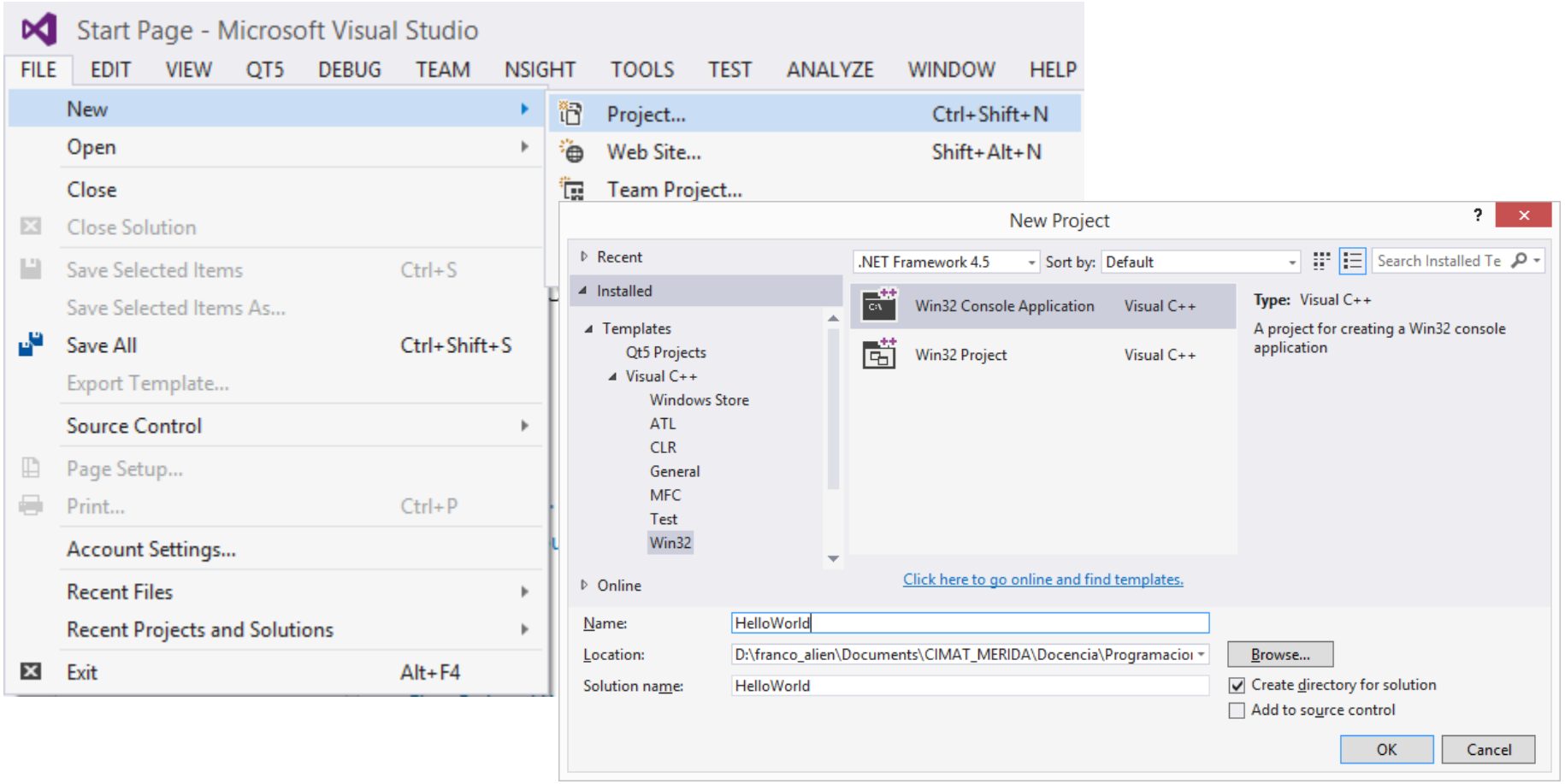
UNIDAD III. CÓMPUTO PARALELO (EJEMPLOS USANDO OPENMP)

Francisco J. Hernández López

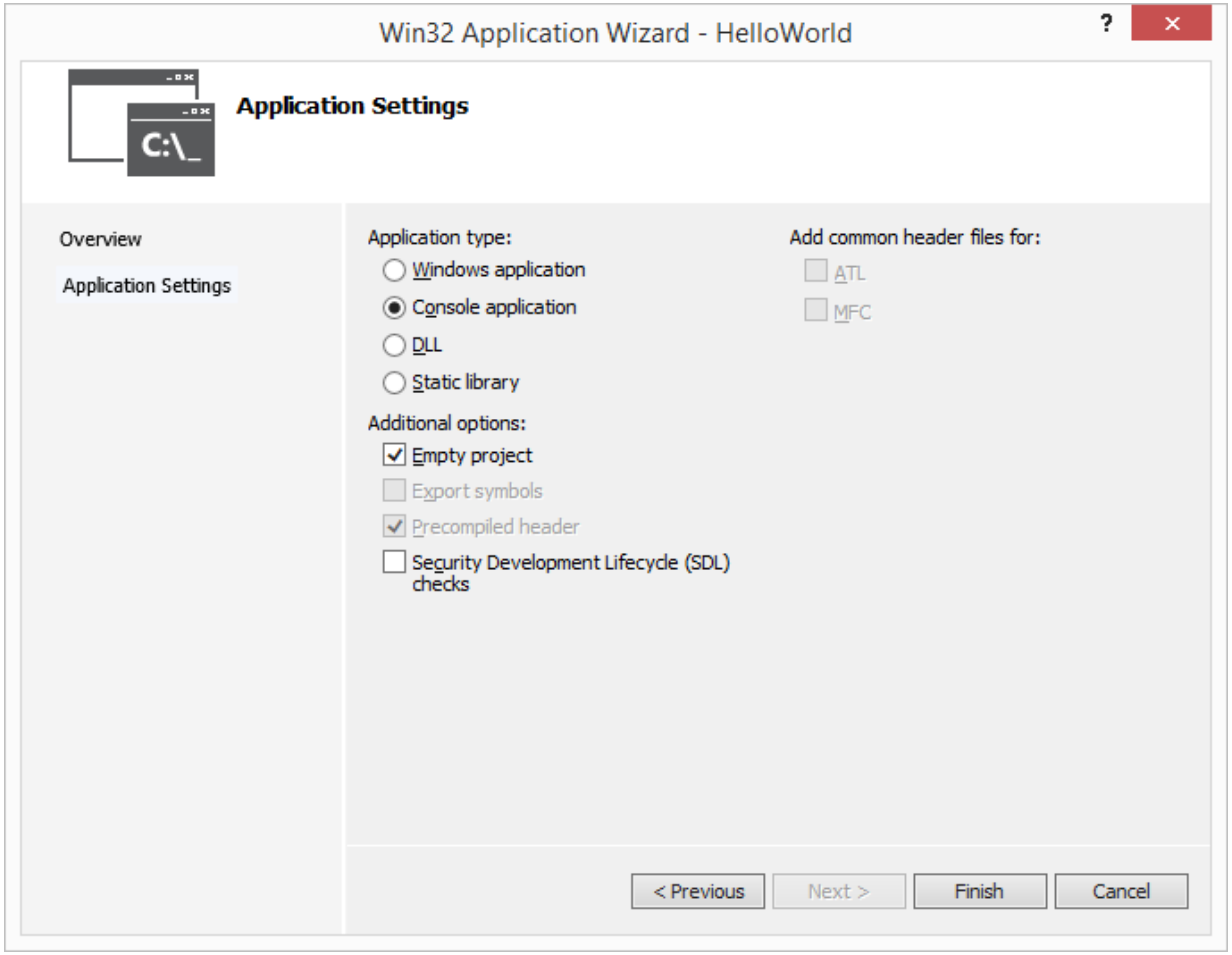
fcoj23@cimat.mx

HELLO WORLD EN VISUAL STUDIO

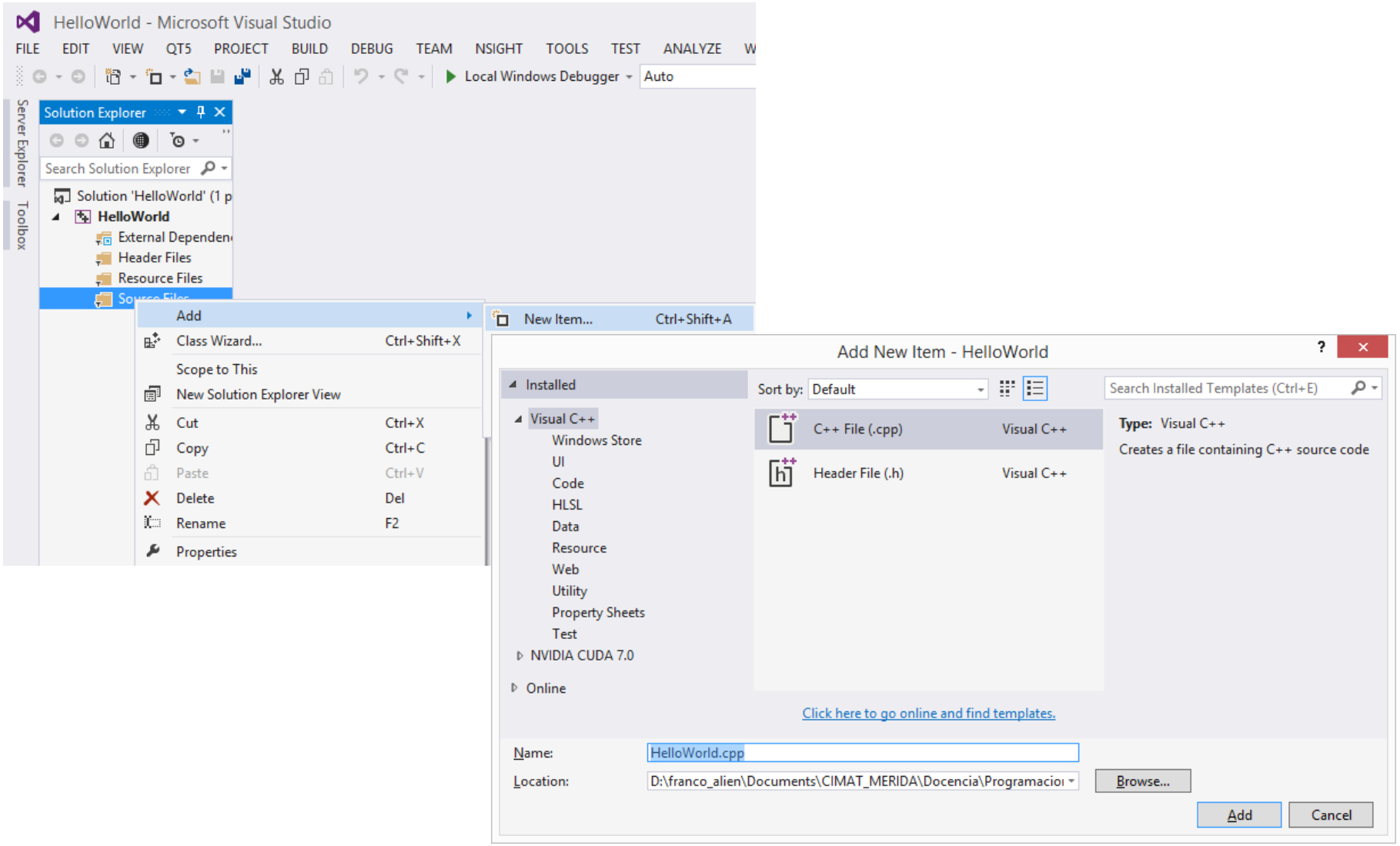
Crear un nuevo proyecto:



MODO CONSOLA Y PROYECTO VACIO



AGREGAMOS UN NUEVO ARCHIVO .CPP

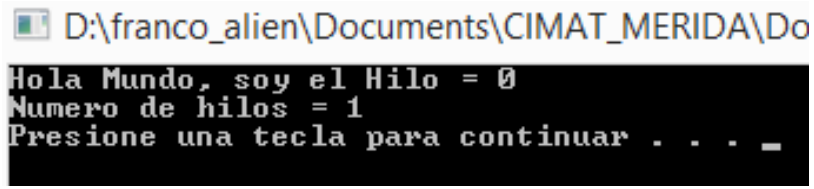


HelloWorld.cpp*

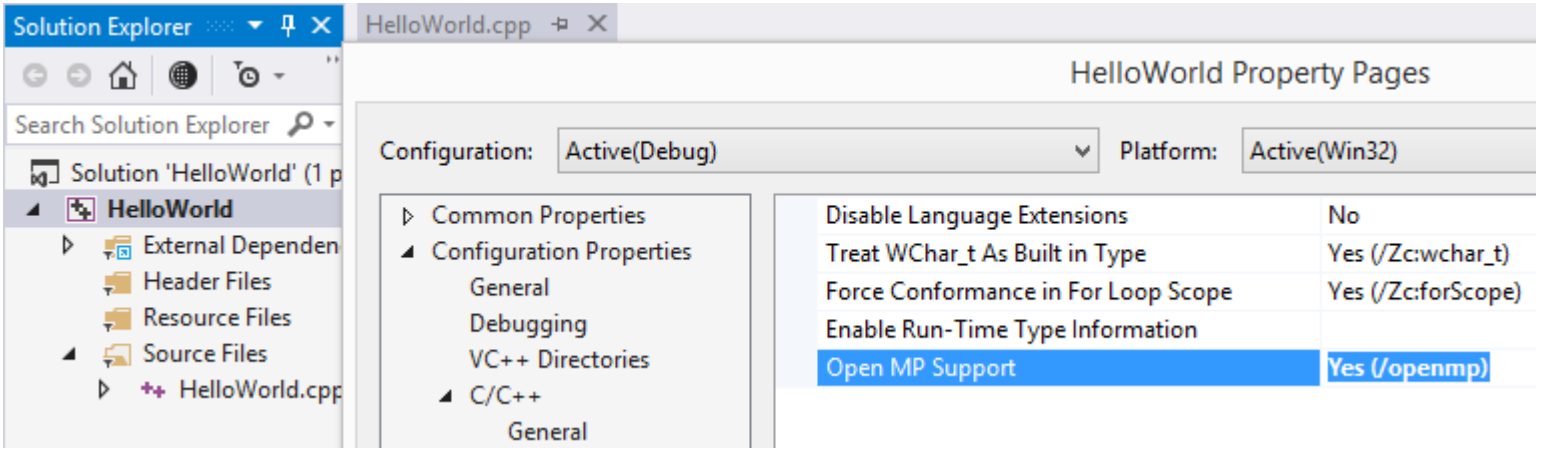
(Global Scope)

```cpp
#include <stdio.h>
#include <iostream>
//OpenMP
#include <omp.h>

int main(void){

    int nthreads, tid;

/*Se hace el Fork para generar los hilos y sus propias copias de variables*/
#pragma omp parallel private(tid)
    {
        /*Se obtiene y se imprime el id de los hilos generados*/
        tid = omp_get_thread_num();
        printf("Hola Mundo, soy el Hilo = %d\n", tid);

        /*Unicamente el hilo con id==0 hace esto*/
        if (tid == 0){
            nthreads = omp_get_num_threads();
            printf("Numero de hilos = %d\n", nthreads);
        }
    }  /* Todos los hilos se sincronizan y terminan aqui con Join */

    system("pause");
    return(0);
}
```

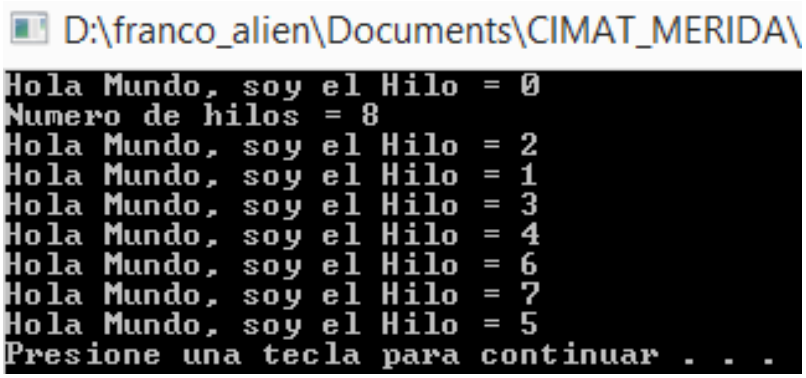
COMPILAR Y EJECUTAR



Click derecho sobre el nombre del proyecto y en Properties→C/C++→Language→OpenMP Support:



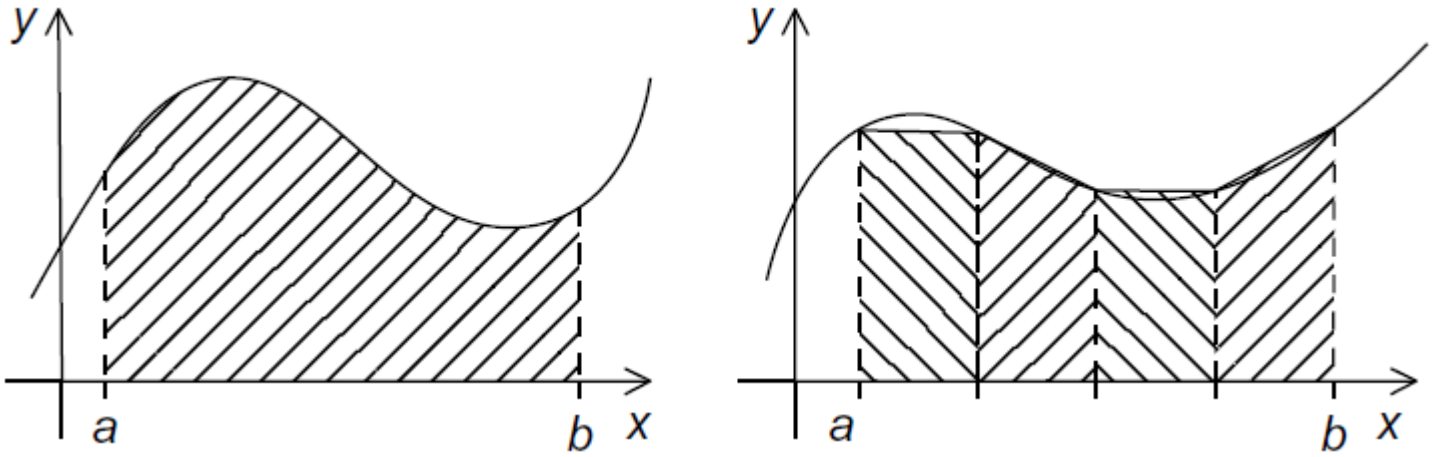
Compilar y ejecutar nuevamente:



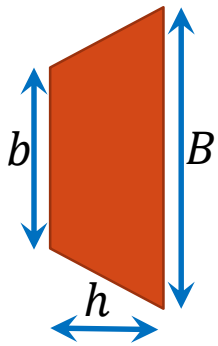
REGLA DEL TRAPECIO

- Sea $f(x)$ una función continua en $[a, b]$, con $a < b$ dos números reales, podemos estimar el área bajo la curva como sigue:
  - Dividir el intervalo $[a, b]$ en n subintervalos
  - Considerando que cada subintervalo tiene la misma longitud $h = \frac{b-a}{n}$ con $x_i = a + ih, \forall\ i = 0,1,..n$ entonces una aproximación sería:

$$A_{aprox} = h\left[\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right]$$



$$A_T = \frac{(B + b)h}{2}$$



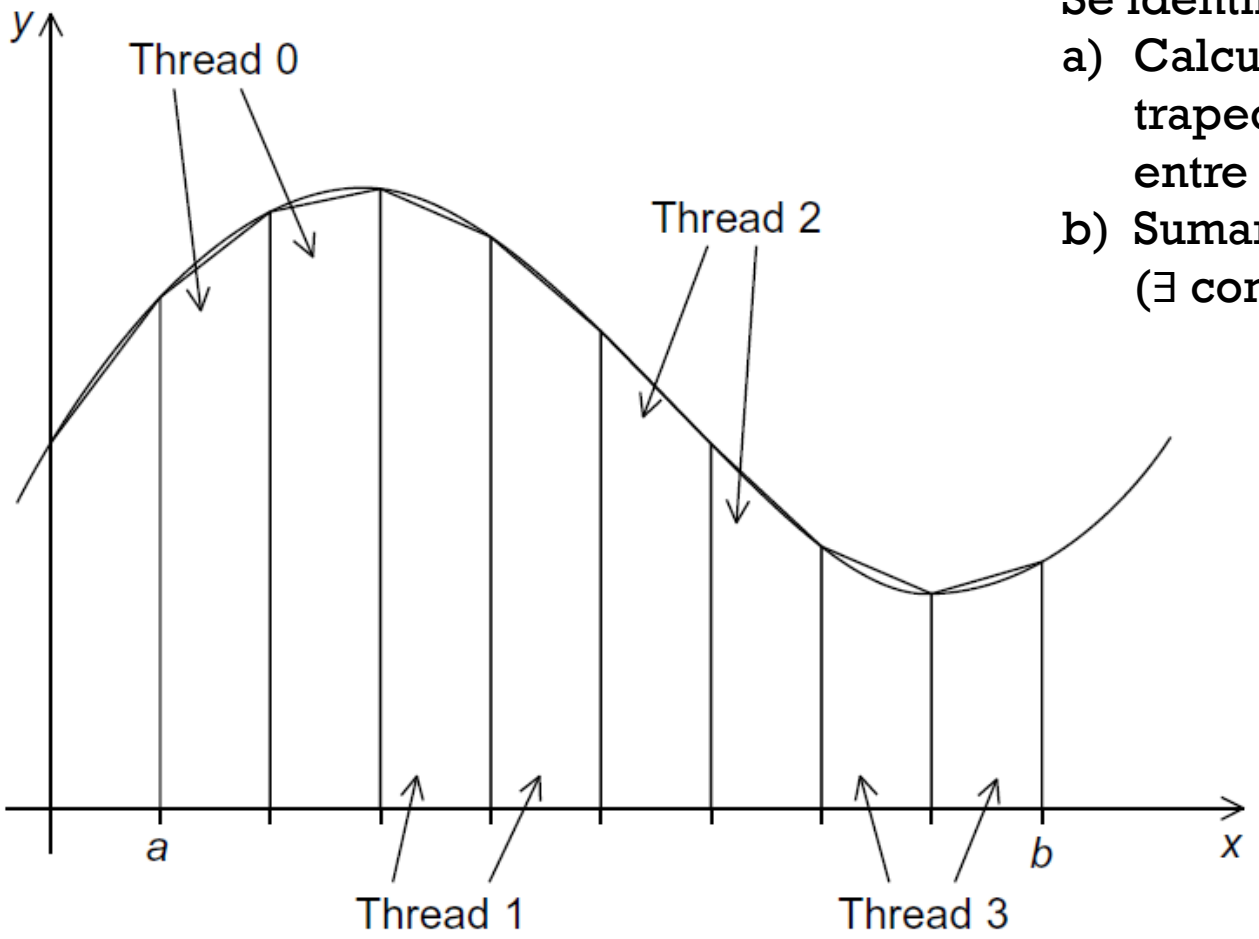
An introduction to parallel programming. Pacheco, Peter. Elsevier, 2011.

CÓDIGO SERIAL

```c
int main(void){
    //Variables

    int n = 24;//Número de subintervalos //1000000000
    double h, a, b;
    double A_aprox = 0.0;//Área aproximada
    //Valores Iniciales
    a = 1.0;
    b = 5.0;
    h = (b - a) / n;//Tamaño del subintervalo o altura el trapecio

    /*********************************************************************/
    /*Código Serial*/
    int i;
    double x_i;
    A_aprox = (Evaluar_f(a) + Evaluar_f(b)) / 2.0;
    for (i = 1; i <= n - 1; i++){
        x_i = a + i*h;
        A_aprox += Evaluar_f(x_i);
    }
    A_aprox = h*A_aprox;
    //Resultado
    printf("\nÁrea aprox.: %lf\n",A_aprox);
    /*********************************************************************/
```

```c
double Evaluar_f(double x){
    double f = 0.0;
    //Función lineal
    f = x;
    //Función cuadrática
    //f = x*x;

    return(f);
}
```

REGLA DEL TRAPECIO EN PARALELO



Se identifican dos tipos de Trabajo:

a) Calculo de las áreas de cada trapecio (no hay comunicación entre los threads)
b) Sumar todas las áreas (∃ comunicación)

An introduction to parallel programming. Pacheco, Peter. Elsevier, 2011.

USANDO OPENMP (VERSIÓN 1)

```
45      /*************************************************************/
46      /*Código Paralelo*/
47      int nthreads = 8;//Número de hilos a utilizar
48      //Op_1
49  #pragma omp parallel num_threads(nthreads)
50      {
51          AreaTrapecio(a, b, h, n, &A_aprox);
52      }
53      //Resultado
54      printf("\nÁrea aprox.: %lf\n", A_aprox);
55      /*************************************************************/
```

```
93  void AreaTrapecio(double a, double b,double h, int n, double *A_aprox_global){
94
95      double x_i, A_aprox_local;
96      double local_a, local_b;
97      int i, local_n;
98      int nthreads = omp_get_num_threads(); //Cuantos hilos hay en total
99      int tid = omp_get_thread_num();     //El id del hilo
100
101     local_n = n / nthreads;//Division exacta, a cada hilo se le asigna el mismo
102                            //numero de subintervalos. Si la división no es exacta,
103                            //obtendremos una mala aproximación.
104     local_a = a + tid*local_n*h;
105     local_b = local_a + local_n*h;
106     A_aprox_local = (Evaluar_f(local_a) + Evaluar_f(local_b)) / 2.0;
107     for (i = 1; i <= local_n - 1; i++){
108         x_i = local_a + i*h;
109         A_aprox_local += Evaluar_f(x_i);
110     }
111     A_aprox_local = h*A_aprox_local;
112     #pragma omp critical
113         *A_aprox_global += A_aprox_local;
114 }
```

USANDO OPENMP (VERSIÓN 2)

```
30      /*************************************************************/
31      /*Código Serial*/
32      int i;
33      double x_i;
34      A_aprox = (Evaluar_f(a) + Evaluar_f(b)) / 2.0;
35      for (i = 1; i <= n - 1; i++){
36          x_i = a + i*h;
37          A_aprox += Evaluar_f(x_i);
38      }
39      A_aprox = h*A_aprox;
40      //Resultado
41      printf("\nÁrea aprox.: %lf\n",A_aprox);
42      /*************************************************************/

56      /*************************************************************/
57      /*Código Paralelo*/
58      int nthreads = 4;//Número de hilos a utilizar
59      //Op_2 Clausulas: Parallel For y Reduction
60      int i;
61      double x_i;
62      A_aprox = (Evaluar_f(a) + Evaluar_f(b)) / 2.0;
63  #pragma omp parallel for num_threads(nthreads) reduction(+:A_aprox)
64      for (i = 1; i <= n - 1; i++){
65          x_i = a + i*h;
66          A_aprox += Evaluar_f(x_i);
67      }
68      A_aprox = h*A_aprox;
69      //Resultado
70      printf("\nÁrea aprox.: %lf\n", A_aprox);
71      /*************************************************************/
```

ESTIMACIÓN DE PI: π

- Una forma de obtener PI es:

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right) = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$$

```
11 #include <stdio.h>
12 #include <iostream>
13 //OpenMP
14 #include <omp.h>
15
16 int main(void){
17     //Variables
18     int k;
19     int n = 10000000000;//Número muy grande de la sumatoria
20     double pi_aprox;
21     double factor = 1.0;
22     double sum = 0.0;
23     int nthreads = 4;
24
25 //#pragma omp parallel for num_threads(nthreads) reduction(+:sum)
26     for (k = 0; k < n; k++){
27         sum += factor / (2 * k + 1);
28         factor = -factor;
29     }
30     pi_aprox = 4.0*sum;
31
32     printf("\nEl valor de PI es: %lf\n", pi_aprox);
33
34     system("pause");
35     return(0);
36 }
```

Código Serial, compilamos y ejecutamos:

```
D:\franco_alien\Documents\CIMAT_MERIDA
El valor de PI es: 3.141593
Presione una tecla para continuar . . .
```

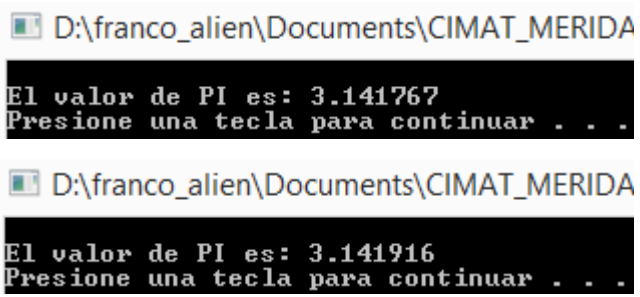
An introduction to parallel programming. Pacheco, Peter. Elsevier, 2011.

ESTIMACIÓN DE PI: π

- Código paralelo:

```
11 #include <stdio.h>
12 #include <iostream>
13 //OpenMP
14 #include <omp.h>
15
16 int main(void){
17     //Variables
18     int k;
19     int n = 10000000000;//Número muy grande de la sumatoria
20     double pi_aprox;
21     double factor = 1.0;
22     double sum = 0.0;
23     int nthreads = 4;
24
25 #pragma omp parallel for num_threads(nthreads) reduction(+:sum)
26     for (k = 0; k < n; k++){
27         sum += factor / (2 * k + 1);
28         factor = -factor;
29     }
30     pi_aprox = 4.0*sum;
31
32     printf("\nEl valor de PI es: %lf\n", pi_aprox);
33
34     system("pause");
35     return(0);
36 }
```

Compilamos y ejecutamos varias veces:



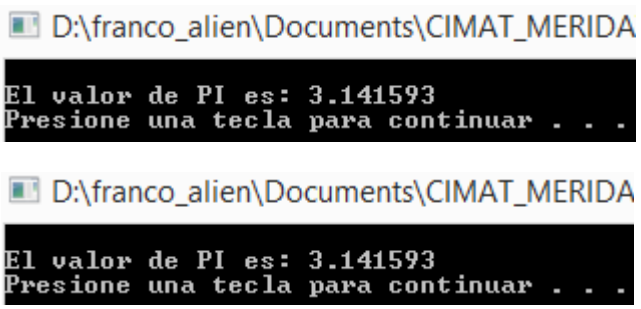
¿Por qué la aproximación es incorrecta y es diferente en cada ejecución?

ESTIMACIÓN DE PI: π

- Código paralelo:

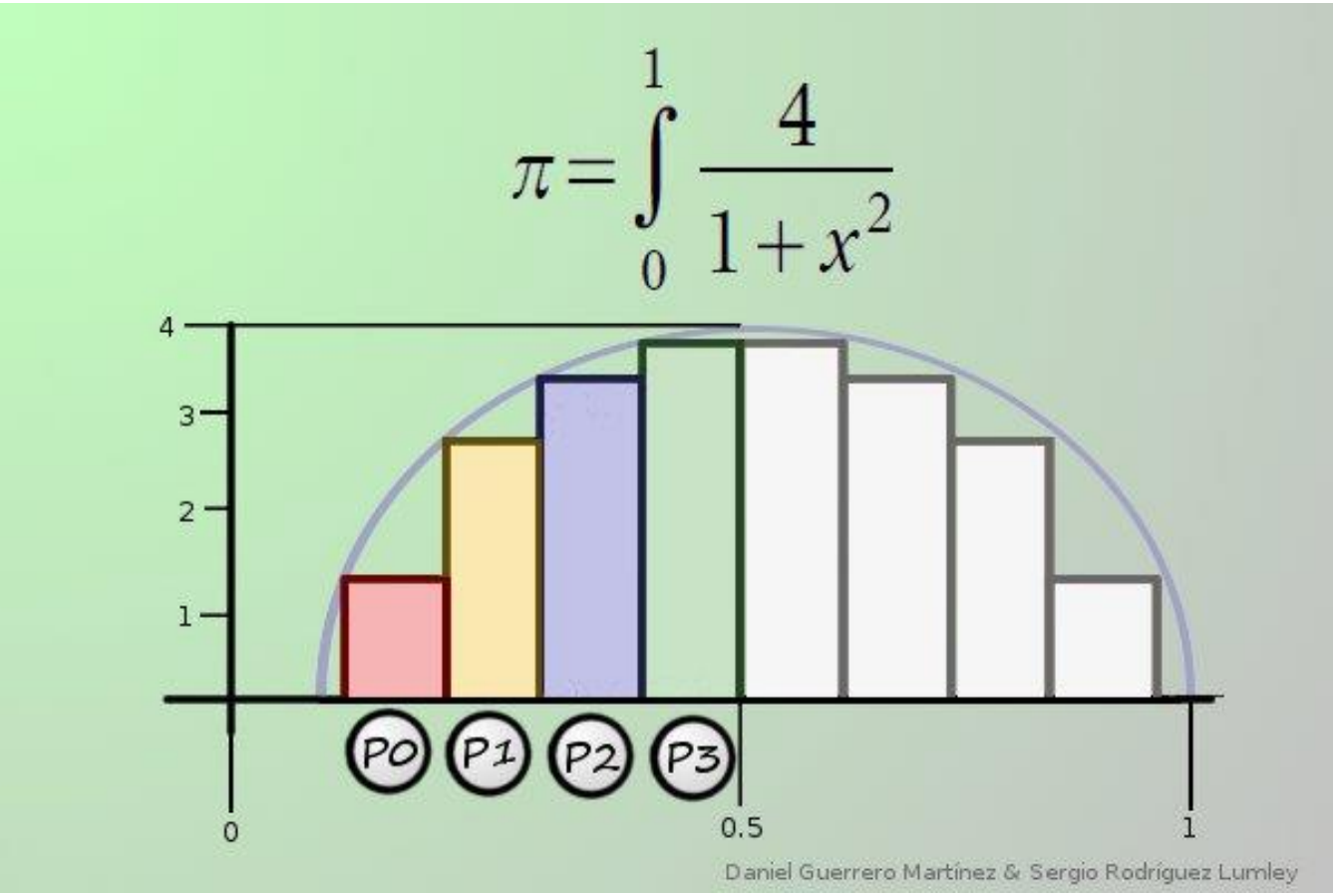
```
11 #include <stdio.h>
12 #include <iostream>
13 //OpenMP
14 #include <omp.h>
15
16 int main(void){
17     //Variables
18     int k;
19     int n = 10000000000;//Número muy grande de la sumatoria
20     double pi_aprox;
21     double factor = 1.0;
22     double sum = 0.0;
23     int nthreads = 4;
24
25 #pragma omp parallel for num_threads(nthreads) reduction(+:sum)\
26                                                 private(factor)
27     for (k = 0; k < n; k++){
28         factor = (k % 2 == 0) ? 1.0 : -1.0;
29         sum += factor / (2 * k + 1);
30         //factor = -factor;
31     }
32     pi_aprox = 4.0*sum;
33
34     printf("\nEl valor de PI es: %lf\n", pi_aprox);
35
36     system("pause");
37     return(0);
38 }
```

Compilamos y ejecutamos varias veces:



Ahora si obtenemos lo mismo que en la ejecución del código serial.

OTRA FORMA DE CALCULAR π



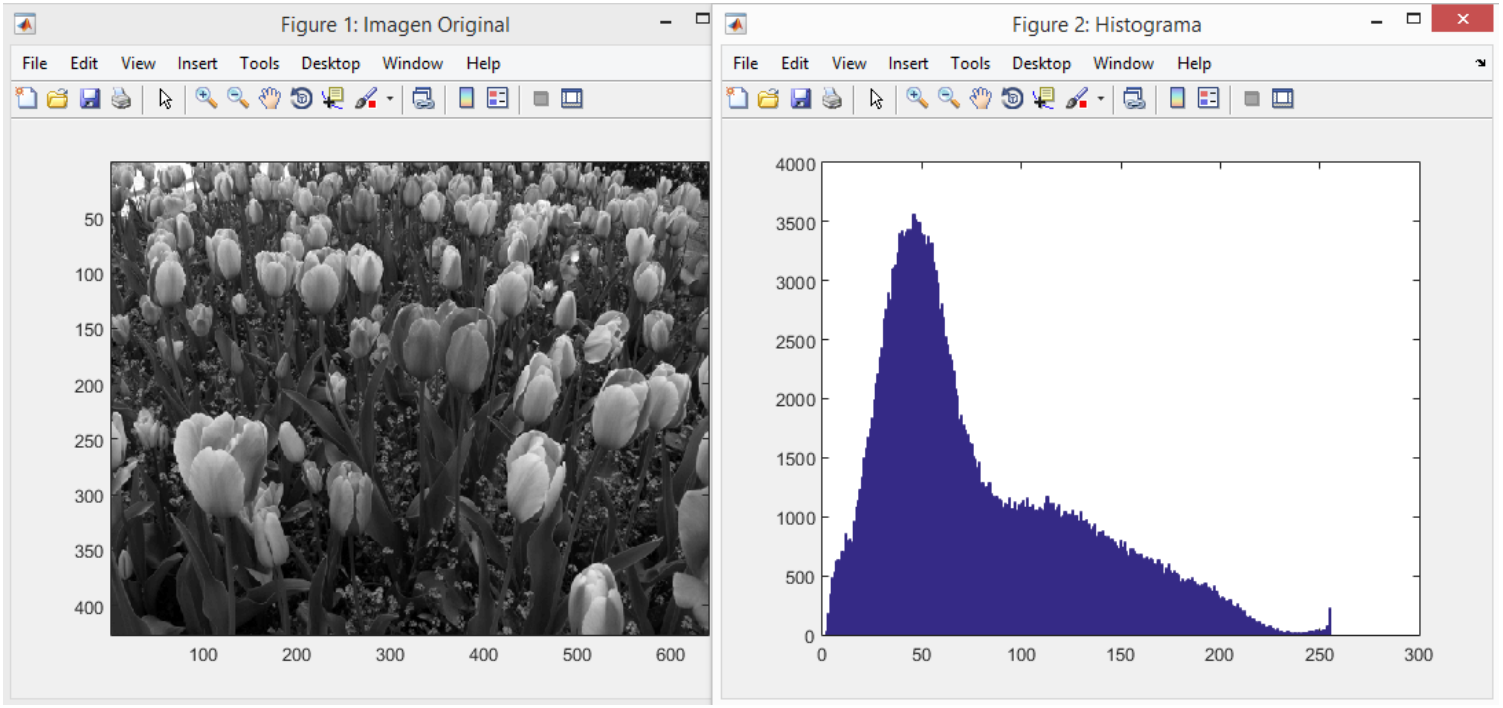
http://lsi.ugr.es/jmantas/pdp/tutoriales/tutorial_mpi.php?tuto=03_pi

OTRA FORMA DE CALCULAR π USANDO OPENMP

```
38      //Op_2 Usando sumas de áreas de rectangulos, otra forma podría ser
39      //     usando la regla del trapecio.
40      double a = 0.0;
41      double b = 1.0;
42      double h = (b - a) / n;//Tamaño de los subintervalos
43      double A_aprox = 0.0;
44      double x_i;
45
46  #pragma omp parallel for num_threads(nthreads) reduction(+:A_aprox)\
47                                                 private(x_i)
48      for (k = 0; k <= n; k++){
49          x_i = a + k*h;
50          A_aprox += 1.0 / (1 + x_i*x_i);
51      }
52      A_aprox = h*A_aprox;
53      pi_aprox = 4*A_aprox;
54
55      printf("\nEl valor de PI es: %lf\n", pi_aprox);
```

HISTOGRAMA DE UNA IMAGEN

- Inicializar con ceros el vector h_I de tamaño 256.
- Para todos los pixeles $\vec{x}$ de la imagen I
  - $idx = I(\vec{x})$ → en C/C++ o $idx = I(\vec{x}) + 1$ → en MatLab
  - $h_I(idx) = h(idx) + 1$



HISTOGRAMA DE UNA IMAGEN (SERIAL)

```
59 void CalculaHistograma(double *hI, cv::Mat Image, int nbins, int nrows, int ncols){
60     int i,j,idx;
61     //Inicializar Histograma en ceros
62     for (i = 0; i < nbins; i++){
63         hI[i] = 0.0;
64     }
65     //Recorrer la imagen y calcular el histograma
66     for (i = 0; i < nrows; i++){
67         for (j = 0; j < ncols; j++){
68             idx = Image.at<unsigned char>(i, j);
69             hI[idx] +=1.0;
70         }
71     }
72 }
```

HISTOGRAMA DE UNA IMAGEN (PARALELO V1)

```
74  void CalculaHistograma_OpenMP(double *hI, cv::Mat Image, int nbins, int nrows, int ncols){
75      int i, j, idx;
76      //Inicializar Histograma en ceros
77      for (i = 0; i < nbins; i++){
78          hI[i] = 0.0;
79      }
80      //Recorrer la imagen y calcular el histograma
81      int nthreads = 4;
82
83      /*****************************************************************/
84      //Op_1
85  #pragma omp parallel for num_threads(nthreads) private(i,j,idx)
86      for (i = 0; i < nrows; i++){
87          for (j = 0; j < ncols; j++){
88              idx = Image.at<unsigned char>(i, j);
89              #pragma omp atomic
90              hI[idx] +=1.0;
91          }
92      }
93      /*****************************************************************/
```

Asegura que una posición de memoria de hI se modifique sin que múltiples hilos intenten escribir en ella de forma simultánea.

HISTOGRAMA DE UNA IMAGEN (PARALELO V2)

```
96      /*************************************************************/
97      //Op_2
98      double *hist_temp;//Para cada hilo vamos a tener un histograma temporal
99      hist_temp = (double *)malloc(nthreads*nbins*sizeof(double));
100     memset(hist_temp, 0, nthreads*nbins*sizeof(double));
101
102 #pragma omp parallel for num_threads(nthreads) private(i,j,idx)
103     for (i = 0; i < nrows; i++){
104         int thread_id = omp_get_thread_num();
105         for (j = 0; j < ncols; j++){
106             idx = Image.at<unsigned char>(i, j);
107             hist_temp[thread_id*nbins + idx] +=1.0;
108         }
109     }
110     //Sumamos los histogramas de cada hilo
111     for (i = 0; i < nbins; i++){
112         for (j = 0; j < nthreads; j++){
113             hI[i] += hist_temp[j*nbins + i];
114         }
115     }
116     free(hist_temp);
117
118     /*************************************************************/
119 }
```

Parte altamente paralelo

Parte serial de menor costo