

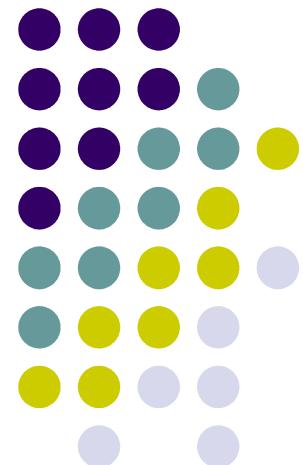
Università di Roma “La Sapienza”

Dipartimento di Matematica

15 – 31 gennaio 2007

OpenMP

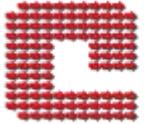
Parte seconda



Fabio Bonaccorso
f.bonaccorso@caspur.it



Sommario



- Reduction
- Race condition
- Subs & thread
- Dynamic vs static binding
- Direttive “orfane”
- Threadprivate
- Esempi:
 - calcolo su matrici
 - FFT
 - Differenze finite



Calcolare statistiche di vettori

- Dato un vettore di reali A, ne vogliamo calcolare la media o il massimo:
- In parallelo sara':

```
do I=1,N  
    a(i) = i  
enddo
```

```
sum=0  
maxA=a(1)
```

```
do I=1,N  
    sum=sum+a(i)  
    maxA=max(maxA,a(i))  
enddo  
avg = sum / N
```

```
write(*,*) "avg=",avg, "vs", (N+1)/2.0  
write(*,*) "max=",maxA, "vs",N
```

```
do i=1,N  
    a(i) = i  
enddo  
sum=0  
maxA=a(1)
```

```
!$omp parallel  
 !$omp do  
    do I=1,N  
 !$omp critical  
     sum=sum+a(i)  
     maxA=max(maxA,a(i))  
 !$omp end critical  
    enddo  
 !$omp end do  
 !$omp end parallel
```

```
avg = sum / N
```

```
write(*,*) "avg=",avg, "vs", (N+1)/2.0  
write(*,*) "max=",maxA, "vs",N
```



Reduction

- In OpenMP per velocizzare un'operazione applicata su un vettore si usa la clausola REDUCTION nella direttiva DO/for
- In Fortran:
 - !\$OMP DO REDUCTION (op:var list)
con op=+,-,* , .and., .or., .eqv., .neqv.
e anche op= max,min, iand, ior, ieor
- In C/C++
 - #pragma omp for reduction(op:var list)
con op=+,-,* , &,|, &&,||



Esempio

- Usando la clausola reduction

```
do i=1,N  
  a(i) = i  
enddo  
sum=0  
maxA=a(1)  
 !$omp parallel  
 !$omp do reduction(+:sum)  
 !$omp& reduction(max:maxA)  
   do I=1,N  
     sum=sum+a(i)  
     maxA=max(maxA,a(i))  
   enddo  
 !$omp end do  
 !$omp end parallel
```

avg = sum / N

write(*,*) "avg=",avg, "vs", (N+1)/2.0

write(*,*) "max=",maxA, "vs",N



Output

- Con 1 thread:

```
bonaccor/OMP-EX> setenv OMP_NUM_THREADS 1 ; ./ReducMedie
```

avg= 50000000.99999999	vs	50000001
------------------------	----	----------

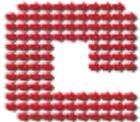
max= 10000001.0000000	vs	10000001
-----------------------	----	----------

- Con 2 threads:

```
bonaccor/OMP-EX> setenv OMP_NUM_THREADS 2 ; ./ReducMedie
```

avg= 50000000.99999999	vs	50000001
------------------------	----	----------

max= 10000001.0000000	vs	10000001
-----------------------	----	----------



Race condition

- Si genera una race condition (tra i thread) quando l'esito di una scrittura in memoria dipende dall'ordine con cui i diversi thread eseguono il codice
- Si manifesta con la produzione di risultati a diversi in run diversi
- Si corregge proteggendo l'aggiornamento con uno dei meccanismi di sincronizzazione (**BARRIER, CRITICAL, ATOMIC**)



Thread e subroutine

- Ogni thread che incontra una chiamata ad una subroutine la esegue (indipendentemente)
- Ogni variabile definita nella subroutine sara' privata al thread
- Le variabili argomenti della subroutine mantengono il loro stato originario



Esempio di subroutine

PROGRAM TEST

!\$OMP PARALLEL

call whoami

!\$OMP END PARALLEL

END

subroutine whoami

include "omp_lib.h"

integer iam

iam=omp_get_thread_num()

print *, "Hello from ", iam

return

end



Output

- Con 1 thread:

```
bonaccor/OMP-EX> setenv OMP_NUM_THREADS 1 ; ./Subs
```

```
Hello from      0
```

- Con 2 thread:

```
bonaccor/OMP-EX> setenv OMP_NUM_THREADS 2 ; ./Subs
```

```
Hello from      0
```

```
Hello from      1
```

- Con 5 thread

```
bonaccor/OMP-EX> setenv OMP_NUM_THREADS 5 ; ./Subs
```

```
Warning: OMP_NUM_THREADS or NCPUS (5) greater than available cpus (2)
```

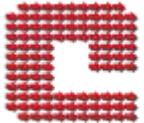
```
Hello from      0
```

```
Hello from      1
```

```
Hello from      2
```

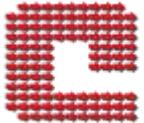
```
Hello from      4
```

```
Hello from      3
```



Subs e thread

- Ogni thread ha eseguito la subroutine indipendentemente
- Se avessero acceduto a variabili condivise senza sincronizzazione =>race conditions
- Cosa fornisce OpenMP?
 - Direttive "orfane"



Binding

- Nel sorgente, la parte di codice della regione parallela (**PARALLEL/END PARALLEL**) e' il binding statico delle direttive OpenMP
- Il binding dinamico e' l'insieme del codice eseguito dal team di thread
 - il binding statico
 - tutte le subroutine chiamate
- Il binding dinamico comprende codice definito in file diversi da quello che apre/chiude la regione parallela



Direttive "orfane"

- Nel codice eseguito nelle subroutine richiamate dai thread, si possono usare le direttive OpenMP:
 - Suddividere il lavoro ([DO/for](#))
 - Sincronizzare un'operazione collettiva
 - Creare altri thread
 - Nesting: ogni thread diventa il master thread di un nuovo team di thread
 - Solo se supportato



Esempio direttive "orfane"

PROGRAM TEST

integer N

paramater (N=1000)

real A(N)

!\$OMP PARALLEL SHARED(A)

 call whoami2(A,N)

!\$OMP END PARALLEL

END

subroutine whoami2(A,N)

implicit none

include "omp_lib.h"

integer i,N,iam

real A(N)

iam = omp_get_thread_num()

!\$omp do

do i=1,N

 A(i) = i + (iam+1)*10000

enddo

!\$omp end do

return

end



Output

- Con 1 thread:

```
bonaccor/OMP-EX> setenv OMP_NUM_THREADS  
1 ; ./Subs2
```

1	10001.00
2	10002.00
...	
60	10060.00

- Con 2 threads:

```
bonaccor/OMP-EX> setenv OMP_NUM_THREADS  
2 ; ./Subs2
```

1	10001.00
2	10002.00
...	
29	10029.00
30	10030.00
31	20031.00
32	20032.00
...	
59	20059.00
60	20060.00



OpenMP e i common del FORTRAN (globals in C/C++)

- I common forniscono un modo per condividere variabili tra chiamante e subroutine (C/C++ -> variabili globali)
- In OpenMP condividerle (non intenzionalmente) potrebbe causare race conditions
- Si rendono privati i common
 - Ogni common FORTRAN puo' contenere piu' variabili
 - privati => non inizializzati (usare clausole opz)
- Quando e' inizializzato?
 - Al primo riferimento



THREADPRIVATE

- La direttiva per privatizzare un intero common (variabile globale in C/C++) e' threadprivate
- Sarà probabilmente una direttiva "orfana"
- In Fortran:
 - `!$OMP PARALLEL THREADPRIVATE(var)`
Il nome del common va racchiuso tra '/'
- In C/C++
 - `#pragma omp threadprivate(var)`



Esempio di THREADPRIVATE

```
subroutine VecSqrt  
parameter (N=1000)  
common/buf/A(N),B(N)  
C$OMP THREADPRIVATE(/buf/)
```

```
do i=1, N  
    A(i) = sqrt(B(i))  
end do
```

```
return
```

```
end
```

- La direttiva compare in una funzione non racchiusa in una regione parallela -> direttiva orfana
- Se/quando thread entra nella routine, lavorera' su una copia privata
- In questo caso non e' inizializzata
 - Si presume che in altre parte ci sara' un'inizializzazione
 - Utile per aree di calcolo temporanee



FIRSTPRIVATE

- E' una clausola che dichiara una variabile privata che verra' inizializzata al valore della variabile originale
- Si usa con PARALLEL,DO,SINGLE
- In Fortran:
 - !\$OMP PARALLEL FIRSTPRIVATE(var)
- In C/C++
 - #pragma omp parallel firstprivate(var)



LASTPRIVATE

- E' una clausola che dichiara una variabile privata. Alla fine del costrutto cui si riferisce, l'ultimo thread copiera' la sua versione privata nella variabile originale
- Si usa con **PARALLEL,DO,SINGLE**
- Si puo' accoppiare a **FIRSTPRIVATE**
- In Fortran:
 - `!$OMP PARALLEL LASTPRIVATE(var)`
- In C/C++
 - `#pragma omp parallel lastprivate(var)`



COPYIN

- L'analogo di FIRSTPRIVATE per le variabili di un common e' copyin
- In Fortran:
 - !\$OMP COPYIN(var)
- In C/C++
 - #pragma omp copyin(var)



Sections

- Quando, in una regione parallela, si vuole assegnare ad ogni thread un'attivita' indipendente si usa la direttiva SECTION
- In Fortran:

```
!$OMP SECTIONS  
  !$OMP SECTION  
    block1  
  !$OMP END SECTION  
  !$OMP SECTION  
    block2  
  !$OMP END SECTION  
 !$OMP END SECTIONS [nowait]
```



Esempio

```
PROGRAM VEC_ADD_SECTIONS
```

```
INTEGER N, I  
PARAMETER (N=1000)  
REAL A(N), B(N), C(N)
```

```
! Some initializations  
DO I = 1, N  
    A(I) = I * 1.0  
    B(I) = A(I)  
ENDDO
```

```
!$OMP PARALLEL SHARED(A,B,C), PRIVATE(I)  
!$OMP SECTIONS  
!$OMP SECTION  
    DO I = 1, N/2  
        C(I) = A(I) + B(I)  
    ENDDO  
!$OMP END SECTION  
  
!$OMP SECTION  
    DO I = 1+N/2, N  
        C(I) = A(I) + B(I)  
    ENDDO  
!$OMP END SECTION  
  
!$OMP END SECTIONS NOWAIT  
!$OMP END PARALLEL  
END
```



Esempi

- Analizziamo alcuni esempi
- Pausa?



Moltiplicazione matrice vettore

- Moltiplicare una matrice A per un vettore B:

```
do i = 1, size
    do j = 1, size
        A(j,i) = j * 1.0
    enddo
    B(i) = i
    C(i) = 0.0
enddo

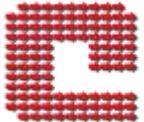
do i = 1, size
    do j = 1, size
        C(i) = C(i) + A(j,i) * B(i)
    enddo
enddo
```

- In parallelo sara':

```
!$omp parallel shared(A,B,C) private(j)
 !$omp do
    do i = 1, size
        do j = 1, size
            A(j,i) = j * 1.0
        enddo
        B(i) = i
        C(i) = 0.0
    enddo
    !$omp end do

    !$omp do
        do i = 1, size
            do j = 1, size
                C(i) = C(i) + A(j,i) * B(i)
            enddo
        enddo
    !$omp end do

    !$omp end parallel
```



Output

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 1 ; time ./MatVec  
1.770u 0.002s 0:02.14 82.7% 0+5045k 0+0io 0pf+0w
```

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 2 ; time ./MatVec  
2.134u 0.368s 0:01.48 168.2% 0+4297k 0+0io 0pf+0w
```

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 4 ; time ./MatVec  
2.350u 0.420s 0:00.99 279.7% 0+4023k 0+0io 0pf+0w
```

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 8 ; time ./MatVec  
4.180u 0.553s 0:01.04 454.8% 0+4505k 0+0io 0pf+0w
```

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 6 ; time ./MatVec  
2.882u 0.454s 0:00.91 365.9% 0+4182k 0+0io 0pf+0w
```



Moltiplicazione di due matrici

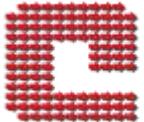
```
do i = 1, size
    do j = 1, size
        A(j,i) = j * 1.0
        B(j,i) = j * 0.25
    enddo
enddo
```

```
do j = 1, size
    do k = 1, size
        do i = 1, size
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
        enddo
    enddo
enddo
```

```
!$omp parallel shared(A,B,C) private(i)
 !$omp do
    do j = 1, size
        do i = 1, size
            A(i,j) = j * 1.0
            B(i,j) = j * 0.25
            C(i,j) = 0.0
        enddo
    enddo
 !$omp end do
```

```
!$omp do private(k)
    do j = 1, size
        do k = 1, size
            do i = 1, size
                C(i,j) = C(i,j) + A(i,k) * B(k,j)
            enddo
        enddo
    enddo
 !$omp end do

 !$omp end parallel
```



Output

- Con 1 thread

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 1 ; time ./MatMat  
Size is 2000 Kb= 488  
18.748u 0.019s 0:18.78 99.8% 0+500k 0+0io 0pf+0w
```

- Con 2 thread

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 2 ; time ./MatMat  
Size is 2000 Kb= 488  
18.867u 0.018s 0:09.46 199.4% 0+486k 0+0io 0pf+0w
```

- Con 4 thread

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 4 ; time ./MatMat  
Size is 2000 Kb= 488  
18.881u 0.028s 0:04.75 397.8% 0+477k 0+0io 0pf+
```

- Con 8 thread

```
<bonaccor@pwr503 ~/private/OMP-EX>setenv OMP_NUM_THREADS 8 ; time ./MatMat  
Size is 2000 Kb= 488  
19.350u 0.048s 0:02.44 794.6% 0+467k 0+0io 0pf+0w
```



Differenze finite

```
c$OMP PARALLEL
c$OMP& default(private)
c$OMP& shared(rho_enuo,unuo,un1,rho_en1,u,rho_e,T,dt,invRe0)
c$OMP& shared(coef3,coefpt,coeft,mu)
c$OMP& shared(fraz1,fraz2,frazp)
c$OMP& shared(fraz1nuo,fraz2nuo,frazpnuo)
c$OMP& shared(fraz1n1,fraz2n1,frazpn1)
c$OMP& shared(iteraz,Damk,Ze,coefCe,coefSc,mu_espo)
c$OMP& reduction(+:chim_rate)
c$OMP& reduction(+:chim_ratef)
c$OMP& reduction(+:chim_rateg)
c$OMP& reduction(min:min_chim1,min_chim2,min_chim3)
c$OMP& reduction(max:max_chim1,max_chim2,max_chim3)
c$OMP DO
    do k = 1+NS,Nz-NS
        do j = 1+NS,Ny-NS
            do i = 1+NS,Nx-NS
                xp = (i-NS-1)*Dx
                yp = (j-NS-1)*Dy
                zp = (k-NS-1)*Dz
                rhoqui = rho_e(1,i,j,k)
                rhoxp1 = rho_e(1,i+1,j,k)
                rhoxm1 = rho_e(1,i-1,j,k)
                rhoyp1 = rho_e(1,i,j+1,k)
                rhoym1 = rho_e(1,i,j-1,k)
                rhozp1 = rho_e(1,i,j,k+1)
                rhozm1 = rho_e(1,i,j,k-1)
                rhoequi = rho_e(2,i,j,k)
                rhoexp1 = rho_e(2,i+1,j,k)
                rhoexm1 = rho_e(2,i-1,j,k)
                rhoeyp1 = rho_e(2,i,j+1,k)
                rhoeym1 = rho_e(2,i,j-1,k)
                rhoezp1 = rho_e(2,i,j,k+1)
                rhoezm1 = rho_e(2,i,j,k-1)
```

```
t11 = 4.0/3.0*( D2x*(u1xp1-u1xm1) )-2.0/3.0*( D2y*(u2yp1-u2ym1) )-2.0/3.0*( D2z*(u3zp1-u3zm1) )
t22 = -2.0/3.0*( D2x*(u1xp1-u1xm1) )+4.0/3.0*( D2y*(u2yp1-u2ym1) )-2.0/3.0*( D2z*(u3zp1-u3zm1) )
t33 = -2.0/3.0*( D2x*(u1xp1-u1xm1) )-2.0/3.0*( D2y*(u2yp1-u2ym1) )+4.0/3.0*( D2z*(u3zp1-u3zm1) )
t12 = ( D2y*(u1yp1-u1ym1) ) + ( D2x*(u2xp1-u2xm1) )
t13 = ( D2z*(u1zp1-u1zm1) ) + ( D2x*(u3xp1-u3xm1) )
t23 = ( D2y*(u3yp1-u3ym1) ) + ( D2z*(u2zp1-u2zm1) )
```

....

....

....

```
ap4 = term3 + u1qui*rap1 + u2qui*rap2 + u3qui*rap3
ap5 = ap3*coef3-ap1-ap2+invRe0*ap4-2*coefCe*chim_ap3
rhoenuo(2,i,j,k)=rhoequi + dt*(ap4+ap5)
enddo
enddo
```

```
c$OMP DO
```



FFT 1

Program Testafft

```
implicit none
integer*4  naux, ndim
integer*4  nptot,np(3)
integer*4  numPre, numMis, numSpagh
include "par_64_1.inc"
parameter (ndim = 3)
parameter (naux = 37754)
parameter (numPre = 1)
parameter (numMis = 200)
complex      b1(inc3y,Nz),b2(inc3y,Nz),b3(inc3y,Nz)
complex      b4(inc3y,Nz),b5(inc3y,Nz),b6(inc3y,Nz)
real         a1(2*inc3y,Nz),a2(2*inc3y,Nz)REAL*8
     dummy, work(naux),scala
REAL*8       rtc, GT1, GT2
real*4        xp, yp, zp
integer       i, j, k,ij,loop
np(1) = Nx
np(2) = Ny
np(3) = Nz
nptot = np(1) * np(2) * np(3)
write (*,*) "FFT di array:",Nx,Ny,Nz," Stride di ",inc2y,inc3y
do k = 1,np(3)
    yp = real(k)/5.0d0
    do i = 1,2*inc3y
        xp = real(i)/3.0d0
        zp = real(i)/13.0d0
        a1(i,k) = xp**2+yp**2+zp**2
        a2(i,k) = xp+yp**2+zp
    enddo
enddo
```

```
scala = 1.0/(float(nptot))
numSpagh = 1
```

```
do loop=1, log(real(Nx)) / log(2.0) + 1
    call iniziafft(numSpagh)
```

```
! Con Tempi
GT1 = rtc()
do i = 1, NumMis
    call fft(a1,b1)
    call fft(a2,b2)
enddo
GT2 = rtc()
```

```
write (*,*) "FFT singola prec. alloc. statica con copia."
write (*,fmt='("Tempo per ",I4," FFT = ", F20.3," sec.")')
numMis*6,GT2-GT1
write (*,fmt='("Tempo medio = ",F20.3," sec./FFT")')
(GT2-GT1) / (numMis*6.0d0)
numSpagh = numSpagh * 2      !! Prox iterazione
enddo
```

```
end
```



FFT 2

```
subroutine iniziafft(m_in)
implicit none
integer ndim,naux1,naux2,naux3,naux4
include "par_64_1.inc"
parameter (naux1 = 20000)
parameter (naux2 = 70000)
parameter (naux3 = 20000)
parameter (naux4 = 70000)
cccccccccc Double
REAL*8 aux1(naux1), aux2(naux2)
REAL*8 aux3(naux3), aux4(naux4),aux3uno(naux3), aux4uno
(naux4)
REAL*8 i_aux1(naux1), i_aux2(naux2)
REAL*8 i_aux3(naux3), i_aux4(naux4),i_aux3uno(naux3),
i_aux4uno(naux4)
cccccccccc Fine Double
real a1,b1,scala1,scala2,scala
integer m,num_spag,ifin,imin,m_in
common /arealavoro/ aux1,aux2,aux3,aux4,aux3uno,aux4uno
common /arealavoroi/
i_aux1,i_aux2,i_aux3,i_aux4,i_aux3uno,i_aux4uno
common /fft_parm/ num_spag,ifin,imin
c$OMP threadprivate /arealavoroi/
c$OMP threadprivate /arealavoro/
num_spag = m_in
if (num_spag .gt. Nx/2 + 1) then
  num_spag = Nx/2
endif
m = num_spag
ifin = (Nx/2 + 1) / m * m
imin = ifin + 1
write(*,*) "Num Spagh=",num_spag
```

```
c$OMP PARALLEL
c$OMP& default(None)
c$OMP& shared(m,a1,b1)
c$OMP& shared(inc2y,inc3y,Nx,Ny,Nz)
c$OMP& private(scala1,scala2)
  scala2 = 1.0/real(Nx*Ny*Nz)
  scala1 = 1.0
  call SRCFT2(1, a1, 2*inc2y, b1, inc2y, Nx, Ny,
&           1, scala2, aux1,naux1, aux2,naux2)
  call SCFT (1, b1,inc3y,1,b1,inc3y,1, Nz,m,
&           1, scala1, aux3,naux3, aux4,naux4)
  call SCFT (1, b1,inc3y,1,b1,inc3y,1, Nz,1,
&           1, scala1, aux3uno,naux3,
aux4uno,naux4)
  scala2 = 1.0
  scala1 = 1.0
  call SCRFT2(1, b1, inc2y, a1, 2*inc2y, Nx, Ny,
&           -1, scala2, i_aux1,naux1, i_aux2,naux2)
  call SCFT (1, b1,inc3y,1,b1,inc3y,1, Nz,m,
&           -1, scala1, i_aux3,naux3, i_aux4,naux4)
  call SCFT (1, b1,inc3y,1,b1,inc3y,1, Nz,1,
&           -1, scala1, i_aux3uno,naux3,
i_aux4uno,naux4)

c$OMP END PARALLEL
end subroutine
```



FFT 3

```
subroutine fft(a1,b1)
implicit none
integer ndim,naux1,naux2,naux3,naux4
integer nptot,np(3)
integer m
include "par_64_1.inc"
parameter (ndim = 3)
parameter (naux1 = 20000)
parameter (naux2 = 70000)
parameter (naux3 = 20000)
parameter (naux4 = 70000)
complex b1(inc3y,Nz)
real a1(2*inc3y,Nz),scala1,scala2,scala
REAL*8 aux1(naux1), aux2(naux2)
REAL*8 aux3(naux3), aux4(naux4),aux3uno(naux3),
       aux4uno(naux4) real xp, yp, zp
integer i, j, k,loop, ifin, imin
common /arealavoro/ aux1,aux2,aux3,aux4,aux3uno,aux4uno
common /fft_parm/ m,ifin, imin
c$OMP threadprivate /arealavoro/
np(1) = Nx
np(2) = Ny
np(3) = Nz
nptot = np(1) * np(2) * np(3)
c  write(*,*) "Num Spagh=",m, " imin=", imin
c  Write (*,*) "Inizio combo."
scala2 = 1.0/real(Nx*Ny*Nz)
scala1 = 1.0
```

```
c$OMP PDO
do k = 1, Nz
    call SRCFT2(0, a1(1,k), 2*inc2y, b1
(1,k), inc2y, Nx, Ny,
& 1, scala2,
aux1,naux1,aux2,naux2)
enddo
c$OMP END PDO
c$OMP PDO
do j= 1, Ny
do i= 1, ifin,m
    call SCFT(0, b1(i*inc2y+j,1),inc3y,1, b1
(i*inc2y+j,1),inc3y,1,
& Nz,m, 1, scala1,
aux3,naux3, aux4,naux4)
enddo
do i= imin, Nx/2 + 1
    call SCFT(0, b1(i*inc2y+j,1),inc3y,1, b1
(i*inc2y+j,1),inc3y,1,
& Nz,1, 1, scala1,
aux3uno,naux3, aux4uno,naux4)
enddo
enddo
c$OMP END PDO
```