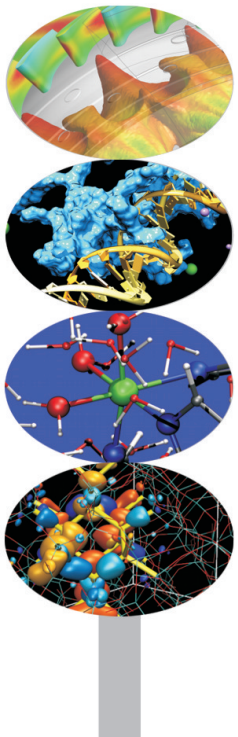
CINECA SCAI
SuperComputing Applications and Innovation

SAPIENZA
UNIVERSITÀ DI ROMA

Programmazione Avanzata

Vittorio Ruggiero

(v.ruggiero@cineca.it)



Roma, Marzo 2017

CINECA

Exercises


```
https://hpc-forge.cineca.it/files/CoursesDev/public/2017/MasterCS/
ProgrammazioneAvanzata/Esercitazioni_I.tar
tar xvf Esercitazioni\_I.tar
```

- Directory tree
- src/eser_?/fortran
- src/eser_?/c

Matrix multiplication

- Go to src/eser_1
- Write the main loop (columns rows product) to Fortran(mm.f90) or/and C(mm.c) code.
- Run the matrix multiplication code
- N=1024

Language	time	Mflops
Fortran		
C		

Makefile

- To compile
  - make
- To clean
  - make clean
- To change the compiler options
  - make "FC=ifort"
  - make "CC=icc"
  - make "OPT=fast"
- To compile using single precision arithmetic
  - make "FC=ifort -DSINGLEPRECISION"
- To compile using double precision arithmetic
  - make "FC=ifort"

Esercitazione: sommario I

- Prodotto matrice matrice:ordine dei loop
- Prodotto matrice matrice:blocking
- Prodotto matrice matrice:blocking e padding
- Misura delle prestazioni delle cache

Ordine dei loop

- Andare su *eser_2* (fortran/mm.f90 o c/mm.c)
- Misurare i tempi per N=??? al variare dell'ordine del loop
- Usare fortran e/o c
- Usare i diversi compilatori senza ottimizzazioni(-O0)

Indici	Fortran	C
i,j,k		
i,k,j		
j,k,i		
j,i,k		
k,i,j		
k,j,i		

src/eser_4/fortran/mm.f90


```
1   ...
2       integer, parameter :: n=1024  ! size of the matrix
3       integer, parameter :: step=4
4       integer, parameter :: npad=0
5   ...
6       real(my_kind) a(1:n+npad,1:n) ! matrix
7       real(my_kind) b(1:n+npad,1:n) ! matrix
8       real(my_kind) c(1:n+npad,1:n) ! matrix (destination)
9
10          do jj = 1, n, step
11            do kk = 1, n, step
12               do ii = 1, n, step
13                  do j = jj, jj+step-1
14                     do k = kk, kk+step-1
15                        do i = ii, ii+step-1
16                           c(i,j) = c(i,j) + a(i,k)*b(k,j)
17                        enddo
18  ...
```

src/eser_4/c/mm.c


```
1  #define nn (1024)
2  #define step (4)
3  #define npad (0)
4
5  double a[nn][nn+npad];        /** matrici**/
6  double b[nn][nn+npad];
7  double c[nn][nn+npad];
8  ...
9   for (ii = 0; ii < nn; ii= ii+step)
10    for (kk = 0; kk < nn; kk = kk+step)
11       for (jj = 0; jj < nn; jj = jj+step)
12          for ( i = ii; i < ii+step; i++ )
13             for ( k = kk; k < kk+step; k++ )
14                for ( j = jj; j < jj+step; j++ )
15                   c[i][j] = c[i][j] + a[i][k]*b[k][j];
16 ...
```

Blocking

- Andate su *eser*_4 (fortran/mm.f90 o c/mm.c)
- Misurare i tempi al variare della dimensione del blocking per matrici con N=???
- Usare Fortran e/o c
- Usare i diversi compilatori con livello di ottimizzazione -O3

Step	Fortran	C
4		
8		
16		
32		
64		
128		
256		

Blocking e Padding

- Andate su *eser*_4 (fortran/mm.f90 o c/mm.c)
- Misurare i tempi al variare della dimensione del blocking per matrici con N=??? e npad=???
- Usare Fortran e/o c
- Usare i diversi compilatori con livello di ottimizzazione -O3

Step	Fortran	C
4		
8		
16		
32		
64		
128		
256		

Misura delle prestazioni delle cache

- valgrind
  - Utilizzare il tool cachegrind per "scoprire" come variano le prestazioni misurate modificando l'ordine dei loop

Esercitazione: sommario II

- Prodotto matrice matrice: unrolling
- Prodotto matrice matrice: unrolling e padding

src/eser_5/fortran/mm.f90


```
1   ...
2       integer, parameter :: n=1024  ! size of the matrix
3       integer, parameter :: step=4
4       integer, parameter :: npad=0
5   ...
6       real(my_kind) a(1:n+npad,1:n) ! matrix
7       real(my_kind) b(1:n+npad,1:n) ! matrix
8       real(my_kind) c(1:n+npad,1:n) ! matrix (destination)
9
10       do j = 1, n, 2
11            do k = 1, n
12               do i = 1, n
13                  c(i,j+0) = c(i,j+0) + a(i,k)*b(k,j+0)
14                  c(i,j+1) = c(i,j+1) + a(i,k)*b(k,j+1)
15               enddo
16            enddo
17       enddo
18  ...
```

src/eser_5/c/mm.c


```
1  #define nn (1024)
2  #define step (4)
3  #define npad (0)
4
5  double a[nn][nn+npad];        /** matrici**/
6  double b[nn][nn+npad];
7  double c[nn][nn+npad];
8  ...
9  for (i = 0; i < nn; i+=2)
10    for (k = 0; k < nn; k++)
11       for (j = 0; j < nn; j++) {
12          c[i+0][j] = c[i+0][j] + a[i+0][k]*b[k][j];
13          c[i+1][j] = c[i+1][j] + a[i+1][k]*b[k][j];
14 }
15 ...
```

Unrolling loop esterno

- Andate su *eser*_5 (fortran/mm.f90 o c/mm.c)
- Misurare i tempi per matrici con N=1024 al variare dell'unrolling del loop esterno
- Usare Fortran e/o c
- Usare i compilatori gnu con livello di ottimizzazione -O3

Unrolling	Fortran	C
2		
4		
8		
16		

Unrolling loop esterno e padding

- Andate su *eser*_5 (fortran/mm.f90 o c/mm.c)
- Misurare i tempi per matrici con N=1024 al variare dell'unrolling del loop esterno con padding npad=9
- Usare Fortran e/o c
- Usare i compilatori gnu con livello di ottimizzazione -O3

Unrolling	Fortran	C
2		
4		
8		
16		

Ultimo sforzo ...

- Qual è la massima prestazione ottenibile facendo uso di:
  - blocking
  - unrolling loop esterno
  - padding
  - . . . quant'altro
- per N=2048?