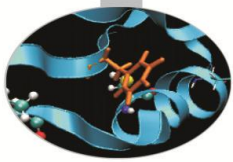
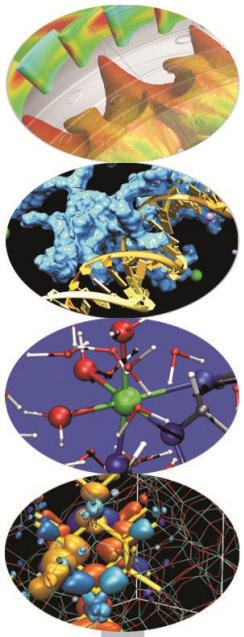
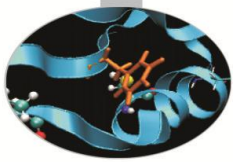


Compiler Exercise



saxpy operation

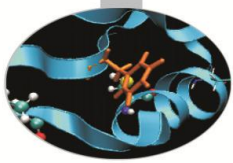


- I flags di compilazione possono aiutare nella generazione di un codice più performante.
- L'obiettivo di questo esercizio è di testare l'impatto delle opzioni di compilazione su una semplice operazione vettore-vettore.

```
#include <omp.h>
#define ARRAY_SIZE 65536
float x[ARRAY_SIZE];
float y[ARRAY_SIZE];
float z[ARRAY_SIZE];
float a;
void saxpy() {
        #pragma omp parallel for
        for(int i = 0; i < ARRAY_SIZE; i++) {
                float product = a*x[i];
                z[i] = product + y[i]; }
}
```

File: saxpy.c
main.c

saxpy operation



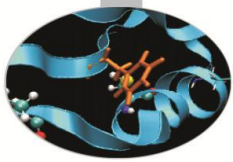
•Vedere la lista completa dei flags di ottimizzazione con il comando:

```
g++ --help=optimizers
```

•Compilare usando diversi livelli di ottimizzazione:

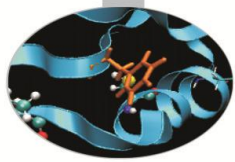
```
g++ -O0 -S saxpy.c -o saxpy-O0.s
g++ -O3 -fno-tree-vectorize -S saxpy.c -o saxpy-O3.s
g++ -O3 -funroll-loops -fno-tree-vectorize -S saxpy.c -o saxpy-O3-unrolled.s
g++ -O3 -funroll-loops -ftree-vectorize -S saxpy.c -o saxpy-O3-unrolled-vector.s
g++ -O3 -funroll-loops -ftree-vectorize -c -fopenmp saxpy.c -o saxpy-O3-unrolled-
vector-openmp.o

g++ -static saxpy-O0.s main.c -o saxpy-O0
g++ -static saxpy-O3.s main.c -o saxpy-O3
g++ -static saxpy-O3-unrolled.s main.c -o saxpy-O3-unrolled
g++ -static saxpy-O3-unrolled-vector.s main.c -o saxpy-O3-unrolled-vector
g++ -static saxpy-O3-unrolled-vector-openmp.o main.c -o saxpy-O3-unrolled-vector-
openmp -fopenmp
```

saxpy operation



- Lanciare le diverse versione e calcolare lo speed-up ottenuto rispetto alla versione non ottimizzata.
- Calcolare lo speed-up relativo. Quanto O3 è più veloce di O0? Quanto l'unrolling è più veloce di O3? Quanto l'unrolling e la vettorizzazione sono più veloci sul solo unrolling? Quanto la versione fopenmp sulla versione unroll&vectorize?

Matrix Multiplication



L'operazione di moltiplicazione matriciale `matrixmul, C=AB,` è un algoritmo alla base di molti codici scientifici.

L'obiettivo di questo esercizio è di modificare l'algoritmo di `matrixmul,` migliorandone l'efficienza.

Implementare le varie versioni richieste e misurare i tempi di calcolo.

La funzione `matrix_multiply0` implementa la base di partenza da cui operare delle ottimizzazioni.
Implementare le funzioni:

- `matrix_multiply1`: migliora la località spaziale del dato, invertendo l'ordine dei loop più interni
- `matrix_multiply2:` semplifica la deferenziazione e rimuove l'aliasing, utilizzando puntatori singoli e la keyword `restrict.`
- `matrix_multiply3:` migliora la località spaziale del dato, operando con la trasposta di B.
- `matrix_multiply4`: migliora la località spaziale del dato, semplifica la deferenziazione e rimuove l'aliasing, utilizzando puntatori singoli, la keyword `restrict e` operando con la trasposta di B.
- `matrix_multiply5`: migliora la località temporale del dato usando tecniche di cache blocking

```
File:
MatrixMultiplyBenchmark.c
MatrixMultiplyBenchmark.h
main.c
Makefile
```