24th Summer School on PARALLEL COMPUTING

Parallelizzazione Laplace 2D: MPI e MPI+OMP

- Cineca -

SuperComputing Applications and Innovation Department

CINECA

Outline

Laplace 2D: parallelizzazione

- Algoritmo semplice per mostrare le strategie di parallelizzazione OpenMP, MPI e ibrida MPI+OpenMP
  - OpenMP: sicuramente conveniente almeno per la semplicità di implementazione
  - MPI: necessaria per sfruttare architetture multinodo
  - MPI+OpenMP: per l'algoritmo testato non mostra vantaggi particolari $\rightarrow$ esempio solo a scopo didattico!

- Decomposizione MPI a blocchi 2D: conveniente soprattutto per problemi grandi perché limita l'impatto delle comunicazioni
  - decomponendo a blocchi 1D ogni processo invia e riceve $2N$ dati
  - decomponendo a blocchi 2D ogni processo invia e riceve $4N/\sqrt{N_{PROC}}$ dati, se il numero di blocchi nelle due direzioni sono uguali

Outline

Laplace 2D: MPI bloccante

Fortran

```
program laplace
   ...DICHIARAZIONE DI VARIABILI...
   call MPI_Init(ierr)
   call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)
   call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
   ...INPUT, ALLOCAZIONE, GESTIONE GRIGLIA CARTESIANA E CONDIZIONE INIZIALE...
   do while (var > tol .and. iter <= maxIter)
      iter = iter + 1 ; var = 0.d0 ; myvar = 0.d0
      buffer_s_rl(1:mymsize_y) = T(1,1:mymsize_y)
      !--- exchange boundary data with neighbours (right->left)
      call MPI_Sendrecv(buffer_s_rl,mymsize_y,MPI_DOUBLE_PRECISION,dest_rl, tag, &
                        buffer_r_rl,mymsize_y,MPI_DOUBLE_PRECISION,source_rl,tag, &
                        cartesianComm, status, ierr)
      if(source_rl >= 0) T(mymsize_x+1,1:mymsize_y) = buffer_r_rl(1:mymsize_y)
      ...SCAMBIO ALTRE HALO TRA PROCESSI MPI...
      do j = 1, mymsize_y
         do i = 1, mymsize_x
            Tnew(i,j) = 0.25d0 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
            myvar = max(myvar, abs( Tnew(i,j) - T(i,j) ))
         enddo
      enddo
      Tmp =>T; T =>Tnew; Tnew => Tmp;
      call MPI_Allreduce(myvar,var,1,MPI_DOUBLE_PRECISION,MPI_MAX,MPI_COMM_WORLD,ierr)
   end do
   ...DEALLOCAZIONE...
   call MPI_Finalize(ierr)
end program laplace
```

Outline

Laplace 2D: MPI non bloccante

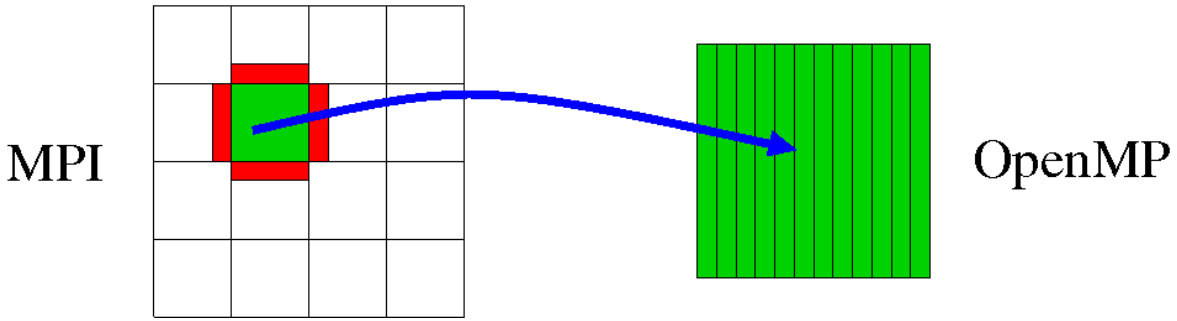
Fortran

```
do while (var > tol .and. iter <= maxIter)
   iter = iter + 1 ; var = 0.d0 ; myvar = 0.d0
   !--- exchange boundary data with neighbours (right->left)
   buffer_s_rl(1:mymsize\_y) = T(1,1:mymsize_y)
   call MPI_Isend(buffer_s_rl, mymsize_y, MPI_DOUBLE_PRECISION, &
        dest_rl, tag, cartesianComm, requests(1), ierr)
   if(source_rl >= 0) then
       call MPI_Irecv(buffer_r_rl, mymsize_y, MPI_DOUBLE_PRECISION, &
            source_rl, tag, cartesianComm, requests(2), ierr)
   endif
   ...SCAMBIO ALTRE HALO TRA PROCESSI MPI...
   ...EVOLVO I PUNTI INTERNI (ESCLUSI I BOUNDARY) ...
   call MPI_Waitall(8, requests, MPI_STATUSES_IGNORE, ierr)
   if(source_rl >= 0) T(mymsize\_x+1,1:mymsize\_y) = buffer\_r\_rl(1:mymsize\_y)
   ...COPIO GLI ALTRI 3 BUFFER ...
   ...EVOLVO I PUNTI DI BOUNDARY ...
   Tmp =>T; T =>Tnew; Tnew => Tmp;
   call MPI_Allreduce(myvar,var,1,MPI_DOUBLE_PRECISION,MPI_MAX,MPI_COMM_WORLD,ierr)
end do
```

Outline

Laplace 2D: parall. ibrida I

- Strategia di parallelizzazione ibrida: ogni blocco MPI esegue operazioni parallelizzate con OpenMP



- La strategia piú semplice è parallelizzare con OpenMP aprendo e richiudendo la regione parallela direttamente in corrispondenza del loop di aggiornamento di T
- Le chiamate MPI funzionano normalmente, perché avvengono al di fuori delle regioni *multi-threaded*
  - funzionano sia **`MPI_THREAD_SINGLE`** che **`MPI_INIT`**

Laplace 2D: *mpi_thread_single*

Fortran

```
program laplace
   ...DICHIARAZIONE DI VARIABILI...
   call MPI_Init(ierr)
   call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)
   call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
   ...INPUT, ALLOCAZIONE, GESTIONE GRIGLIA CARTESIANA E CONDIZIONE INIZIALE...
   do while (var > tol .and. iter <= maxIter)
      iter = iter + 1 ; var = 0.d0 ; myvar = 0.d0
      buffer_s_rl(1:mymsize_y) = T(1,1:mymsize_y)
      !--- exchange boundary data with neighbours (right->left)
      call MPI_Sendrecv(buffer_s_rl,mymsize_y,MPI_DOUBLE_PRECISION,dest_rl, tag, &
                        buffer_r_rl,mymsize_y,MPI_DOUBLE_PRECISION,source_rl,tag, &
                        cartesianComm, status, ierr)
      if(source_rl >= 0) T(mymsize_x+1,1:mymsize_y) = buffer_r_rl(1:mymsize_y)
      ...SCAMBIO ALTRE HALO TRA PROCESSI MPI...
      !$omp parallel do reduction(max:myvar)
      do j = 1, mymsize_y
         do i = 1, mymsize_x
            Tnew(i,j) = 0.25d0 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
            myvar = max(myvar, abs( Tnew(i,j) - T(i,j) ))
         enddo
      enddo
      !$omp end parallel do
      Tmp =>T; T =>Tnew; Tnew => Tmp;
      call MPI_Allreduce(myvar,var,1,MPI_DOUBLE_PRECISION,MPI_MAX,MPI_COMM_WORLD,ierr)
   end do
   ...DEALLOCAZIONE...
   call MPI_Finalize(ierr)
end program laplace
```

Laplace 2D: parall. ibrida II

- Allarghiamo la regione parallela OpenMP in modo da includere le chiamate MPI di scambio halo
- Il vantaggio è la possibilità di sovrapporre calcolo e comunicazioni MPI
  - i *master thread* eseguono le comunicazioni delle halo e poi aggiornano le variabili che sono vicino al bordo
  - gli altri *thread* con il *master* quando ha terminato il punto precedente eseguono i calcoli che non richiedono le halo
  - è opportuno indicare uno *scheduling* per dare al *master* meno carico sui nodi interni da aggiornare
- Con la direttiva **`master`** il blocco di codice associato è eseguito solo dal *Master Thread*, gli altri thread lo saltano senza fermarsi
- Si dovrebbe usare **`MPI_THREAD_FUNNELED`** ...

Laplace 2D: *mpi_thread_funneled*

Fortran

```
do while (var > tol .and. iter <= maxIter)
   iter = iter + 1 ; var = 0.d0 ; myvar = 0.d0 ; mastervar = 0.d0
   !$omp parallel
   !$omp master
    ...SCAMBIO HALO TRA PROCESSI MPI...
   do j = 1, mymsize_y, mymsize_y-1  ;  do i = 1, mymsize_x
       Tnew(i,j) = 0.25 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
       mastervar = max(mastervar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   do j = 1, mymsize_y  ; do i = 1, mymsize_x, mymsize_x-1
       Tnew(i,j) = 0.25 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
       mastervar = max(mastervar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   !$omp end master
   !$omp do reduction(max:myvar) schedule(dynamic,125)
   do j = 2, mymsize_y-1  ; do i = 2, mymsize_x-1
         Tnew(i,j) = 0.25d0 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
         myvar = max(myvar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   !$omp end do
   !$omp master
   myvar = max(myvar,mastervar)
   !$omp end master
   !$omp end parallel
   Tmp =>T; T =>Tnew; Tnew => Tmp;
   call MPI_Allreduce(myvar,var,1,MPI_DOUBLE_PRECISION,MPI_MAX,MPI_COMM_WORLD,ierr)
end do
```

Laplace 2D: parall. ibrida III

- Per limitare gli overhead OpenMP, è possibile far sì che la regione parallela OpenMP includa anche il ciclo **`while`**
- Analogamente a prima, è necessario utilizzare barriere OpenMP esplicite per gestire il ciclo iterativo
  - È necessaria inoltre una barriera (implicita) dopo **`MPI_Allreduce`**. Perchè?

- La chiamata **`MPI_Allreduce`** all'interno della direttiva **`single`** richiederebbe almeno **`MPI_THREAD_SERIALIZED`**
  - ma sia livelli di supporto *thread* inferiori che **`MPI_Init`** possono funzionare

Laplace 2D: *mpi_thread_serialized*

Fortran

```
!$omp parallel
do while (var > tol .and. iter <= maxIter)
   !$omp barrier
   !$omp single
   iter = iter + 1 ; var = 0.d0 ; myvar = 0.d0 ; mastervar = 0.d0
   !$omp end single
   !$omp master
    ...SCAMBIO HALO TRA PROCESSI MPI...
   do j = 1, mymsize_y, mymsize_y-1  ;  do i = 1, mymsize_x
       Tnew(i,j) = 0.25 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
       mastervar = max(mastervar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   do j = 1, mymsize_y  ; do i = 1, mymsize_x, mymsize_x-1
       Tnew(i,j) = 0.25 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
       mastervar = max(mastervar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   !$omp end master
   !$omp do reduction(max:myvar) schedule(dynamic,125)
   do j = 2, mymsize_y-1 ; do i = 2, mymsize_x-1
         Tnew(i,j) = 0.25d0 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
         myvar = max(myvar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   !$omp end do
   !$omp single
   Tmp =>T; T =>Tnew; Tnew => Tmp;
   !$omp end single nowait
   !$omp single
   myvar = max(myvar,mastervar)
   call MPI_Allreduce(myvar,var,1,MPI_DOUBLE_PRECISION,MPI_MAX,MPI_COMM_WORLD,ierr)
   !$omp end single
enddo
!$omp end parallel
```

Laplace 2D: parallel. ibrida IV

- Le 4 fasi di comunicazione sono effettuate ognuna da un thread usando **`single nowait`**
  - di fatto le chiamate MPI diventano non bloccanti
  - l'aggiornamento delle halo viene posticipato dopo quello dei nodi interni

- È necessario utilizzare **`MPI_THREAD_MULTIPLE`**
  - più *thread* che comunicano simultaneamente possono incrementare le performance di bandwidth
  - ma aumentando il supporto *thread* può aumentare anche l'*overhead* della libreria MPI

Laplace 2D: *mpi_thread_multiple*

Fortran

```
!$omp parallel
do while (var > tol .and. iter <= maxIter)
   !$omp barrier
   !$omp single
   iter = iter + 1 ; var = 0.d0 ; myvar = 0.d0
   !$omp end single
   !$omp single !--- exchange boundary data with neighbours (right->left)
   buffer_s_rl(1:mymsize_y) = T(1,1:mymsize_y) ; call MPI_Sendrecv(buffer_s_rl,...)
   if(source_rl >= 0) T(mymsize_x+1,1:mymsize_y) = buffer_r_rl(1:mymsize_y)
   !$omp end single nowait
    ...SCAMBIO ALTRE HALO TRA PROCESSI MPI TRA SINGLE NOWAIT...
   !$omp do reduction(max:myvar) schedule(dynamic,125)
   do j = 2, mymsize_y-1 ; do i = 2, mymsize_x-1
      Tnew(i,j) = 0.25d0 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
      myvar = max(myvar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   !$omp end do
   !$omp do reduction(max:myvar)
   do j = 1, mymsize_y, mymsize_y-1  ;  do i = 1, mymsize_x
      Tnew(i,j) = 0.25 * ( T(i-1,j) + T(i+1,j) + T(i,j-1) + T(i,j+1) )
      myvar = max(myvar, abs( Tnew(i,j) - T(i,j) ))
   enddo  ; enddo
   !$omp end do
    ...AGGIORNAMENTO ALTRE HALO...
   !$omp single
   Tmp =>T; T =>Tnew; Tnew => Tmp;
   !$omp end single nowait
   !$omp single
   call MPI_Allreduce(myvar,var,1,MPI_DOUBLE_PRECISION,MPI_MAX,MPI_COMM_WORLD,ierr)
   !$omp end single
enddo
!$omp end parallel
```

Scaling Laplace 2D

- Strong scaling - Griglia 5000 $\times$ 5000 - 200 iterate
- Configuriamo l'ambiente e completiamo le tabelle
  - `module load module load profile/advanced`
  - `module load intel/cs-xe-2015--binary intelmpi/5.0.1--binary`
- MPI blocking *vs non-blocking*

MPI	*1*	*2*	*4*	*8*	*16*	*32*
Blocking						
Non-blocking						

- *MPI+OMP (MPI_THREAD_SINGLE version)*

MPI/OMP	*1*	*2*	*4*	*8*	*16*
1					
2					
4					
8					
16					
32					

- *Attenzione al settaggio dei processi per il caso ibrido e multi-nodo!*

Scaling Laplace 2D

- Strong scaling - Griglia 5000 $\times$ 5000 - 200 iterate
- Configuriamo l'ambiente e completiamo le tabelle
  - `module load module load profile/advanced`
  - `module load intel/cs-xe-2015--binary intelmpi/5.0.1--binary`
- MPI blocking *vs non-blocking*

MPI	*1*	*2*	*4*	*8*	*16*	*32*
Blocking	*9.87*	*4.91*	*2.7*	*1.97*	*1.78*	*0.97*
Non-blocking	*9.86*	*4.93*	*2.7*	*1.84*	*1.87*	*0.98*

- *MPI+OMP (MPI_THREAD_SINGLE version)*

MPI/OMP	*1*	*2*	*4*	*8*	*16*
1	*13.3*	*7.37*	*4.36*	*3.9*	*4.1*
2	*6.67*	*3.54*	*2.19*	*1.96*	*2.26*
4	*3.59*	*2.08*	*1.91*	*1.14*	-
8	*2.13*	*1.95*	*1.02*	-	-
16	*1.75*	*1.06*	-	-	-
32	*0.99*	-	-	-	-

- *Attenzione al settaggio dei processi per il caso ibrido e multi-nodo!*

Scaling Laplace 2D / 2

- Weak scaling - Griglia 800 $\times$ 800 per processo/thread - 200 iterate
- MPI blocking *vs non-blocking*

MPI	*1*	*2*	*4*	*8*	*16*	*32*
Blocking						
Non-blocking						

- *MPI+OMP (MPI_THREAD_SINGLE version)*

MPI/OMP	*1*	*2*	*4*	*8*	*16*
1					
2					
4					
8					
16					
32					

- *Conviene l'ibrido?*

Scaling Laplace 2D / 2

- Weak scaling - Griglia 800 $\times$ 800 per processo/thread - 200 iterate
- MPI blocking *vs non-blocking*

MPI	*1*	*2*	*4*	*8*	*16*	*32*
Blocking	*0.27*	*0.33*	*0.34*	*0.48*	*0.91*	*0.95*
Non-blocking	*0.28*	*0.28*	*0.31*	*0.48*	*0.91*	*0.99*

- *MPI+OMP (MPI_THREAD_SINGLE version)*

MPI/OMP	*1*	*2*	*4*	*8*	*16*
1	*0.36*	*0.46*	*0.44*	*1.1*	*2.01*
2	*0.39*	*0.40*	*0.64*	*1.03*	*2.25*
4	*0.41*	*0.63*	*0.92*	*1.16*	-
8	*0.56*	*1.11*	*1.05*	-	-
16	*0.92*	*1.03*	-	-	-
32	*1.0*	-	-	-	-

- *Conviene l'ibrido?*