

Taller 1:

Producto de matrices

El producto de $A = (a_{ij})$ y $B = (b_{ij})$, en donde A es una matriz $m \times k$ y B es una matriz $k \times n$, es la matriz $C = (c_{ij})$, en donde

$$c_{ij} = \sum_{l=1}^k a_{il}b_{lj}, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n. \quad (1)$$

1. Cree una clase llamada `MatrixOne` que represente una matriz de enteros, y cuya interfaz se presenta a continuación:

```
1  class MatrixOne {
2  private:
3      int *array;
4      int dimx;
5      int dimy;
6
7  public:
8      MatrixOne(int m, int n);
9      ~MatrixOne();
10
11     int get_dimx() { return dimx; }
12     int get_dimy() { return dimy; }
13     T get(int x, int y) { return array[y*dimx+x]; }
14     void set(int x, int y, int val) { array[y*dimx+x] = val; }
15
16     MatrixOne& operator=(MatrixOne& M);
17     void display();
18 };
```

El constructor de la clase debe separar la memoria a la que apunta `array` con el operador `new` e inicializar los elementos a 1. El destructor debe liberar la memoria. El operador `=` crea una nueva matriz igual a la que se pasa por argumento (copia). Finalmente, el método `display` debe imprimir la matriz en pantalla con una fila en cada renglón.

2. Defina el operador producto entre matrices

```
1 ||      MatrixOne operator*(MatrixOne &M1, MatrixOne &M2);
```

que recibe como argumentos dos objetos `MatrixOne` y retorna su producto.

3. Compruebe el correcto funcionamiento del operador producto y de la clase implementando el siguiente programa

```
1 ||      const int DIMX = 30;
2 ||      const int DIMY = 50;
3
4 ||      int main(int argc, char** argv) {
5 ||          int size = atoi(argv[1]);
6
7 ||          MatrixOne<int> M1(DIMY, size);
8 ||          MatrixOne<int> M2(size, DIMX);
9 ||          //M1.display();
10 ||         //M2.display();
11
12 ||         double tstart = gettimeofday();
13 ||         MatrixOne<int> R = M1*M2;
14 ||         double tstop = gettimeofday();
15
16 ||         //R.display();
17 ||         printf("%d\n", R.get(0,0));
18 ||         printf("Time: %f\n", tstop-tstart);
19
20 ||         return 0;
21 ||     }
```

en donde

```
1 ||      #include <sys/time.h>
2 ||      double gettimeofday() {
3 ||          struct timeval tp;
4 ||          gettimeofday(&tp, NULL);
5 ||          return tp.tv_sec + tp.tv_usec/(double)1.0e6;
6 ||      }
```

4. Implemente una versión de `operator*` que permita paralelizar el cálculo usando `thread` de C++.
5. Compruebe la correctitud de su implementación paralela comparándola con la serial. Verifique el *speedup* que logra su implementación repitiendo los cálculos seriales y paralelos varias veces e imprimiendo los tiempos y sus cuadrados en un archivo de texto. Con los dato del archivo calcule el speedup promedio y su desviación estándar.