



# Soluzioni **OpenMP**



1. Accedere a Marconi con le credenziali assegnate:  
`ssh username@login.marconi.cineca.it`
2. Scaricare dal repository il file `OpenMP_Exercises.tar.gz`.
3. Decomprimere il file:  
`tar -xzvf OpenMP_Exercises.tar.gz`
4. Accedere, nella cartella appena decompressa, al percorso:  
`OpenMP_Exercises/C`  
oppure  
`OpenMP_Exercises/Fortran`  
a seconda del linguaggio di programmazione preferito.
5. Caricare il modulo intel:  
`module load intel`



## Esercizio 1

# Hello world!

### To do:

- (1) Compilare e lanciare “Hello world”.
- (2) Se si riscontrano errori, correggerli.
- (3) Lanciare l’eseguibile varie volte con valore di OMP\_NUM\_THREADS differenti



1. Accedere alla cartella `Hello`.
2. Compilare il file `hello.<ext>` senza aggiungere alcuna opzione:  

```
icc hello.c -o hello_s.exe
```

oppure  

```
ifort hello.F90 -o hello_s.exe
```
3. Lanciare l'eseguibile appena generato:  

```
./hello_s.exe
```

per ottenere il messaggio dal codice seriale.
4. Ricompilare il file aggiungendo l'opzione di compilazione per OpenMP:  

```
icc -qopenmp hello.c -o hello_p.exe
```

oppure  

```
ifort -qopenmp hello.F90 -o hello_p.exe
```
5. Lanciare l'eseguibile appena generato:  

```
./hello_p.exe
```

per ottenere il messaggio dal codice parallelo.



1. Quanti messaggi sono stati ottenuti? **36.**
2. Il numero di messaggi è quello atteso? **Si? No? Forse?**
3. I messaggi sono corretti? **No.**
4. Cosa c'è che non va? **Troppe ripetizioni di numeri.**
5. Aprire il file `hello.<ext>` con un editor; ad esempio:  
    `vi hello.c`  
oppure  
    `vi hello.F90`
6. Correggere, salvare e ricompilare il file con l'opzione di compilazione per OpenMP.
7. Lanciare l'eseguibile appena generato:  
    `./hello_p.exe`  
per ottenere il messaggio dal codice parallelo.



## hello.c

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif

int main(int argc, char **argv)
{
#ifdef _OPENMP
    int We_are_in;
    int Who_am__I;
    #pragma omp parallel private(Who_am__I)
    { /* The parallel block starts here */
        We_are_in = omp_get_num_threads();
        Who_am__I = omp_get_thread_num();
        #pragma omp critical
        printf("Hello from thread %d of %d\n", Who_am__I, We_are_in);
    } /* The parallel block ends here */
#else
    /* The serial block starts here */
    printf("Hello, this is a serial program.\n");
    printf("I am alone.... :(\n");
    /* The serial block ends here */
#endif

    return 0;
}
```



## hello.F90

```
Program Hello
#ifdef _OPENMP
    use omp_lib
#endif

    implicit none
    integer :: We_are_in, Who_am__I

#ifdef _OPENMP
!$omp parallel private (Who_am__I)
!   The parallel block starts here
    We_are_in = omp_get_num_threads()
    Who_am__I = omp_get_thread_num()
!$omp critical
    write(*,*) 'Hello from thread ', Who_am__I, ' of ', We_are_in
!$omp end critical
!   The parallel block ends here
!$omp end parallel
#else
!   The serial block starts here
    write(*,*) 'Hello, this is a serial program.'
    write(*,*) 'I am alone.... :('
!   The serial block ends here
#endif
end program Hello
```



1. Ora va meglio?

Decisamente.

2. Quanti messaggi sono stati ottenuti?

Sempre 36.

3. Il numero di messaggi è quello atteso?

Si? No? Forse?

4. Verificare il valore della variabile d'ambiente OMP\_NUM\_THREADS:

```
echo $OMP_NUM_THREADS
```

5. Cosa c'è che non va?

Non è definita!

6. Impostare il valore per la variabile d'ambiente OMP\_NUM\_THREADS:

```
export OMP_NUM_THREADS=4 [oppure 8, 16, 32 o 64]
```

7. Lanciare l'eseguibile generato in precedenza:

```
./hello_p.exe
```

per ottenere il messaggio dal codice parallelo.

8. Ora va meglio?

Decisamente!





## Esercizio 2

# Prodotto tra matrici

### To do:

- (1) Compilare e lanciare il codice seriale del prodotto **Matrice-Matrice**:  
l'e eseguibile va lanciato dichiarando la taglia della matrice;  
ad esempio: `./MM.exe 100` .
- (2) Prendere nota del tempo impiegato.
- (3) Parallelizzare il ciclo principale del codice seriale.
- (4) Confrontare i tempi della versione seriale e parallela.
- (5) Ripetere il confronto compilando con `-O0` ed scegliendo 1000 per la taglia.



## MM.c

```
#include <stdio.h>
...
#ifdef _OPENMP
#include <omp.h>
#endif

int main(int argc, char **argv)
{
    ...

#ifdef _OPENMP
    double time1 = omp_get_wtime();
#else
    time_t time1 = clock();
#endif

#pragma omp parallel for private(k,j)
    for (i=0; i<n; i++)
        for (k=0; k<n; k++)
            for (j=0; j<n; j++)
                c[i][j] += a[i][k]*b[k][j];
```

```
#ifdef _OPENMP
    time2 = omp_get_wtime() - time1;
#else
    time2 = (clock() - time1) / (double)CLOCKS_PER_SEC;
#endif
...
}
```



## MM.F90

```
Program MM
#ifdef _OPENMP
    use omp_lib
#endif
...

#ifdef _OPENMP
    real(kind(1.d0)) :: time1, time2
#else
    real :: time1, time2
#endif

...

#ifdef _OPENMP
    time1 = omp_get_wtime()
#else
    call cpu_time(time1)
#endif

!$omp parallel do
    do j = 1,n
        do k = 1,n
            do i = 1,n
                c(i,j) = c(i,j) + a(i,k)*b(k,j)
            end do
        end do
    end do
!$omp end parallel do

#ifdef _OPENMP
    time2 = omp_get_wtime()
#else
    call cpu_time(time2)
#endif

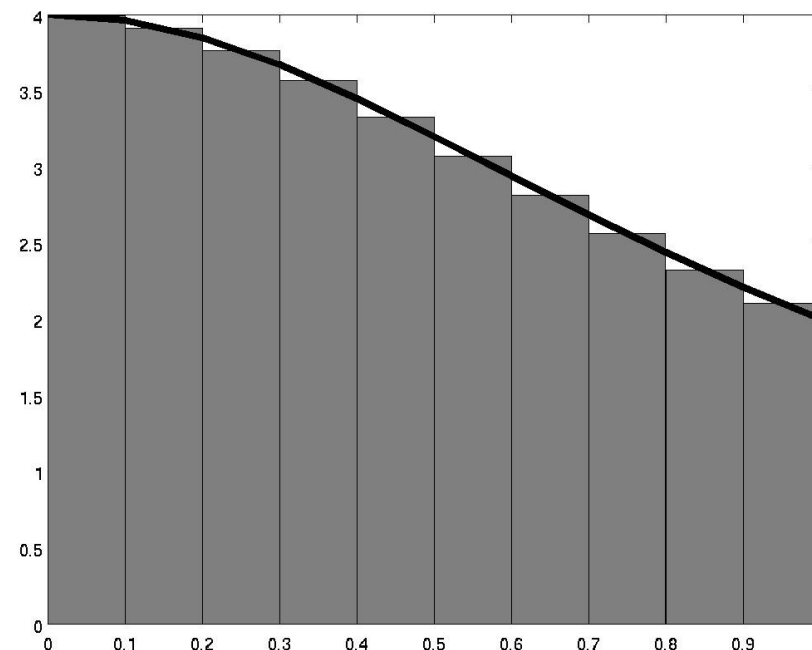
...

end program MM
```



## Esercizio 3

# Calcolo di $\pi$



### To do:

- (1) Compilare e lanciare il codice seriale del calcolo di  $\pi$ .
- (2) Prendere nota del tempo impiegato.
- (3) Parallelizzare il ciclo principale del codice seriale.
- (4) Confrontare i tempi della versione seriale e parallela.
- (5) Ripetere il confronto compilando con `-O0` e variando il numero di threads.



## pi.c

```
#include <stdio.h>
...
#ifdef _OPENMP
#include <omp.h>
#endif

int main(int argc, char **argv)
{
    ...

#ifdef _OPENMP
    double time1 = omp_get_wtime();
#else
    time_t time1 = clock();
#endif

    ...

#pragma omp parallel for private(x,f) \
reduction(+:sum)
    for (i=1; i<=intervals; i++) {
        x = dx * ((double)(i - 0.5));
        f = 4.0 / (1.0 + x*x);
        sum = sum + f;
    }

#ifdef _OPENMP
    time2 = omp_get_wtime() - time1;
#else
    time2 = (clock() - time1) / (double)CLOCKS_PER_SEC;
#endif

    ...
}
```



## pi.F90

```
program pigreco
#ifdef _OPENMP
    use omp_lib
#endif

    ...

#ifdef _OPENMP
    real(kind(1.d0)) :: time1, time2
    time1 = omp_get_wtime()
#else
    real :: time1, time2
    call cpu_time(time1)
#endif

    ...

    ...

!$omp parallel do private(x,f) &
!$omp reduction(+:psum)
    do i = 1, intervals
        x = dx*(i-0.5d0)
        f = 4.d0/(1.d0+x*x)
        psum = psum + f
    end do
!$omp end parallel do

    ...

#ifdef _OPENMP
    time2 = omp_get_wtime()
#else
    call cpu_time(time2)
#endif

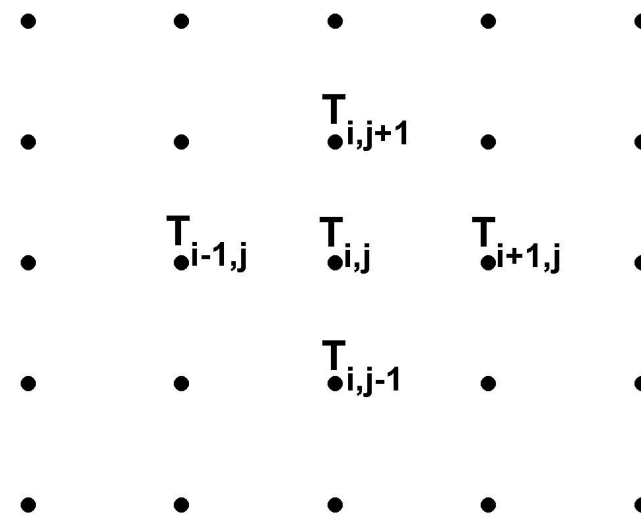
    ...

end program pigreco
```



## Esercizio 4

# Laplace (v1)



### To do:

- (1) Compilare e lanciare il codice seriale del calcolo del Laplace.
- (2) Prendere nota del tempo impiegato.
- (3) Parallelizzare il ciclo principale del codice seriale.
- (4) Confrontare i tempi della versione seriale e parallela.



## laplace-v1-3.0.c

```

#include <stdio.h>
...
#ifdef _OPENMP
#include <omp.h>
#endif

int main()
{
    ...

#ifdef _OPENMP
    double startTime = omp_get_wtime();
#else
    time_t startTime = clock();
#endif

    ...

    while(var > tol && iter <= maxIter) {
        ++iter;
        var = 0.0;
#pragma omp parallel
        {
            double pvar = 0.0;

#pragma omp for private(j)
            for (i=1; i<=n; i++)
                for (j=1; j<=n; j++) {
                    Tnew[i*n2+j] = 0.25*(
                        T[(i-1)*n2+j] + T[(i+1)*n2+j]
                        + T[i*n2+(j-1)] + T[i*n2+(j+1)] );
                    pvar = fmax(pvar, fabs(Tnew[i*n2+j] - T[i*n2+j]));
                }
#pragma omp critical
                if (pvar > var) var = pvar;
            }
            Tmp=T; T=Tnew; Tnew=Tmp;
            if (iter%100 == 0)
                printf("iter: %8u, variation = %12.41E\n", iter, var);
        }
        ...

#ifdef _OPENMP
            endTime = omp_get_wtime() - startTime;
#else
            endTime = (clock() - startTime) / (double)CLOCKS_PER_SEC;
#endif

            ...

        }
    }

```





## laplace-v1-3.1.c

```

#include <stdio.h>
...
#ifdef _OPENMP
#include <omp.h>
#endif

int main()
{
    ...

#ifdef _OPENMP
    double startTime = omp_get_wtime();
#else
    time_t startTime = clock();
#endif

    ...

while(var > tol && iter <= maxIter) {
    ++iter;
    var = 0.0;
#pragma omp parallel for private(j) reduction(max:var)
    {
        for (i=1; i<=n; i++)
            for (j=1; j<=n; j++) {
                Tnew[i*n2+j] = 0.25*(
                    T[(i-1)*n2+j] + T[(i+1)*n2+j]
                    + T[i*n2+(j-1)] + T[i*n2+(j+1)] );
                var = fmax(var, fabs(Tnew[i*n2+j] - T[i*n2+j]));
            }
        Tmp=T; T=Tnew; Tnew=Tmp;
        if (iter%100 == 0)
            printf("iter: %8u, variation = %12.41E\n", iter, var);
    }
    ...

#ifdef _OPENMP
    endTime = omp_get_wtime() - startTime;
#else
    endTime = (clock() - startTime) / (double)CLOCKS_PER_SEC;
#endif

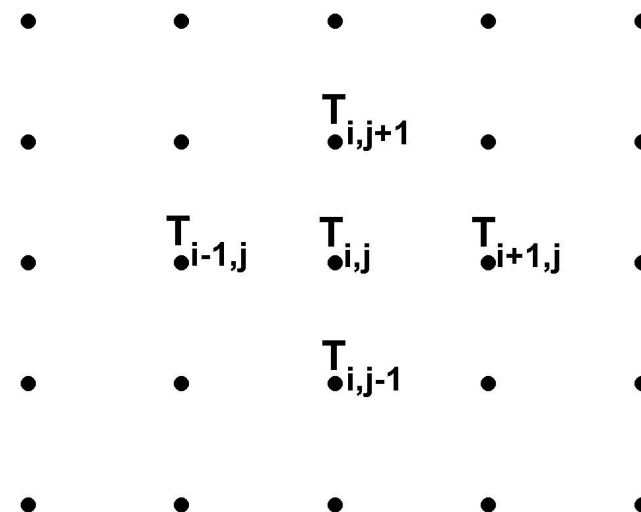
    ...
}

```



## Esercizio 5

# Laplace (v2)



### To do:

- (1) Copiare il precedente file Laplace.
- (2) Parallelizzare il ciclo while esterno al ciclo principale.
- (3) Confrontare i tempi delle due versioni.
- (4) FAC: introdurre il costrutto `workshare` nell'ultima versione (solo Fortran).



## laplace-v2-3.1.c

```
#include <stdio.h>
...
#pragma omp parallel
{
    while(var > tol && iter <= maxIter)
    {
        #pragma omp barrier
        #pragma omp single
        {
            ++iter;
            Var = 0.0;
        }
        #pragma for private(j) reduction(max:var)
        for (i=1; i<=n; i++)
            for (j=1; j<=n; j++) {
                Tnew[i*n2+j] = 0.25*( T[(i-1)*n2+j] + T[(i+1)*n2+j]
                                     + T[i*n2+(j-1)] + T[i*n2+(j+1)] );
                var = fmax(var, fabs(Tnew[i*n2+j] - T[i*n2+j]));
            }
        #pragma omp single nowait
        {
            Tmp=T; T=Tnew; Tnew=Tmp;
            if (iter%100 == 0)
                printf("iter: %8u, variation = %12.4lE\n", iter, var);
        }
    }
}
...
}
```



## laplace-v3.F90

```

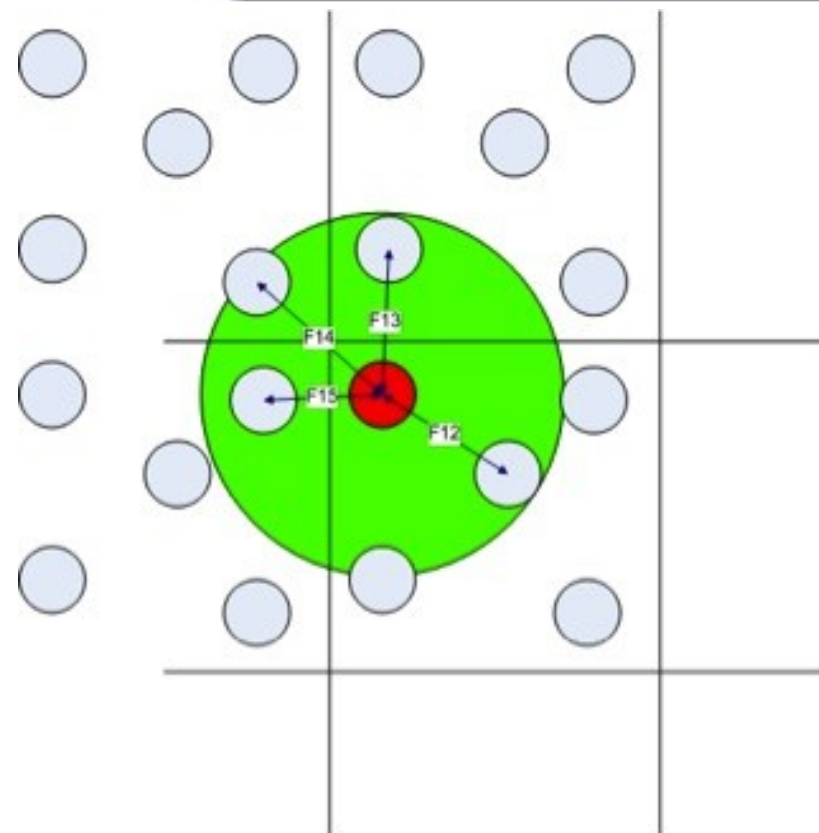
!$omp parallel
!$omp workshare
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
!$omp end workshare
!$omp single
  Tnew = T
!$omp end single nowait
  do while (var > tol .and. iter <= maxIter)
!$omp barrier
!$omp single
  iter = iter + 1
  var = 0.d0
!$omp end single
!$omp do reduction(max:var)
  do j = 1, n
    do i = 1, n
      Tnew(i,j) = 0.25d0 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
      var = max(var, abs( Tnew(i,j) - T(i,j) ))
    end do
  end do
!$omp end do
!$omp single
  Tmp =>T; T =>Tnew; Tnew => Tmp;
  if( mod(iter,100) == 0 ) write(*,"(a,i8,e12.4)") &
    ' iter, variation:', iter, var
!$omp end single nowait
  end do
!$omp end parallel

```



## Esercizio 6

# N-body



### To do:

- (1) Compilare e lanciare il codice seriale:  
`icc -O3 -DDIM=55000 Nbody.c -o Nbody_ser.exe`
- (2) Parallelizzare il codice seriale aggiornando le forze atomicamente.
- (3) Parallelizzare il codice seriale aggiornando le forze con una riduzione.
- (4) Provare differenti schemi tramite `schedule` e valutare le prestazioni.



## Nbody\_atomic.c

```
#include <stdio.h>
...
#pragma omp parallel for private(i,j,k,rij,d,d2,d3) reduction(+:ene)
for(i=0; i<nbodies; ++i)
    for(j=i+1; j<nbodies; ++j)
    {
        d2 = 0.0;
        for(k=0; k<3; ++k)
        {
            rij[k] = pos[i][k] - pos[j][k];
            d2 += rij[k]*rij[k];
        }
        if (d2 <= cut2)
        {
            d = sqrt(d2);
            d3 = d*d2;
            for(k=0; k<3; ++k)
            {
                double f = -rij[k]/d3;
                #pragma omp atomic
                forces[i][k] += f;
                #pragma omp atomic
                forces[j][k] -= f;
            }
            ene += -1.0/d;
        }
    }
...
}
```



## Nbody\_atomic.F90

```
program Nbody
...
!$omp parallel do private(i,j,k,rij,d,d2,f) reduction(+:ene)
  do i = 1, DIM
    do j = i+1, DIM
      rij(:) = pos(:,i) - pos(:,j)
      d2 = 0.d0
      do k = 1, 3
        d2 = d2 + rij(k)**2
      end do
      if (d2 .le. cut2) then
        d = sqrt(d2)
        f(:) = - 1.d0 / d**3 * rij(:)
        do k = 1, 3
!$omp atomic
          forces(k,i) = forces(k,i) + f(k)
!$omp atomic
          forces(k,j) = forces(k,j) - f(k)
        end do
        ene = ene + (-1.d0/d)
      end if
    end do
  end do
!$omp end parallel do
...
end program Nbody
```



## Nbody\_reduction.c

```

#include <stdio.h>
...
double (*pos)[3], (*forces)[3], (*gforces)[3];
...
#pragma omp parallel private(i,j,k,rij,d,d2,d3)
{
#ifdef _OPENMP
  tot_threads = omp_get_num_threads();
#else
  tot_threads = 1;
#endif
#pragma omp single
  gforces = calloc(nbodies*tot_threads, sizeof(*gforces));
  double (*pforces)[3];
#ifdef _OPENMP
  pforces = gforces + nbodies*omp_get_thread_num();
#else
  pforces = gforces;
#endif
#pragma omp for reduction(+:ene)
  for(i=0; i<nbodies; ++i)
    for(j=i+1; j<nbodies; ++j)
    {
      d2 = 0.0;
      for(k=0; k<3; ++k)
      {
        rij[k] = pos[i][k] - pos[j][k];
        d2 += rij[k]*rij[k];
      }

      if (d2 <= cut2)
      {
        d = sqrt(d2);
        d3 = d*d2;
        for(k=0; k<3; ++k)
        {
          double f = -rij[k]/d3;
          pforces[i][k] += f;
          pforces[j][k] -= f;
        }
        ene += -1.0/d;
      }
    }
#pragma omp for
  for(i=0; i<nbodies; ++i)
    for(j=0; j<tot_threads; j++)
      for(k=0; k<3; ++k)
        forces[i][k] += gforces[i+j*nbodies][k];
}
...
}

```





## Nbody\_reduction.F90

```
program Nbody
...
!$omp parallel do private(i,j,k,rij,d,d2,f) reduction(+:ene,forces)
  do i = 1, DIM
    do j = i+1, DIM
      rij(:) = pos(:,i) - pos(:,j)
      d2 = 0.d0
      do k = 1, 3
        d2 = d2 + rij(k)**2
      end do
      if (d2 .le. cut2) then
        d = sqrt(d2)
        f(:) = - 1.d0 / d**3 * rij(:)
        forces(:,i) = forces(:,i) + f(:)
        forces(:,j) = forces(:,j) - f(:)
        ene = ene + (-1.d0/d)
      end if
    end do
  end do
!$omp end parallel do
...
end program Nbody
```



## Nbody\_atomic.c

```
#include <stdio.h>
...
#pragma omp parallel for private(i,j,k,rij,d,d2,d3) reduction(+:ene) schedule(guided)
for(i=0; i<nbodies; ++i)
    for(j=i+1; j<nbodies; ++j)
    {
        d2 = 0.0;
        for(k=0; k<3; ++k)
        {
            rij[k] = pos[i][k] - pos[j][k];
            d2 += rij[k]*rij[k];
        }
        if (d2 <= cut2)
        {
            d = sqrt(d2);
            d3 = d*d2;
            for(k=0; k<3; ++k)
            {
                double f = -rij[k]/d3;
                #pragma omp atomic
                forces[i][k] += f;
                #pragma omp atomic
                forces[j][k] -= f;
            }
            ene += -1.0/d;
        }
    }
...
}
```



## Nbody\_atomic.F90

```
program Nbody
...
!$omp parallel do private(i,j,k,rij,d,d2,f) reduction(+:ene) schedule(guided)
  do i = 1, DIM
    do j = i+1, DIM
      rij(:) = pos(:,i) - pos(:,j)
      d2 = 0.d0
      do k = 1, 3
        d2 = d2 + rij(k)**2
      end do
      if (d2 .le. cut2) then
        d = sqrt(d2)
        f(:) = - 1.d0 / d**3 * rij(:)
        do k = 1, 3
!$omp atomic
          forces(k,i) = forces(k,i) + f(k)
!$omp atomic
          forces(k,j) = forces(k,j) - f(k)
        end do
        ene = ene + (-1.d0/d)
      end if
    end do
  end do
!$omp end parallel do
...
end program Nbody
```



## Nbody\_reduction.c

```

#include <stdio.h>
...
double (*pos)[3], (*forces)[3], (*gforces)[3];
...
#pragma omp parallel private(i,j,k,rij,d,d2,d3)
{
#ifdef _OPENMP
tot_threads = omp_get_num_threads();
#else
tot_threads = 1;
#endif
#pragma omp single
gforces = calloc(nbodies*tot_threads, sizeof(*gforces));
double (*pforces)[3];
#ifdef _OPENMP
pforces = gforces + nbodies*omp_get_thread_num();
#else
pforces = gforces;
#endif
#pragma omp for reduction(+:ene) schedule(guided)
for(i=0; i<nbodies; ++i)
    for(j=i+1; j<nbodies; ++j)
    {
        d2 = 0.0;
        for(k=0; k<3; ++k)
        {
            rij[k] = pos[i][k] - pos[j][k];
            d2 += rij[k]*rij[k];
        }

        if (d2 <= cut2)
        {
            d = sqrt(d2);
            d3 = d*d2;
            for(k=0; k<3; ++k)
            {
                double f = -rij[k]/d3;
                pforces[i][k] += f;
                pforces[j][k] -= f;
            }
            ene += -1.0/d;
        }
    }
}

#pragma omp for
for (i=0; i<nbodies; ++i)
    for (j=0; j<tot_threads; j++)
        for (k=0; k<3; ++k)
            forces[i][k] += gforces[i+j*nbodies][k];
}

...
}

```



## Nbody\_reduction.F90

```
program Nbody
...
!$omp parallel do private(i,j,k,rij,d,d2,f) reduction(+:ene,forces) schedule(guided)
  do i = 1, DIM
    do j = i+1, DIM
      rij(:) = pos(:,i) - pos(:,j)
      d2 = 0.d0
      do k = 1, 3
        d2 = d2 + rij(k)**2
      end do
      if (d2 .le. cut2) then
        d = sqrt(d2)
        f(:) = - 1.d0 / d**3 * rij(:)
        forces(:,i) = forces(:,i) + f(:)
        forces(:,j) = forces(:,j) - f(:)
        ene = ene + (-1.d0/d)
      end if
    end do
  end do
!$omp end parallel do
...
end program Nbody
```