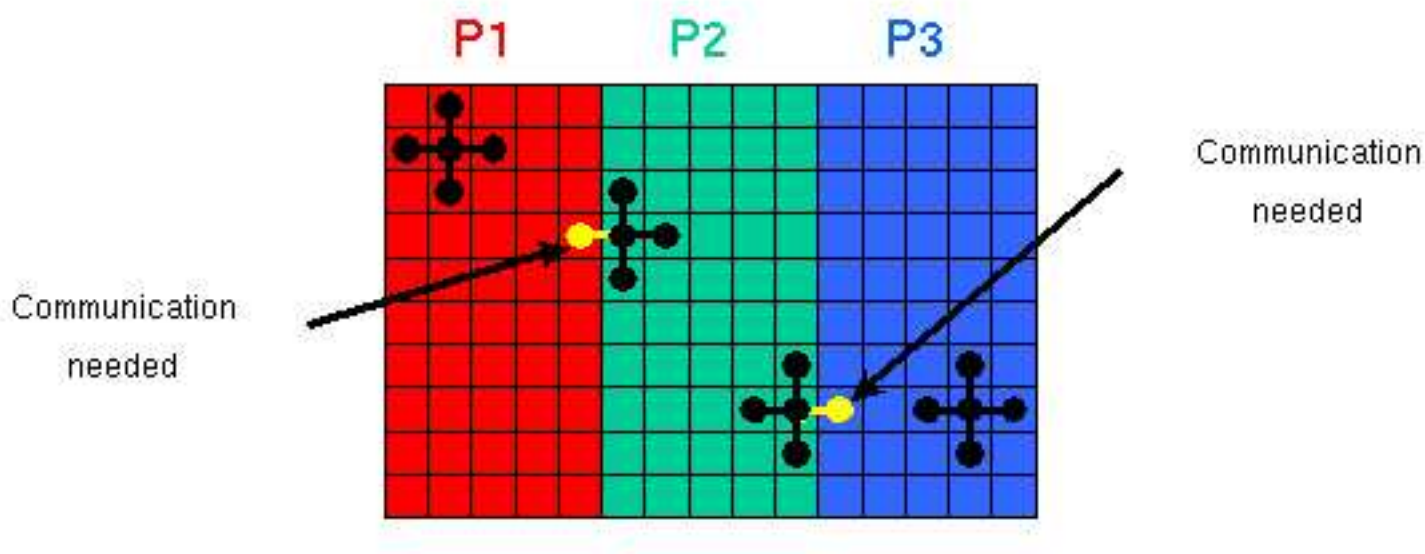
Calcolo a primi vicini


```
x(i,j) = y(i,j) + y(i+1,j) + y(i-1,j) + y(i,j-1) + y(i,j+1)
```

?Stencil?: `x(i,j)` depends on:

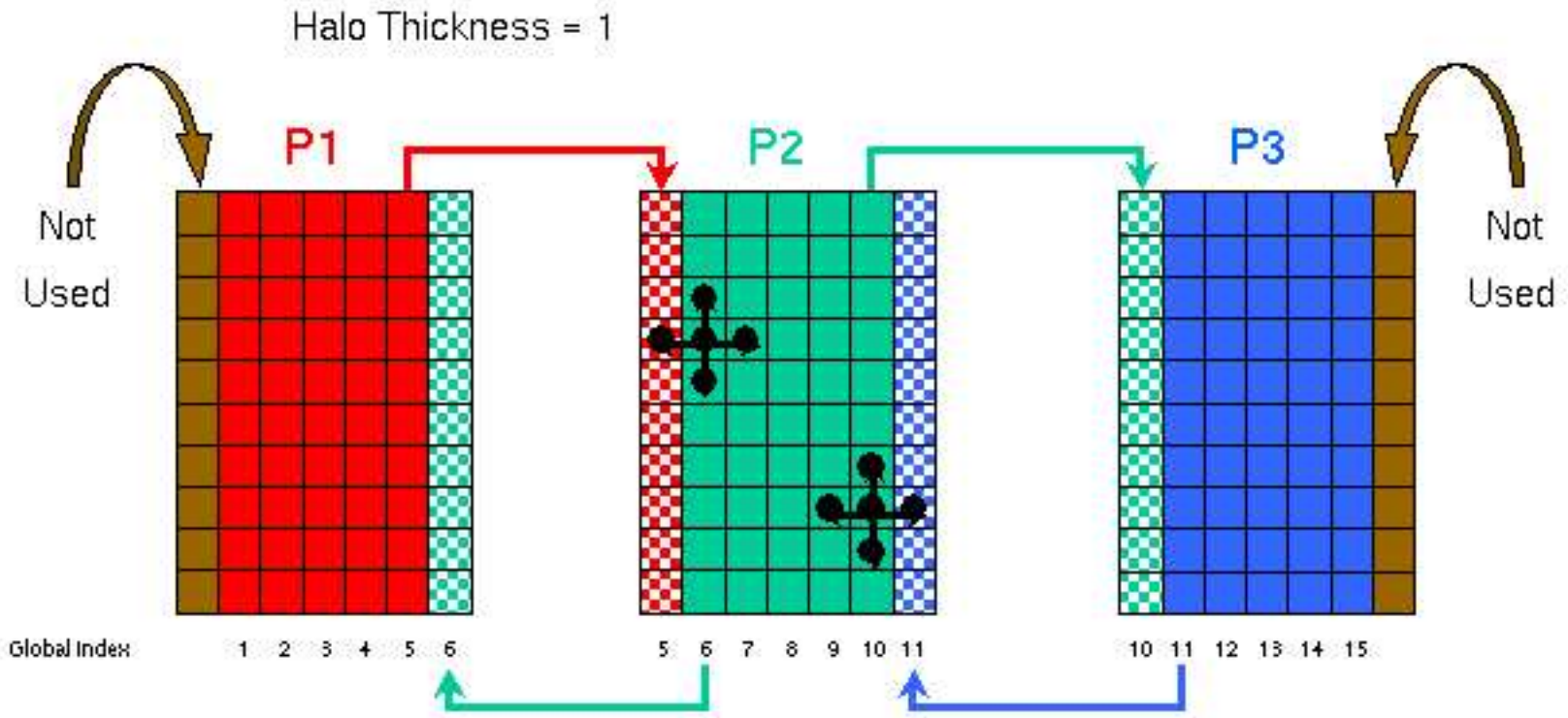


Calcolo a primi vicini: regioni di *halo*




```
y(i,j) = x(i,j) + x(i+1,j) + x(i-1,j)
       + x(i,j+1) + x(i,j-1)
```

Aggiornamento delle regioni di *halo*



Direttiva SMS:

CSMS$EXCHANGE

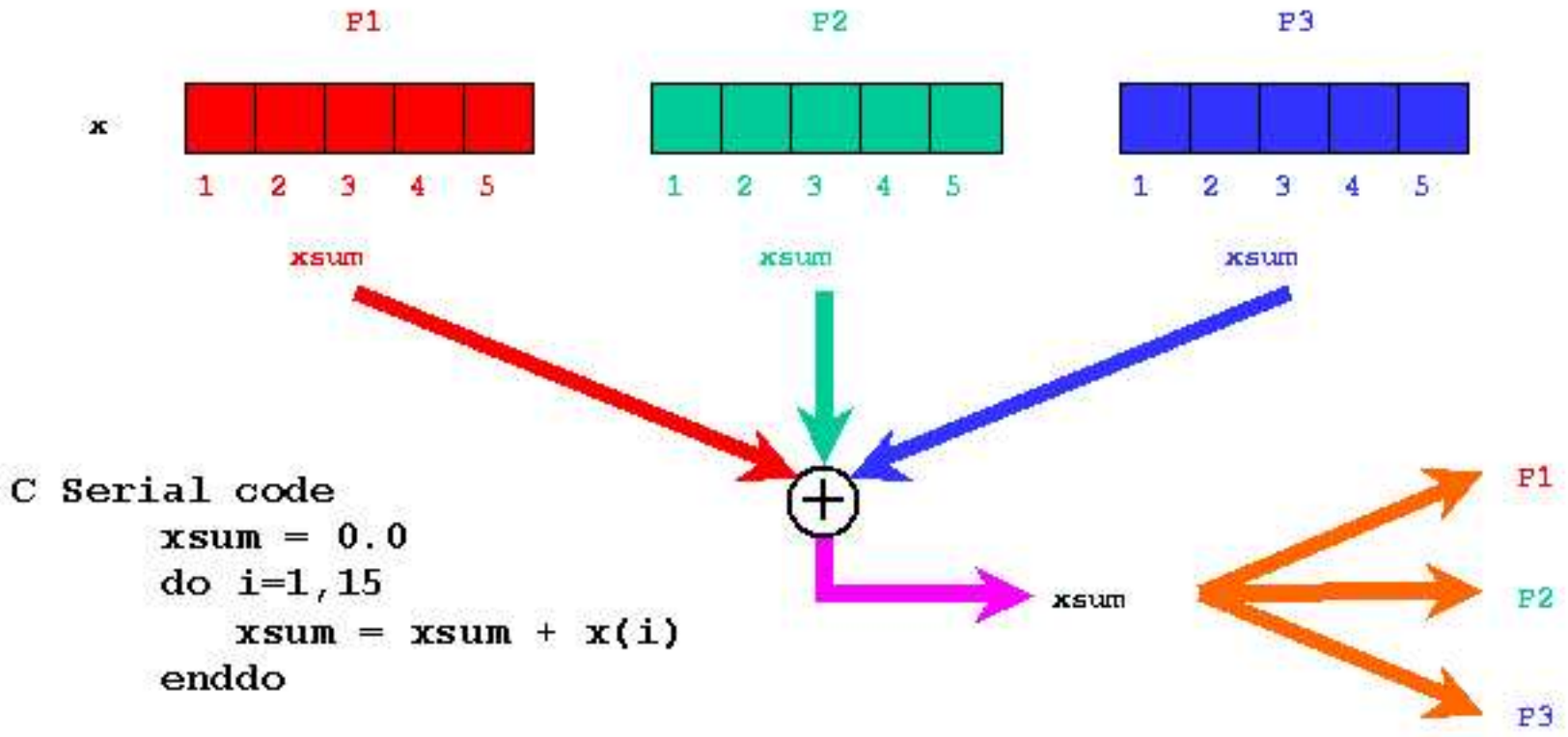
Aggiornamento delle regioni di *halo*

CSMS$EXCHANGE(var1,var2,..,var*n*)

Riferendosi alle variabili indicate tra parentesi:

- Aggiorna le regioni di *halo* con i processori vicini per rendere possibile i calcoli.
- Invia i dati appropriati ai vicini.
- Conserva nelle regioni di *halo* i dati ricevuti.

Riduzione *non esatta*



Direttiva SMS:

CSMS$REDUCE

Riduzione *non esatta*

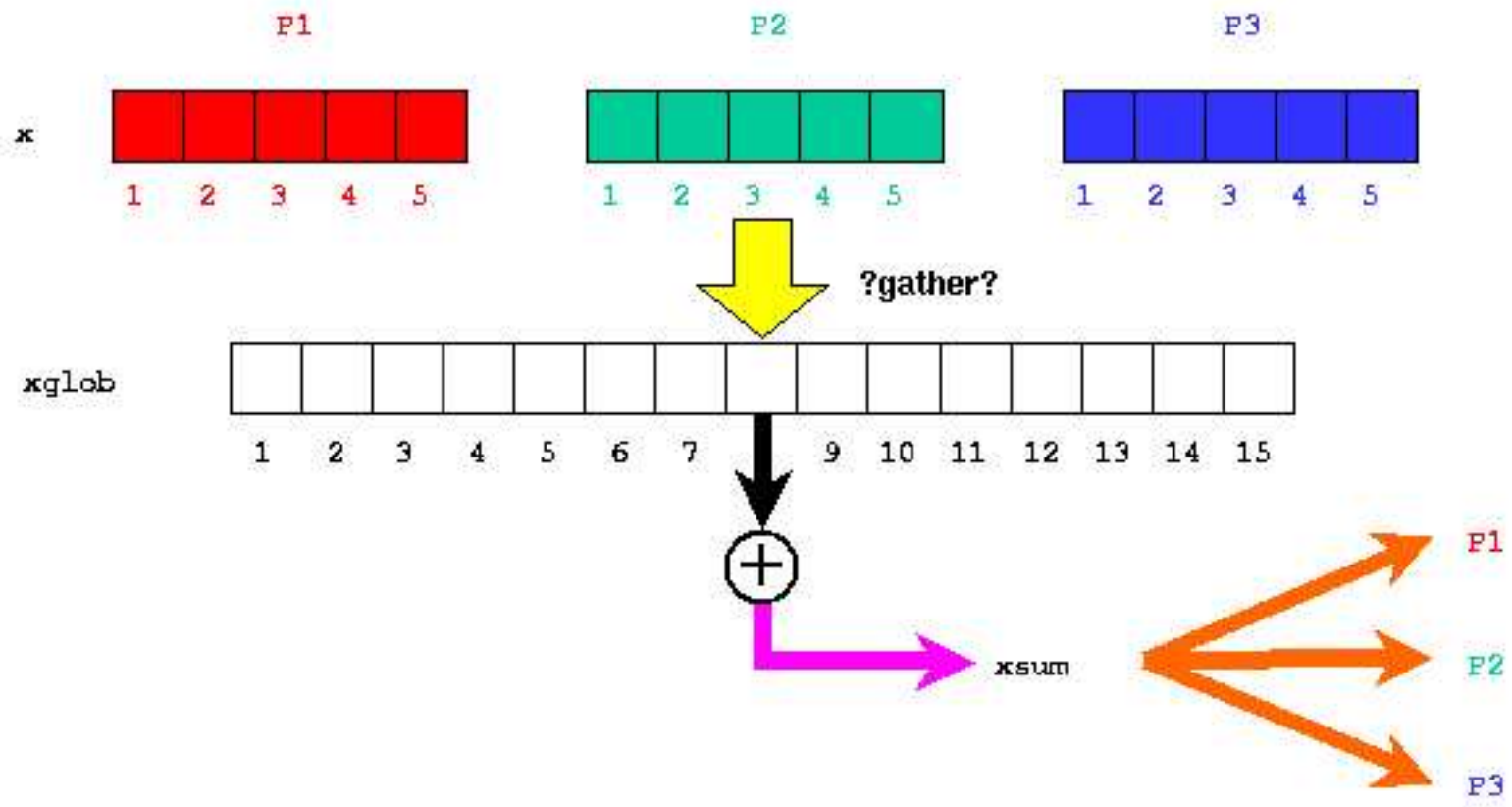
CSM$REDUCE(var1_nondecomp,var2_nondecomp,...,OP)

"OP"="SUM","MIN","MAX"

Determina la Somma,Massimo e Minimo globale su tutti i processori.
Ad esempio per la somma:

- Calcola la "somma locale" su ogni processore.
- Somma le "somme locali" sui processori per ottenere il risultato finale.
- Il risultato differisce lievemente al variare del numero dei processori; l'addizione floating point non è associativa.

Riduzione *bitwise esatta*



Direttiva SMS:

CSMS$REDUCE

Riduzione *bitwise esatta*

```
CSMS$REDUCE(var_nondecomp,SUM) BEGIN
...
CSMS$REDUCE END
```

- Il risultato è indipendente dal numero dei processori.
- Sono penalizzate le performance.
- Viene usata in fase di *debugging* del codice.
- Per usarla definire la variabile ambiente SMS_BITWISE EXACT.

Riduzione *bitwise esatta*

```
CSMS$PARALLEL(my_decomp,<i>) BEGIN
 xsum=0.0
c-SMS do i=1,15
c-SMS xsum=xsum+x(i)
c-SMS end do
CALL SMS_GATHER(x,x_global)
do I=1,15
xsum=xsum+x_global(i)
end do
CALL SMS_BROADCAST(xsum)
```

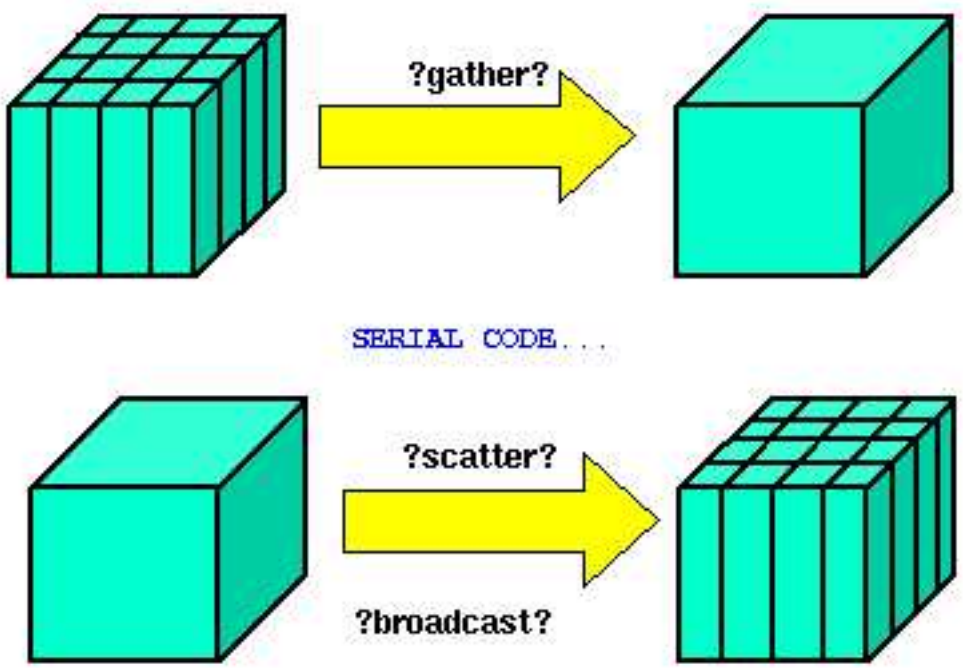
Esempio

```
program SLOW
      implicit none
      integer im
      parameter(im = 30)
      integer jm
      parameter(jm = 5)
      integer iterations
      parameter(iterations = 3)

CSMS$DECLARE_DECOMP(my_dh, <im/3 + 2>)
CSMS$DISTRIBUTE(my_dh, <im>) BEGIN

      real a(im),real b(im,jm),c(im,jm)

CSMS$DISTRIBUTE END

      real ysum
      integer i,j,iter

CSMS$CREATE_DECOMP(my_dh, <im>, <1>)

      ysum = 0.0
      b = 0.0
      c = 0.0
```

```
      do j = 1, jm

CSMS$PARALLEL(my_dh, <i>) BEGIN

        do i = 1, im

CSMS$TO_GLOBAL(<1, i>) BEGIN

          a(i) = real(3*i + 2 + j)

CSMS$TO_GLOBAL END

        end do
        do iter = 1, iterations

CSMS$EXCHANGE(a)

          do i = 2, im-1
            b(i,j) = a(i+1) + a(i-1)
            c(i,j) = b(i,j) + c(i,j)
          end do

CSMS$EXCHANGE(b)
CSMS$EXCHANGE(c)
```

```
          do i = 2, im-1
            a(i) = b(i+1,j) + b(i-1,j) + c(i+1,j) - c(i-1,j)
          end do
        end do
        do i = 2, im - 1
          ysum = ysum + a(i)
        end do
      end do

CSMS$REDUCE(ysum, SUM)

      print *, 'SMS_REG: ysum is ', ysum

CSMS$PARALLEL END

      end
```

Parallelizzazione incrementale



Direttiva SMS:

CSMS$SERIAL

Parallellizzazione incrementale

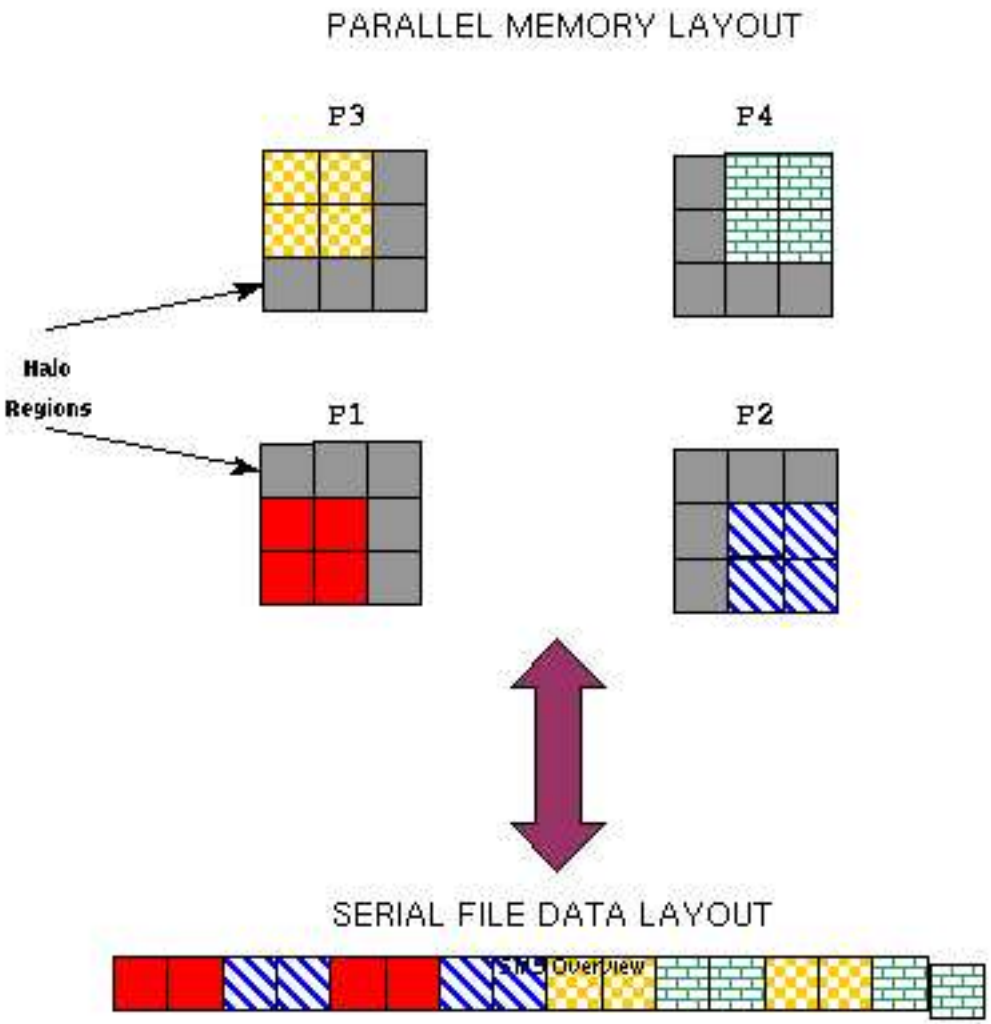
CSMS$SERIAL(specifica_comunicazioni) BEGIN
...
CSMS$SERIAL END

- Il codice compreso tra BEGIN e END viene eseguito dal processore designato come *root*.
- Risulta utile quando:
  - le direttive SMS non possono essere applicate a quella parte di codice;
  - le performance non sono determinanti (fase di inizializzazione).
- Specifica_comunicazioni serve per migliorare le performance.

I/O *non formattato*

- Il formato di *default* è il *binary* nativo del Fortran.
- Sono supportati altri formati tra cui quelli contenuti in MPI-IO.
  - il formato si può scegliere a *runtime* utilizzando variabili di ambiente.
- Non sono richieste direttive SMS.
- È consentito mischiare variabili decomposte e non.
- L'ordinamento seriale dei dati è mantenuto.

I/O di dati decomposti



I/O formattato

Viene gestito in modo automatico con le seguenti eccezioni:

- in Input non posso essere letti dati decomposti, usare CSMS$SERIAL
- in Output il comportamento dell'istruzione print

I/O formattato

CSMS$PRINT_MODE(mode) BEGIN
....
CSMS$PRINT_MODE END

Per la variabile mode sono possibili i seguenti valori:
ASYNC: ogni processore stampa la sua stringa quando esegue questa istruzione. I processori non sono sincronizzati. L'ordine dei messaggi può differire da un run all'altro.
ORDERED: in ordine fissato, ogni processore stampa la sua stringa.Viene usato per il debugging. I processori sono sincronizzati (possibile deadlock).
ROOT Il processore designato come *root* stampa la sua stringa. I processori non sono sincronizzati.

CSM$PRINT_MODE:esempio

```
program print_modes
      implicit none
      integer, parameter :: im = 12
      integer xmax, i

CSMS$DECLARE_DECOMP(dh, 1)
CSMS$DISTRIBUTE(dh, 1) BEGIN

      integer, allocatable :: x(:)

CSMS$DISTRIBUTE END
CSMS$CREATE_DECOMP(dh, <im>, <0>)

      allocate(x(im))

CSMS$SERIAL BEGIN

      do i = 1, im
        x(i) = i
      end do

CSMS$SERIAL END
CSMS$PARALLEL(dh,<i>) BEGIN

      xmax = 0
```

```
      do i = 1,im
        xmax = max(xmax,x(i))
        if (x(i) .ge. 9) then

CSMS$PRINT_MODE(ASYNC) BEGIN

          print *,'SMS_REG: WARNING:  x .ge. 9 !!  '

CSMS$PRINT_MODE END

        endif
      end do

CSMS$PARALLEL END
CSMS$PRINT_MODE(ORDERED) BEGIN
CSMS$INSERT      print *,'SMS_REG: DEBUG:  local maximum value = ',xmax
CSMS$REDUCE(xmax,MAX)
CSMS$PRINT_MODE(ROOT) BEGIN

      write( *,900) 'SMS_REG: maximum value = ',xmax
  900 format(a, i4)

CSMS$PRINT_MODE END

      end
```

CSM$PRINT_MODE:esempio

CSMS$INSERT
Permette di specificare righe di codice che devono essere parallelizzate da ppp.

2 proc

```
SMS_REG: WARNING:  x .ge. 9 !!
SMS_REG: WARNING:  x .ge. 9 !!
SMS_REG: WARNING:  x .ge. 9 !!
SMS_REG: WARNING:  x .ge. 9 !!
SMS_REG: DEBUG:  local maximum value =  6
SMS_REG: DEBUG:  local maximum value =  12
SMS_REG: maximum value =   12
```

Compilazione ed esecuzione (IBM)

Uso di ppp

ppp [opzioni] source_file.f genera source_file_sms.f
Le opzioni:
alcune ricalcano un analogo fortran;
altre servono a creare un codice generato più leggibile o per il debugging.

Compilazione

mpxlf90_r [opzioni] -I/$SMS/include -c source_file_sms.f

"Linkaggio"

mpxlf90_r -o par_code [opzioni] source_file_sms.o -L/$SMS/lib -lsms

Esecuzione

setenv [variabili_ambiente]

smsRun -np [numero_processori] par_code

Equazione di Laplace: seriale

```
program laplace
implicit none
real t(0:1001),tnew(1000),tol
real dif,difmax
integer i,n

print*, 'valori di n e di tol',n,tol
read*,n,tol

do i=0,n
 t(i)=0.0
end do
t(n+1)=100.0

10   continue

do i=1,n
  tnew(i)=(t(i-1)+t(i+1))/2.
end do

difmax=0.0
do i=1,n
  dif= abs(tnew(i)- t(i))
```

```
if(dif.gt.difmax) difmax=dif
t(i)=tnew(i)
end do
if(difmax.gt.tol) goto 10

do i=1,n
  write(*,*) t(i)
end do

end
```

Equazione di Laplace: codice SMS

```
program laplaceSMS
      implicit none
      integer i, n
      real tol,dif,difmax
      real t0,t1

!SMS$DECLARE_DECOMP(DECOMP_IJ, <1001+3>:<0>)
!SMS$DISTRIBUTE(DECOMP_IJ, 1) BEGIN

      real t(0:1001),tnew(1000)

!SMS$DISTRIBUTE END
!SMS$CREATE_DECOMP(DECOMP_IJ, <1001+1>, <1>)
!SMS$SERIAL BEGIN

       open(10,file='input.d',form='formatted')
       read(10,*) n,tol
       print*, 'valori di n e di tol',n,tol

!SMS$SERIAL END
!SMS$PARALLEL(DECOMP_IJ, <i>) BEGIN

        do i=0,n
          t(i)=0.0
```

```
        end do
!SMS$GLOBAL_INDEX(1) BEGIN
        t(n+1)=100.0
!SMS$GLOBAL_INDEX END
10      continue
!sms$EXCHANGE(t)
        do i=1,n
        tnew(i)=(t(i-1)+t(i+1))/2.
        end do
        difmax=0.0
        do i=1,n
        dif= abs(tnew(i)- t(i))
        if(dif.gt.difmax) difmax=dif
        t(i)=tnew(i)
        end do
CSMS$REDUCE(difmax, MAX)
        if(difmax.gt.tol) goto 10
```

```
!SMS$PARALLEL END
!SMS$SERIAL BEGIN

        do i=1,n
        write(*,*) t(i)
        end do

!SMS$SERIAL END

      end
```

Equazione di Laplace: SMS dopo ppp

```
program laplaceSMS
      use nnt_types_module
      implicit none
      integer i, n
      real tol,dif,difmax
C!SMS$DECLARE_DECOMP(DECOMP_IJ, <1001+3>:<0>)
      INTEGER decomp_ij__MAX_GLOBAL_SIZE
      PARAMETER (decomp_ij__MAX_GLOBAL_SIZE = 10000)
      INTEGER decomp_ij__MAX_HALO_THICK
      PARAMETER (decomp_ij__MAX_HALO_THICK = 6)
      INTEGER decomp_ij__PPP_MAX_REGIONS
      PARAMETER (decomp_ij__PPP_MAX_REGIONS = 1)
      INTEGER decomp_ij(1), decomp_ij__Index, decomp_ij__Ignore,
     +   decomp_ij__nRegions, decomp_ij__NestLevel
      CHARACTER*32 decomp_ij__DecompName
      INTEGER decomp_ij__MaxNests
      PARAMETER (decomp_ij__MaxNests = 1)
      INTEGER decomp_ij__NestLevels(decomp_ij__MaxNests)
      INTEGER decomp_ij__LowBound_1
      PARAMETER (decomp_ij__LowBound_1 = 0)
      INTEGER decomp_ij__LowBounds(PPP_MAX_DECOMPOSED_DIMS,
     +   decomp_ij__MaxNests)
      INTEGER decomp_ij__UpperBounds(PPP_MAX_DECOMPOSED_DIMS,
```

```
     +   decomp_ij__MaxNests)
INTEGER decomp_ij__LocalSize(PPP_MAX_DECOMPOSED_DIMS,
     +   decomp_ij__MaxNests)
      INTEGER decomp_ij__GlobalSize(PPP_MAX_DECOMPOSED_DIMS,
     +   decomp_ij__MaxNests)

      INTEGER decomp_ij__Local_UB(PPP_MAX_DECOMPOSED_DIMS,
     +   decomp_ij__MaxNests)
      INTEGER decomp_ij__Local_LB(PPP_MAX_DECOMPOSED_DIMS,
     +   decomp_ij__MaxNests)

      INTEGER decomp_ij__HaloSize(PPP_MAX_DECOMPOSED_DIMS,
     +   decomp_ij__MaxNests)
      INTEGER decomp_ij__BoundaryType(PPP_MAX_DECOMPOSED_DIMS)

      INTEGER decomp_ij__Declared_Size_1
      PARAMETER (decomp_ij__Declared_Size_1 = 1001+3)
      INTEGER decomp_ij__Local_to_Global1(decomp_ij__LowBound_1:
     +   decomp_ij__MAX_GLOBAL_SIZE+decomp_ij__LowBound_1-1,
     +   decomp_ij__MaxNests)
      INTEGER decomp_ij__Global_to_Local1(decomp_ij__LowBound_1:
     +   decomp_ij__MAX_GLOBAL_SIZE+decomp_ij__LowBound_1-1,
     +   decomp_ij__MaxNests)
```

```
      INTEGER decomp_ij__S1(decomp_ij__LowBound_1:
     +   decomp_ij__MAX_GLOBAL_SIZE+decomp_ij__LowBound_1-1, 0:
     +   decomp_ij__MAX_HALO_THICK, decomp_ij__MaxNests)
      INTEGER decomp_ij__E1(decomp_ij__LowBound_1:
     +   decomp_ij__MAX_GLOBAL_SIZE+decomp_ij__LowBound_1-1, 0:
     +   decomp_ij__MAX_HALO_THICK, decomp_ij__MaxNests)

 COMMON /decomp_ij__common/ decomp_ij, decomp_ij__nRegions,
     +   decomp_ij__NestLevel, decomp_ij__NestLevels,
     +   decomp_ij__LocalSize, decomp_ij__GlobalSize,
     +   decomp_ij__BoundaryType, decomp_ij__LowBounds,
     +   decomp_ij__UpperBounds, decomp_ij__HaloSize,
     +   decomp_ij__Local_UB, decomp_ij__Local_LB,
     +   decomp_ij__Local_to_Global1, decomp_ij__Global_to_Local1,
     +   decomp_ij__S1,decomp_ij__E1

        ! ******* END - DECLARE DECOMP TRANSLATION *********

C!SMS$DISTRIBUTE(DECOMP_IJ, 1) BEGIN
C-SMS       real t(0:1001),tnew(1000)
      real t(decomp_ij__LowBound_1:decomp_ij__Declared_Size_1+
     +    decomp_ij__LowBound_1-1),tnew(decomp_ij__Declared_Size_1+
     +    decomp_ij__LowBound_1-1)
```

```
C!SMS$DISTRIBUTE END
      logical PPP_EXACT
integer GlobalSizeR1(PPP_MAX_REDUCE_VARS)
      integer DataTypeR2(PPP_MAX_REDUCE_VARS)
      integer i1
      integer PPP__GlobalSize(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__GlobalUpper(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__GlobalLower(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__Permute(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__GlobalStart(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__GlobalStop(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__HaloLower(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__HaloUpper(PPP_MAX_RANK,PPP_MAX_VARS)
      integer PPP__DecompType(PPP_MAX_VARS)
      integer PPP__DataType(PPP_MAX_VARS)
      logical IAM_ROOT
      integer, save :: PPP__CommunicationTag0 = 0
      real, allocatable :: t__g(:)
      integer PPP__PeriodicUsedLower(PPP_MAX_DECOMPOSED_DIMS)
      integer PPP__PeriodicUsedUpper(PPP_MAX_DECOMPOSED_DIMS)
      integer GlobalLower1(PPP_MAX_RANK,PPP_MAX_EXCHANGE_VARS)
      integer GlobalUpper2(PPP_MAX_RANK,PPP_MAX_EXCHANGE_VARS)
      integer Permute_Decomp3(PPP_MAX_RANK,PPP_MAX_EXCHANGE_VARS)
```

```
      integer GlobalStart4(PPP_MAX_RANK,PPP_MAX_EXCHANGE_VARS)
      integer GlobalStop5(PPP_MAX_RANK,PPP_MAX_EXCHANGE_VARS)
      character*32 VarName6
      logical SMS_DEBUGGING_ON
      integer HaloLower7(PPP_MAX_RANK,PPP_MAX_EXCHANGE_VARS)
      integer HaloUpper8(PPP_MAX_RANK,PPP_MAX_EXCHANGE_VARS)
      integer DataType9(PPP_MAX_EXCHANGE_VARS)
      integer DecompType10(PPP_MAX_EXCHANGE_VARS)
      integer, save :: ExchangeTag11 = 0
      call nnt_init(ppp__status)
      call nnt_chkstat('./laplaceSMS_sms.f:10',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)

C!SMS$CREATE_DECOMP(DECOMP_IJ, <1001+1>, <1>)
      decomp_ij__NestLevel = 1
      decomp_ij__nRegions = 1
      decomp_ij__GlobalSize(1, decomp_ij__NestLevel) = 1001+1
      decomp_ij__GlobalSize(2, decomp_ij__NestLevel) = 1
      decomp_ij__GlobalSize(3, decomp_ij__NestLevel) = 1
      decomp_ij__LocalSize(1, decomp_ij__NestLevel) =
     +   decomp_ij__Declared_Size_1
      decomp_ij__LocalSize(2, decomp_ij__NestLevel) = 1
      decomp_ij__LocalSize(3, decomp_ij__NestLevel) = 1
```

```
 decomp_ij__HaloSize(1, decomp_ij__NestLevel) = 1
 decomp_ij__HaloSize(2, decomp_ij__NestLevel) = 0
 decomp_ij__HaloSize(3, decomp_ij__NestLevel) = 0

 decomp_ij__BoundaryType(1) = NNT_NONPERIODIC_BDY
 decomp_ij__BoundaryType(2) = NNT_NONPERIODIC_BDY
 decomp_ij__BoundaryType(3) = NNT_NONPERIODIC_BDY

 decomp_ij__LowBounds(1,decomp_ij__NestLevel) = 0
 decomp_ij__LowBounds(2,decomp_ij__NestLevel) = 1
 decomp_ij__LowBounds(3,decomp_ij__NestLevel) = 1
 decomp_ij__UpperBounds(1,decomp_ij__NestLevel) =
+   decomp_ij__GlobalSize(1,decomp_ij__NestLevel) +
+   decomp_ij__LowBounds(1,decomp_ij__NestLevel) -1
 decomp_ij__UpperBounds(2,decomp_ij__NestLevel) =
+   decomp_ij__GlobalSize(2,decomp_ij__NestLevel) +
+   decomp_ij__LowBounds(2,decomp_ij__NestLevel) -1
 decomp_ij__UpperBounds(3,decomp_ij__NestLevel) =
+   decomp_ij__GlobalSize(3,decomp_ij__NestLevel) +
+   decomp_ij__LowBounds(3,decomp_ij__NestLevel) -1
 PPP__PeriodicUsedLower(:) = decomp_ij__LowBounds(:,1)
 PPP__PeriodicUsedUpper(:) = decomp_ij__UpperBounds(:,1)
```

```
if ((decomp_ij__GlobalSize(1,decomp_ij__NestLevel)
     +   .gt.decomp_ij__MAX_GLOBAL_SIZE) .or. (decomp_ij__GlobalSize(2,
     +   decomp_ij__NestLevel).gt.decomp_ij__MAX_GLOBAL_SIZE) .or.
     +   (decomp_ij__GlobalSize(3,decomp_ij__NestLevel)
     +   .gt.decomp_ij__MAX_GLOBAL_SIZE)) then
        print *,
     +     'ERROR:  Global Array Sizes are larger than the default '
     +     //'limit.  SMS Exiting ...'
        call nnt_stop('./laplaceSMS_sms.f:15.0',0,PPP_ABORT)
      endif

      if ((decomp_ij__HaloSize(1,decomp_ij__NestLevel)
     +   .gt.decomp_ij__MAX_HALO_THICK) .or. (decomp_ij__HaloSize(2,
     +   decomp_ij__NestLevel).gt.decomp_ij__MAX_HALO_THICK) .or.
     +   (decomp_ij__HaloSize(3,decomp_ij__NestLevel)
     +   .gt.decomp_ij__MAX_HALO_THICK)) then
        print *,
     +     'ERROR:  Halo Array Sizes are larger than the default limit.'
     +     //'  SMS Exiting ...'
        call nnt_stop('./laplaceSMS_sms.f:15.0',0,PPP_ABORT)
       endif

      decomp_ij__DecompName = 'decomp_ij'
```

```
      call ppp_decomp( NNT_DECOMP_1,decomp_ij__BoundaryType,
     +   decomp_ij__GlobalSize(1, decomp_ij__NestLevel),
     +   decomp_ij__HaloSize(1, decomp_ij__NestLevel),
     +   decomp_ij__LowBounds(1, decomp_ij__NestLevel), PPP_NULL_DECOMP,
     +    decomp_ij__LocalSize(1, decomp_ij__NestLevel),
     +   PPP__PeriodicUsedLower(1),PPP__PeriodicUsedUpper(1),
     +   decomp_ij__Local_LB(1, decomp_ij__NestLevel),
     +   decomp_ij__Local_UB(1, decomp_ij__NestLevel),
     +   decomp_ij__DecompName, decomp_ij(decomp_ij__NestLevel),
     +   PPP_MAX_DECOMPOSED_DIMS,ppp__status)
      call nnt_chkstat('./laplaceSMS_sms.f:15.1',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
o decomp_ij__index=0, decomp_ij__HaloSize(1,
     +   decomp_ij__NestLevel)
        call ppp_loops_op(decomp_ij(decomp_ij__NestLevel),1 ,
     +     decomp_ij__HaloSize(1, decomp_ij__NestLevel)-
     +     decomp_ij__Index,decomp_ij__S1(0,decomp_ij__Index,
     +     decomp_ij__NestLevel), decomp_ij__E1(0,decomp_ij__Index,
     +     decomp_ij__NestLevel), 1,decomp_ij__nRegions,ppp__status)
        call nnt_chkstat('./laplaceSMS_sms.f:15.2.0',' ',ppp__status,
     +     NNT_ABORT_ON_ERROR, ppp__status)
      enddo
      call ppp_trans_indx(decomp_ij(decomp_ij__NestLevel),1,
```

```
     +   decomp_ij__Local_to_Global1(0,decomp_ij__NestLevel),
     +   decomp_ij__Global_to_Local1(0,decomp_ij__NestLevel),
     +   decomp_ij__PPP_MAX_REGIONS,decomp_ij__Ignore,ppp__status)
      call nnt_chkstat('./laplaceSMS_sms.f:15.3.0',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)

        ! ******* END - CREATE DECOMP DECLARATION *********
C!SMS$SERIAL BEGIN

      if (IAM_ROOT()) then
       open(10,file='input.d',form='formatted')
       read(10,*) n,tol
       print*, 'valori di n e di tol',n,tol
      endif
 PPP__GlobalSize = 1
      call PPP_BCAST(n,NNT_INTEGER,PPP__GlobalSize,
     +   PPP_MAX_DECOMPOSED_DIMS,ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:18.0',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
      PPP__GlobalSize = 1
      call PPP_BCAST(tol,NNT_REAL,PPP__GlobalSize,
     +   PPP_MAX_DECOMPOSED_DIMS,ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:18.1',' ',ppp__status,
```

```
     +   NNT_ABORT_ON_ERROR, ppp__status)
      PPP__GlobalSize = 1
C!SMS$SERIAL END
C!SMS$PARALLEL(DECOMP_IJ, <i>) BEGIN
C-SMS      do i=0,n
      do i = decomp_ij__S1(0,0,decomp_ij__NestLevel), decomp_ij__E1(n,
     +    0,decomp_ij__NestLevel)
          t(i)=0.0
        end do
C!SMS$GLOBAL_INDEX(1) BEGIN
      do i1 = decomp_ij__S1(n+1,0,decomp_ij__NestLevel),
     +   decomp_ij__E1(n+1,0,decomp_ij__NestLevel)
C-SMS        t(n+1)=100.0
      t(i1) = 100.0

      enddo
C!SMS$GLOBAL_INDEX END
C!sms$compare_var(t,'DOPO')
if (SMS_DEBUGGING_ON()) then
      GlobalLower1(:,1:1) = 1
      GlobalUpper2(:,1:1) = 1
      Permute_Decomp3(:,1:1) = 0
      GlobalStart4(:,1:1) = 1
```

```
      GlobalStop5(:,1:1) = 1

!-SMS          storing information about variable: t

      GlobalLower1(1,1) = 0
      GlobalUpper2(1,1) = 1001
      Permute_Decomp3(1,1) = 1
      GlobalStart4(1,1) = 0
      GlobalStop5(1,1) = 1001

      VarName6= 't'
      ppp__str= 'DOPO'
      call PPP_COMPARE_VAR(decomp_ij(decomp_ij__NestLevel),t,NNT_REAL,
     +   GlobalUpper2(:,1),Permute_Decomp3(:,1),GlobalStart4(:,1),
     +   GlobalStop5(:,1),GlobalLower1(:,1),1,VarName6,ppp__str,
     +   ppp__status)
      call NNT_CHKSTAT('./laplaceSMS_sms.f:29.0',' ', ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
      endif    ! end of SMS_DEBUGGING_ON()
10       continue
C!sms$EXCHANGE(t)
      GlobalLower1(:,1:1) = 1
      GlobalUpper2(:,1:1) = 1
```

```
      Permute_Decomp3(:,1:1) = 0
      HaloLower7(:,1:1) = 0
      HaloUpper8(:,1:1) = 0
      GlobalStart4(:,1:1) = 1
      GlobalStop5(:,1:1) = 1

!-SMS          storing information about variable: t

      GlobalLower1(1,1) = 0
      GlobalUpper2(1,1) = 1001
      Permute_Decomp3(1,1) = 1
      HaloLower7(1,1) = decomp_ij__HaloSize(1,decomp_ij__NestLevel)
      HaloUpper8(1,1) = decomp_ij__HaloSize(1,decomp_ij__NestLevel)
      GlobalStart4(1,1) = 0
      GlobalStop5(1,1) = 1001
      DataType9(1) = NNT_REAL
      DecompType10(1) = decomp_ij(decomp_ij__NestLevel)
      call PPP_EXCHANGE_1(ExchangeTag11,GlobalLower1,GlobalUpper2,
     +   GlobalStart4,GlobalStop5,Permute_Decomp3,HaloLower7,HaloUpper8,
     +   DecompType10,DataType9,ppp__status,t)
      call NNT_CHKSTAT('./laplaceSMS_sms.f:31.0',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
C-SMS       do i=1,n
```

```
      do i = decomp_ij__S1(1,0,decomp_ij__NestLevel), decomp_ij__E1(n,
     +    0,decomp_ij__NestLevel)
        tnew(i)=(t(i-1)+t(i+1))/2.
      end do
      difmax=0.0
C-SMS      do i=1,n
      do i = decomp_ij__S1(1,0,decomp_ij__NestLevel), decomp_ij__E1(n,
     +    0,decomp_ij__NestLevel)
        dif= abs(tnew(i)- t(i))
      if(dif.gt.difmax) difmax=dif
      t(i)=tnew(i)
      end do
CCSMS$REDUCE(difmax, MAX)
      GlobalSizeR1(1:1) = 1

!-SMS          storing information about variable: difmax

 DataTypeR2(1) = NNT_REAL
      call ppp_reduce_1(GlobalSizeR1,DataTypeR2,NNT_max,ppp__status,
     +   difmax)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:41.1',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
      if(difmax.gt.tol) goto 10
```

```
C!sms$compare_var(t,'DOPO DEFPM u2')

      if (SMS_DEBUGGING_ON()) then
      GlobalLower1(:,1:1) = 1
      GlobalUpper2(:,1:1) = 1
      Permute_Decomp3(:,1:1) = 0
      GlobalStart4(:,1:1) = 1
      GlobalStop5(:,1:1) = 1

!-SMS          storing information about variable: t

      GlobalLower1(1,1) = 0
      GlobalUpper2(1,1) = 1001
      Permute_Decomp3(1,1) = 1
      GlobalStart4(1,1) = 0
      GlobalStop5(1,1) = 1001

      VarName6= 't'
      ppp__str= 'DOPO DEFPM u2'
      call PPP_COMPARE_VAR(decomp_ij(decomp_ij__NestLevel),t,NNT_REAL,
     +   GlobalUpper2(:,1),Permute_Decomp3(:,1),GlobalStart4(:,1),
     +   GlobalStop5(:,1),GlobalLower1(:,1),1,VarName6,ppp__str,
     +   ppp__status)
```

```
      call NNT_CHKSTAT('./laplaceSMS_sms.f:43.0',' ', ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
      endif    ! end of SMS_DEBUGGING_ON()
C!SMS$PARALLEL END
C!SMS$SERIAL BEGIN
      allocate(t__g(0:1001), STAT=ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:45.0',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
      PPP__GlobalUpper = 1
      PPP__GlobalLower = 1
      PPP__Permute = 0
      PPP__HaloLower = 0
      PPP__HaloUpper = 0
      PPP__GlobalLower(1,1) = 0
      PPP__GlobalUpper(1,1) = 1001
      PPP__Permute(1,1) = 1
      PPP__HaloLower(1,1) = decomp_ij__HaloSize(1,decomp_ij__NestLevel)
      PPP__HaloUpper(1,1) = decomp_ij__HaloSize(1,decomp_ij__NestLevel)
      PPP__GlobalStart = PPP__GlobalLower
      PPP__GlobalStop = PPP__GlobalUpper
      PPP__DataType(1) = NNT_REAL
      PPP__DecompType(1) = decomp_ij(decomp_ij__NestLevel)
      call SMS_GATHER(PPP__GlobalLower,PPP__GlobalUpper,
```

```
     +   PPP__GlobalStart,PPP__GlobalStop, PPP__Permute,PPP__DecompType,
     +   PPP__DataType, .FALSE.,t,t__g,ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:45.1',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)

if (IAM_ROOT()) then
      do i=1,n
C-SMS         write(*,*) t(i)
      write (*, *) t__g(i)
      end do
      endif
 PPP__GlobalSize = 1
      call PPP_BCAST(i,NNT_INTEGER,PPP__GlobalSize,
     +   PPP_MAX_DECOMPOSED_DIMS,ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:47.0',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
      PPP__GlobalUpper = 1
      PPP__GlobalLower = 1
      PPP__Permute = 0
      PPP__HaloLower = 0
      PPP__HaloUpper = 0
      PPP__GlobalLower(1,1) = 0
      PPP__GlobalUpper(1,1) = 1001
```

```
      PPP__Permute(1,1) = 1
      PPP__HaloLower(1,1) = decomp_ij__HaloSize(1,decomp_ij__NestLevel)
      PPP__HaloUpper(1,1) = decomp_ij__HaloSize(1,decomp_ij__NestLevel)
      PPP__GlobalStart = PPP__GlobalLower
      PPP__GlobalStop = PPP__GlobalUpper
      PPP__DataType(1) = NNT_REAL
      PPP__DecompType(1) = decomp_ij(decomp_ij__NestLevel)
      call SMS_SCATTER(PPP__CommunicationTag0,PPP__GlobalLower,
     +   PPP__GlobalUpper,PPP__GlobalStart,PPP__GlobalStop,PPP__Permute,
     +   PPP__HaloLower,PPP__HaloUpper,PPP__DecompType,PPP__DataType,
     +   t__g,t,ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:48.0',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
      deallocate(t__g,STAT=ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:48.1',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)

PPP__GlobalSize = 1
      call PPP_BCAST(t1,NNT_REAL,PPP__GlobalSize,
     +   PPP_MAX_DECOMPOSED_DIMS,ppp__status)
      CALL NNT_CHKSTAT('./laplaceSMS_sms.f:50.0',' ',ppp__status,
     +   NNT_ABORT_ON_ERROR, ppp__status)
C!SMS$SERIAL END
```

```
call nnt_stop('./laplaceSMS_sms.f:53',0,PPP_EXIT)
end
```

Miglioramento delle Prestazioni

- Aggregazione delle comunicazioni.
- Ottimizzare gli scambi.
- Evitare le comunicazioni facendo calcoli ridondanti (CSMS$HALO_COMP).
- Ottimizzare CSMS$SERIAL.

Aggregazione delle comunicazioni

- Riduce il numero dei messaggi
  - messaggi meno numerosi=minore tempo di start-up (latenza).
- Migliori prestazioni su macchine ad alta latenza.
- Si applica a CSMS$ECHANGE e a CSMS$REDUCE

Ottimizzazione degli scambi

Riduce gli scambi di dati.

- Fare scambi parziali nelle regioni di halo.

```
CSMS$CREATE_DECOMP(dh,<IM>,<2>)
 real U(IM,JM)
CSMS$EXCHANGE(u<1>,<1>)
```

- Limitare lo scambio a regioni selezionate all'interno della zona di halo.

```
CSMS$CREATE_DECOMP(dh,<IM>,<2>)
CSMS$EXCHANGE(u(:,2:5)<1,0>)
 do j=2,5
do i=1,IM
v(i,j)= u(i-1,j)-u(i,j)
```

Evitare gli scambi facendo...

CSMS$HALO_COMP(<dim1_inf,dim1_sup>, <dim2_inf,dim2_sup>)

dim1(2)_inf è il numero di punti della parte inferiore della regione di halo della prima(seconda) dimensione decomposta per il quale verrà effettuato il calcolo.

dim1(2)_sup è il numero di punti della parte superiore della regione di halo della prima(seconda) dimensione decomposta per il quale verrà effettuato il calcolo.

In alcuni casi può ridurre la latenza e la quantità di dati comunicati.

CSMS$HALO_COMP:esempio

```
CSMS$PARALLEL(dh,<i>) BEGIN
CSMS$EXCHANGE(a<1,1>)
do i=3,10
y(i)= a(i)-a(i+1)-a(i-1)
z(i)= a(i)+a(i+1)-a(i-1)
end do
CSMS$EXCHANGE(y<1,1>,z<1,1>)
do i=3,10
x(i)=y(i)*z(i)+y(i+1)*z(i-1)+y(i-1)*z(i+1)
end do
CSMS$PARALLEL END
```

CSMS$HALO_COMP:esempio

```
CSMS$PARALLEL(dh,<i>) BEGIN
CSMS$EXCHANGE(a<2,2>)
CSMS$HALO_COMP(<1,1>) BEGIN
do i=3,10
y(i)= a(i)-a(i+1)-a(i-1)
z(i)= a(i)+a(i+1)-a(i-1)
end do
CSMS$HALO_COMP END
! CSMS$EXCHANGE(y<1,1>,z<1,1>)
do i=3,10
x(i)=y(i)*z(i)+y(i+1)*z(i-1)+y(i-1)*z(i+1)
end do
CSMS$PARALLEL END
```

Ottimizzare CSMS$SERIAL

Se x ed y sono variabili distribuite:
CSMS$SERIAL(<x,in>,<y,out>: default=ignore) BEGIN
do i=1,IM
y(i)=x(i)
end do
CSMS$SERIAL END
In questa situazione:

- x viene "gatherato"
- y viene "scatterato".

Se uso CSMS$SERIAL BEGIN:

- x,y vengono "gatherate"
- x,y vengono "scatterate".
- i "broadcastato".

MPI vs SMS

```
program laplace
implicit none
include 'mpif.h'

real t(0:1001),tnew(1000)

integer left,right,ierr,nproc
integer i,n,my_id,low,high,master
integer status(MPI_STATUS_SIZE)
real dif,difmax,gdmax,tol
call MPI_INIT(ierr)
call MPI_COMM_RANK &
(MPI_COMM_WORLD,my_id,ierr)
call MPI_COMM_SIZE &
(MPI_COMM_WORLD,nproc,ierr)
master=0
if(my_id.eq.master) then
print*, 'valori di n e di tol',n,tol
read*,n,tol
endif
call MPI_BCAST &
(n,1,MPI_INTEGER,MASTER,&
MPI_COMM_WORLD,ierr)
call MPI_BCAST &
(tol,1,MPI_REAL,MASTER,&
MPI_COMM_WORLD,ierr)
```

```
laplaceSMS
implicit none
!SMS$DECLARE_DECOMP(DECOMP_IJ, <1001+3>:<0>)
!SMS$DISTRIBUTE(DECOMP_IJ, 1) BEGIN
real t(0:1001),tnew(1000)
!SMS$DISTRIBUTE END

integer i, n

real tol,dif,difmax
!SMS$CREATE_DECOMP (DECOMP_IJ,<1001+1>, <1>)




!SMS$SERIAL BEGIN
open(10,file='input.d',form='formatted')
read(10,*) n,tol
print*, 'valori di n e di tol',n,tol
!SMS$SERIAL END
```

MPI vs SMS

```
low=my_id*n/nproc+1
high=low+n/nproc-1
if(my_id.eq.0) then
left=MPI_PROC_NULL
else
left=my_id-1
endif
if(my_id.eq.nproc-1) then
right=MPI_PROC_NULL
else
right=my_id+1
endif
do i=low,high
t(i)=0.0
end do

if(my_id.eq.nproc-1) t(n+1)=100.0
```

```
!SMS$PARALLEL(DECOMP_IJ, <i>) BEGIN

do i=0,n
t(i)=0.0
end do
!SMS$GLOBAL_INDEX(1) BEGIN
t(n+1)=100.0
!SMS$GLOBAL_INDEX END
```

MPI vs SMS

```
10 continue
call MPI_SENDRECV(t(low),1,MPI_REAL, &
left,1,t(high+1),1,MPI_REAL,&
right,1,MPI_COMM_WORLD,status,ierr)
call MPI_SENDRECV(t(high),1,MPI_REAL,&
right,2,t(low-1),1,MPI_REAL,&
left,2,MPI_COMM _WORLD,status,ierr)
do i=low,high
tnew(i)=(t(i-1)+t(i+1))/2.
end do
difmax=0.0
do i=low,high
dif= abs(tnew(i)- t(i))
if(dif.gt.difmax) difmax=dif
t(i)=tnew(i)
end do
call MPI_ALLREDUCE(difmax,gdmax,1,MPI_REAL,&
MPI_MAX,MPI_COMM_WORLD,ierr)
if(gdmax.gt.tol) goto 10
```

```
10 continue
!sms$EXCHANGE(t)

do i=1,n
tnew(i)=(t(i-1)+t(i+1))/2.
end do
difmax=0.0
do i=1,n
dif= abs(tnew(i)- t(i))
if(dif.gt.difmax) difmax=dif
t(i)=tnew(i)
end do
CSMS$REDUCE(difmax, MAX)

if(difmax.gt.tol) goto 10
```

MPI vs SMS

```
call MPI_GATHER(t(low),n/nproc,MPI_REAL,&
t(1),n/nproc,MPI_REAL,master,&
MPI_COMM_WORLD,ierr)
if(my_id.eq.master) then
do i=1,n
write(*,*) t(i)
end do
endif
call MPI_FINALIZE(ierr)
end
```

```
!SMS$PARALLEL END
!SMS$SERIAL BEGIN


do i=1,n
write(*,*) t(i)
end do

!SMS$SERIAL END
end
```

SMS debugging

Inserire delle direttive SMS all'interno del codice per controllare che i valori degli array siano corretti.

- Eseguire contemporaneamente
  - il codice seriale con il codice parallelo
  - due copie del codice parallelo (con diverso numero di CPU)

  confrontare i risultati durante l'esecuzione: CSMS$COMPARE_VAR
  - è richiesta la riduzione *bitwise* esatta
  - errori di *round off* possono mascherare i veri problemi
- Verificare le regioni di Halo:CSMS$CHECK_HALO

SMS debugging CSMS$COMPARE_VAR

CSMS$COMPARE_VAR(x,'sono qui')

- Verifica che la variabile specificata abbia lo stesso valore nei due *run* separati.
- Controlla i punti interni della variabile non le regioni di halo.
- È possibile controllare solo variabili semplici (real,integer,...).
- Se la variabile non ha lo stesso valore l'esecuzione termina con un errore.

```
compare_var : x sono qui
Values for first, second run : 25.0 12.0
Incorrect at indices 24 3 3
```

SMS debugging CSMS$COMPARE_VAR

Due modi di esecuzione

- un solo programma
  - smsRun -np 1 par_code -np 4 par_code -cv
- due programmi
  - il programma SMS tradotto e compilato normalmente
  - il programma seriale tradotto sono nelle direttive CSMS$COMPARE_VAR
    ppp -CompareOnly progr_ser.f

  - smsRun -np 1 ser_code -np 4 par_code -cv

SMS debugging CSMS$CHECK_HALO

CSMS$CHECK_HALO(x,'sono qui')

- La sua sintassi simile a CSMS$EXHANGE
- Verifica che la regione di halo della variabile specificata è stata aggiornata.
- Se non è aggiornata termina con un messaggio di errore

  Halo check failed fo variable x, sono qui.

- Degrada le performance.
- È attiva quando la varibile ambiente CSMS$CHECK_HALO ON

Shallow water model

$$\begin{cases} \frac{\partial u}{\partial t} = (\xi + f)v - \frac{\partial B}{\partial x} + \frac{\tau_x}{\rho_0 h} + A\Delta u - ru \\ \frac{\partial v}{\partial t} = -(\xi + f)v - \frac{\partial B}{\partial y} + A\Delta v - rv \\ \frac{\partial h}{\partial t} = -\frac{\partial (hu)}{\partial x} - \frac{\partial (hv)}{\partial y} \end{cases}$$

- u,v componenti della della vorticità orizzontale
- h elevazione della superficie
- $B = \frac{1}{2}(u^2 + v^2) + gh$ funzione di Bernoulli
- $\xi = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$ vorticità relativa
- f parametro di Coriolis
- A laplaciano del coefficiente di diffusività
- r coefficiente di frizione sul fondo
- τ_x wind stress zonale.

Il modello è discretizzato usando uno schema centrato alle differenze finite del secondo ordine nello spazio e nel tempo su una griglia Arakawa C (staggered).

Shallow water model

```
Program Swc
use commo
implicit none
integer it,itdeb,itfin,i,j,nxp2,nyp2
real time0
open(12,file='input.d',status='old')
read(12,*)
read(12,*)nx,ny
read(12,*)
read(12,*) itdeb,itfin,isauv
read(12,*)
read(12,*) dt
nxp1 = nx + 1
nyp1 = ny + 1
nxp2 = nx + 2
nyp2 = ny + 2
!SMS$CREATE_DECOMP(dec, <nxp2, nyp2>,<1,1>)
allocate(up(0:nxp1,0:nyp1),u(0:nxp1,0:nyp1),um(0:nxp1,0:nyp1))
allocate(uc(0:nxp1,0:nyp1))
allocate(vp(0:nxp1,0:nyp1),v(0:nxp1,0:nyp1),vm(0:nxp1,0:nyp1))
allocate(vc(0:nxp1,0:nyp1))
allocate(hp(0:nxp1,0:nyp1),h(0:nxp1,0:nyp1),hm(0:nxp1,0:nyp1))
allocate(hc(0:nxp1,0:nyp1),vr(0:nxp1,0:nyp1))
```

```
allocate(b(0:nx,0:ny),lapu(0:nx,0:ny),lapv(0:nx,0:ny))
allocate(fu(1:ny),fv(2:ny),taux(1:ny))
dtt = 2.*dt
! Size of the computationnal domain
L = 2e6
! Space step
ds = L / real(nx)
ds2 = ds*ds
! Parameters for the model
A = 300. ! viscosity (Laplacian diffusion)
r = 0.9e-7 ! bottom friction coefficient
rho0 = 1000. ! sea water density
g = 0.02 ! reduced gravity
beta = 2.e-11 ! gradient of coriolis force
f0 = 7.e-5
tau0 = 0.05 ! wind amplitude
h0 = 500. ! water height at rest
gamma = 0.02 ! asselin time filter coefficient
      call Initgrille()
! Time integration of the grid hierarchy
Do it = itdeb,itfin
     call Step()
End Do
```

```
End


Subroutine Step()
use commo
implicit none
Real aux1,aux2,AA,BB,CC,DD,dtl
Integer i,j,jp1,jm1,ip1,im1,itnew,irhox,irhoy
If (mod(nbstep,1000).EQ.0) Then
print*,'step ',nbstep,ds,dt
End If
itnew = nbstep + 1
!SMS$PARALLEL(dec, <i>,<j>) BEGIN
!SMS$COMPARE_VAR(u,v,' prima')
 If (itnew .EQ. 1) then
Do j = 0,nyp1
  Do i = 1,nxp1
    uc(i,j) = u(i,j)
  End Do
End Do
Do j = 1,nyp1
  Do i = 0,nxp1
    vc(i,j) = v(i,j)
```

```
  End Do
End Do
Do j = 0,nyp1
  Do i = 0,nxp1
    hc(i,j) = h(i,j)
  End Do
End Do
Else
! Asselin Filter
Do j = 0,nyp1
  Do i = 1,nxp1
    uc(i,j) = um(i,j) + gamma*(uc(i,j) - &
    2.*um(i,j) + u(i,j))
  End Do
End Do
Do j = 1,nyp1
  Do i = 0,nxp1
    vc(i,j) = vm(i,j) + gamma*(vc(i,j) - &
    2.*vm(i,j) + v(i,j))
  End Do
End Do
Do j = 0,nyp1
  Do i = 0,nxp1
```

```
      hc(i,j) = hm(i,j) + gamma*(hc(i,j) - &
      2.*hm(i,j) + h(i,j))
    End Do
End Do
Endif
!SMS$CHECK_HALO(u,v,h)
!SMS$EXCHANGE(u<0:1,1:0>,v<1:0,0:1>)
 ! Calculation of the Bernouilli potential
Do j = 1,ny
    Do i = 1,nx
      ip1 = i + 1
      aux1 = 0.5 * (u(i,j)+u(ip1,j))
      aux2 = 0.5 * (v(i,j)+v(i,j+1))
      b(i,j) = 0.5 * (aux1*aux1+aux2*aux2)+g*h(i,j)
    End Do
End Do
! calculation of the relative vorticity
!SMS$COMPARE_VAR(u,v,' prima calcolo vort rel')
 Do j = 1,nyp1
    Do i = 1,nxp1
      vr(i,j) = ((v(i,j)-v(i-1,j)-u(i,j)+u(i,j-1))/ds)
    End Do
End Do
```

```
!SMS$COMPARE_VAR(vr,' dopo calcolo vort rel')
!SMS$EXCHANGE(vr<0:1,0:1>,uc,vc,b<1:0,1:0>,h)
 ! Calculation of u and v Laplacians
Do j = 1,ny
  Do i = 2,nx
    ip1 = i + 1
    im1 = i - 1
    lapu(i,j) = (uc(im1,j)+uc(ip1,j)+uc(i,j+1)+ &
    uc(i,j-1)-4.*uc(i,j)) / ds2
  End Do
End Do
Do j = 2,ny
  Do i = 1,nx
    ip1 = i + 1
    im1 = i - 1
    lapv(i,j) = (vc(im1,j)+ vc(ip1,j)+vc(i,j+1) &
    +vc(i,j-1)-4.*vc(i,j)) / ds2
  End Do
End Do
!SMS$GLOBAL_INDEX(1) BEGIN
 Do j=1,ny
    up(1,j)=0.
```

```
      up(nxp1,j)=0.
End Do
!SMS$GLOBAL_INDEX END
!SMS$GLOBAL_INDEX(2) BEGIN
 Do i=1,nx
      vp(i,1)=0.
      vp(i,nyp1)=0.
End Do
Do i=1,nyp1
      up(i,0)=-up(i,1)
      up(i,nyp1)=-up(i,ny)
End Do
!SMS$GLOBAL_INDEX END
!SMS$GLOBAL_INDEX(1) BEGIN
 Do j=1,nxp1
      vp(0,j)=-vp(1,j)
      vp(nxp1,j)=-vp(nx,j)
End Do
!SMS$GLOBAL_INDEX END
 ! Time step
If (itnew .EQ. 1) Then
dtl = dt
Else
```

```
dtl = dtt
End If
! Pronostic equation for u
Do j = 1,ny
jp1 = j+1
  Do i = 2,nx
    ip1 = i + 1
    im1 = i - 1
    AA = fu(j) + 0.5*(vr(i,j)+vr(i,jp1))
    BB = 0.25 * (v(im1,j)+v(i,j)+      v(im1,jp1)+v(i,jp1))
    CC = (b(i,j)-b(im1,j)) / ds
    DD = 0.5 * (h(i,j)+h(im1,j))
    up(i,j) = uc(i,j) + dtl*(AA*BB-CC+taux(j)/(rho0*DD) &
    + A*lapu(i,j)-r*u(i,j))
  End Do
End Do
! Pronostic equation for v
Do j = 2,ny
jm1 = j - 1
  Do i = 1,nx
    ip1 = i + 1
    im1 = i - 1
    AA = fv(j) + 0.5*(vr(i,j)+vr(ip1,j))
```

```
      BB = 0.25 * (u(i,jm1)+u(i,j)+      u(ip1,jm1)+u(ip1,j))
      CC = (b(i,j)-b(i,jm1)) / ds
      vp(i,j) = vc(i,j) + dtl*(-AA*BB-CC+A*lapv(i,j)-r*v(i,j))
    End Do
  End Do
  ! Pronostic equation for h
  Do j = 1,ny
  jp1 = j + 1
  jm1 = j - 1
    Do i = 1,nx
      ip1 = i + 1
      im1 = i - 1
      AA = 0.5 * ((h(ip1,j)+      h(i,j))*u(ip1,j) -&
      (h(im1,j)+h(i,j))*u(i,j)) / ds
      BB = 0.5 * ((h(i,jp1)+h(i,j))*v(i,jp1) -&
      (h(i,j)+h(i,jm1))*v(i,j))/ds
      hp(i,j) = hc(i,j) + dtl*(-AA-BB)
    End Do
  End Do
  ! Velocity correction : Ensures mass flux equality at interfaces
  ! Time update of u (um = u(n-1), u = u(n))
  Do j = 0,nyp1
    Do i = 1,nxp1
```

```
      um(i,j) = u(i,j)
      u(i,j) = up(i,j)
    End Do
  End Do
  ! Time update of v (vm = v(n-1), v = v(n))
  Do j = 1,nyp1
    Do i = 0,nxp1
      vm(i,j) = v(i,j)
      v(i,j) = vp(i,j)
    End Do
  End Do
  ! Time update of h (hm = h(n-1), h = h(n))
  Do j = 0,nyp1
    Do i = 0,nxp1
      hm(i,j) = h(i,j)
      h(i,j) = hp(i,j)
    End Do
  End Do
  !SMS$SERIAL BEGIN
   If (mod(nbstep,isauv).eq.0) Then
  write(istok,*)1.
  End If
  write(11,*) itnew*dt,h(nx/2,ny/2)
```

```
!SMS$SERIAL END
 nbstep = nbstep + 1
!SMS$PARALLEL END
 Return
End Subroutine Step


Subroutine Initgrille
use commo
implicit none
Integer i,j
Character*80 line1,filout
Real yu,yv
Integer i0
Real pi,y0
!SMS$PARALLEL(dec, <i>,<j>) BEGIN
 nbstep = 0
y0=0.
pi = 4.*atan(1.)
Do j = 1,ny
     yu = (float(j)-0.5) * ds + y0
     fu(j) = f0 + beta * yu
     taux(j) = tau0 * sin((2*pi/L)*(yu-L/4.))
```

```
End Do
Do j = 2,ny
    yv = float(j-1) * ds + y0
    fv(j) = f0 + beta * yv
End Do
Do j = 0,nyp1
  Do i = 0,nxp1
    um(i,j) = 0.
    u(i,j) = 0.
    uc(i,j) = 0.
    up(i,j) = 0.
  End Do
End Do
!SMS$COMPARE_VAR(u,v,' init1')
 Do j = 1,nyp1
  Do i = 0,nxp1
    vm(i,j) = 0.
    v(i,j) = 0.
    vc(i,j) = 0.
    vp(i,j) = 0.
  End Do
End Do
Do j = 0,nyp1
```

```
  Do i = 0,nxp1
    h(i,j) = h0
    hm(i,j) = h0
    hc(i,j) = h0
    hp(i,j) = h0
  End Do
End Do
!SMS$COMPARE_VAR(u,v,' init')
 ! open files for output
istok = 99
filout='sortie.bimg'
open(istok,file=filout)
line1='grid'
write(istok,*)line1
write(istok,*)line1
write(istok,*)line1
write(istok,*)line1
write(istok,*)nx,ny,1,100,1,1
write(istok,*)0.,0.,ds,ds,0.
write(istok,*)500.,500.
!SMS$PARALLEL END
 Return
```

```
End Subroutine Initgrille


MODULE COMMO
real dt,ds
integer istok,nbstep


integer nx,ny
integer nxp1,nyp1



!SMS$DECLARE_DECOMP(dec:<0,0>)
!SMS$DISTRIBUTE(dec, 1, 2) BEGIN
 real,allocatable :: up(:,:),u(:,:),um(:,:),uc(:,:)
real,allocatable :: vp(:,:),v(:,:),vm(:,:),vc(:,:)
real,allocatable :: hp(:,:),h(:,:),hm(:,:),hc(:,:)
real,allocatable :: vr(:,:)
real,allocatable :: umask(:,:),vmask(:,:),hmask(:,:)
```

```
real,allocatable :: fmask(:,:),(:,:), lapu(:,:),lapv(:,:)
!SMS$DISTRIBUTE END
 real,allocatable :: fu(:),fv(:),taux(:)
real dtt,ds2
real L,A,r,rho0,g,beta,f0,tau0,h0,gamma
integer isauv,noslip
END MODULE COMMO
```

SW: Compilazione ed esecuzione

```
/usr/local++/SMS/2.9/bin/ppp --r8 --FreeFormat commo.f
/usr/local++/SMS/2.9/bin/ppp --r8 --Fmodule=commo --FreeFormat globale.f
mpxlf90_r  -O3 -qstrict -qarch=auto -q64 -qrealsize=8 -I/usr/local++/SMS/2.9/inc
lude -c commo_sms.f
mpxlf90_r -O3 -qstrict -qarch=auto -q64 -qrealsize=8 -I/usr/local++/SMS/2.9/incl
ude -c globale_sms.f commo_sms.o
mpxlf90_r -qfixed -o sw.exe -O3 -qstrict -qarch=auto -qrealsize=8 -q64 globale
_sms.o -L/usr/local++/SMS/2.9/lib -lsms
```

```
#!/bin/csh
setenv SMS_SERVER_MODE serverless
setenv SMS_BITWISE EXACT
smsRun ($numero_processori) sw.exe
```

SW:output

1 processore

```
SMS:: Program started: 2007:01:26::17:21:14 (serverless) (SMS Version: 2.9)
SMS:  BITWISE EXACT reductions will be used when requested.

SMS:  Using default process layout ( 1  x  1 ) for decomposition dec
step  0 400.000000000000000 25.0000000000000000
  .....
```

8 processori

```
SMS:: Program started: 2007:01:26::17:25:03 (serverless) (SMS Version: 2.9)
SMS:  BITWISE EXACT reductions will be used when requested.

SMS:  Using default process layout ( 2  x  4 ) for decomposition dec
step  0 400.000000000000000 25.0000000000000000
....
```

Tempo di esecuzione del singolo step con un processore 4714 millisecondi.

N.ro proc.	1	4	8	12	16	20	24	28	32
Speed-up	1	4	7.7	11.3	14.9	18.6	22.9	27.2	30

SW: diverso layout

```
setenv SMS_SERVER_MODE serverless
setenv SMS_BITWISE EXACT
smsRun -cf my_config sw.exe
.....
[my_config]
&decomp
decomp1_name = 'dec',
decomp1_nps = 7 4  /
```

OUTPUT

```
SMS:: Program started: 2007:01:30::15:23:43 (serverless) (SMS Version: 2.9)
SMS:  BITWISE EXACT reductions will be used when requested.

SMS: For processor layout of decomposition dec, using config file : my_config
step  0 400.000000000000000 25.0000000000000000
....
```

Parallelizzazione del codice EPOM

- Versione modificata del Princeton Ocean Model (POM) sviluppata da ENEA Casaccia e Caspur per studiare la circolazione nel Mediterraneo.
- Codice Fortran 90 risolve le equazioni di Navier–Stokes nelle variabili primitive attraverso il metodo delle differenze finite.
- La griglia è $408 \times 166 \times 31$ e si vogliono simulare i cosiddetti *run* climatici ($\sim$ 100 anni di simulazione).
- Il codice seriale impiega 14.9 secondi su un singolo processore Pwr4 (1300 Mhz di clock) per singolo *time–step* ($\sim \frac{1}{100}$ di anno)
- Le 20000 linee del codice EPOM sono parallelizzate mediante 700 direttive SMS (3.5 % totale).

N.ro proc.	1	4	8	12	16	20	24	28	32
Speed-up	1	4.1	7.7	10.8	13.3	15.1	17.5	19.5	21.2

Parallelizzazione del codice 3d

- Codice sviluppato presso l'Università di Praga ed usato dall'INGV per lo studio di effetti sismici nella città di Roma.
- Codice Fortran 90 risolve le equazioni dell'elasticità in dominio 3D attraverso il metodo delle differenze finite.
- La griglia è $563 \times 852 \times 94$ e si simula per circa 5000 passi temporali.
- Il codice seriale impiega 25 secondi su un singolo processore Pwr5 (1.9 Ghz di clock) per singolo *time–step*
- Le circa 12000 linee del codice sono parallelizzate mediante 500 direttive SMS (circa 4% del totale).

N.ro proc.	1	4	8	12	16	20	24	28	32
Speed-up	1	4.4	8.3	11.5	14.5	17	20.6	22.2	24

Riferimenti Web:

- **SMS** http://www-ad.fsl.noaa.gov/ac/sms.html