



Ciencia y Tecnología

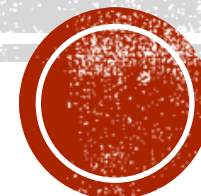
Secretaría de Ciencia, Humanidades, Tecnología e Innovación



CIMAT
UNIDAD MÉRIDA

OPENMP, EJEMPLOS

Dr. Francisco J. Hernández López
SECIHTI – CIMAT-Mérida
fcoj23@cimat.mx, www.cimat.mx/~fcoj23



EJEMPLO 1: SUMA DE VECTORES

- Suma de vectores ($c_h = a_h + b_h$)
 - Creamos los vectores “a_h”, “b_h” y “c_h” en el host
 - Inicializamos con cualquier valor los vectores “a_h” y “b_h” (Paralelizar con OpenMP esta parte)
 - Sumamos a_h y b_h; el resultado lo guardamos en c_h (Paralelizar con OpenMP esta parte)
 - Desplegamos la suma de los vectores.

EJEMPLO: SUMA DE VECTORES EN OPENMP

```
11 #include <stdio.h>
12 // #include <conio.h>
13 #include <stdlib.h>
14 #include <iostream>
15 #include <math.h>
16 // #include <sys\timeb.h>
17 #include <time.h>
18 // OpenMP
19 #include <omp.h>

36 int main(void){
37     float *a_h, *b_h, *c_h; //Punteros a arreglos en el Host
38     long int N = 100000000; //Número de elementos en el arreglo
39     long int i;
40     //struct timeb start, end;
41     double t_ini, t_fin, time_cpu_seconds;

44     size_t size_float = N * sizeof(float);
45     a_h = (float *)malloc(size_float); // Pedimos memoria en el Host
46     b_h = (float *)malloc(size_float);
47     c_h = (float *)malloc(size_float); //También se puede con cudaMallocHost
48
49     omp_set_num_threads(4); //Fijamos cuantos hilos vamos a lanzar

51     //Inicializamos los arreglos a,b en el Host
52     /*#pragma omp parallel shared(a_h,b_h,N) private(i)
53     {
54     #pragma omp for*/
55     for (i = 0; i < N; i++){
56         a_h[i] = (float)i;
57         b_h[i] = (float)(i + 1);
58     }
59     //}

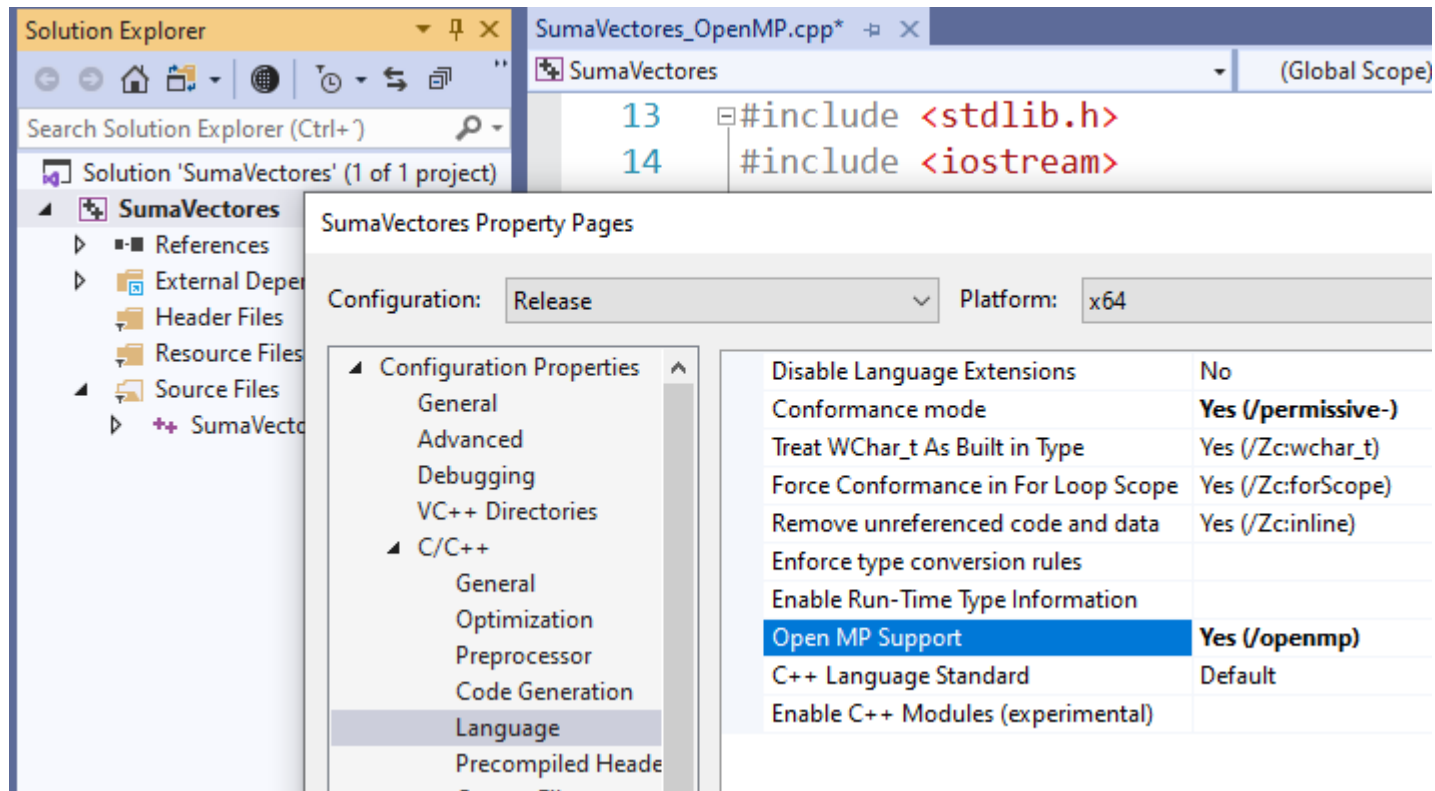
68     t_ini = clock();
69     Suma_vectores(c_h, a_h, b_h, N);
70     t_fin = clock();
71     time_cpu_seconds = (t_fin - t_ini) / CLOCKS_PER_SEC;

21 void Suma_vectores(float *c, float *a, float *b, long int N)
22 {
23     long int i;
24     #pragma omp parallel shared(a,b,c,N) private(i)
25     {
26         #pragma omp for
27         for (i = 0; i < N; i++){
28             c[i] = a[i] + b[i];
29             //c[i] = cos(a[i]) + (int)(b[i])%3;
30         }
31     }
32 }
33

85 // Liberamos la memoria del Host
86 free(a_h);
87 free(b_h);
88 free(c_h);
89 return(0);
90 }
```

COMPILACIÓN

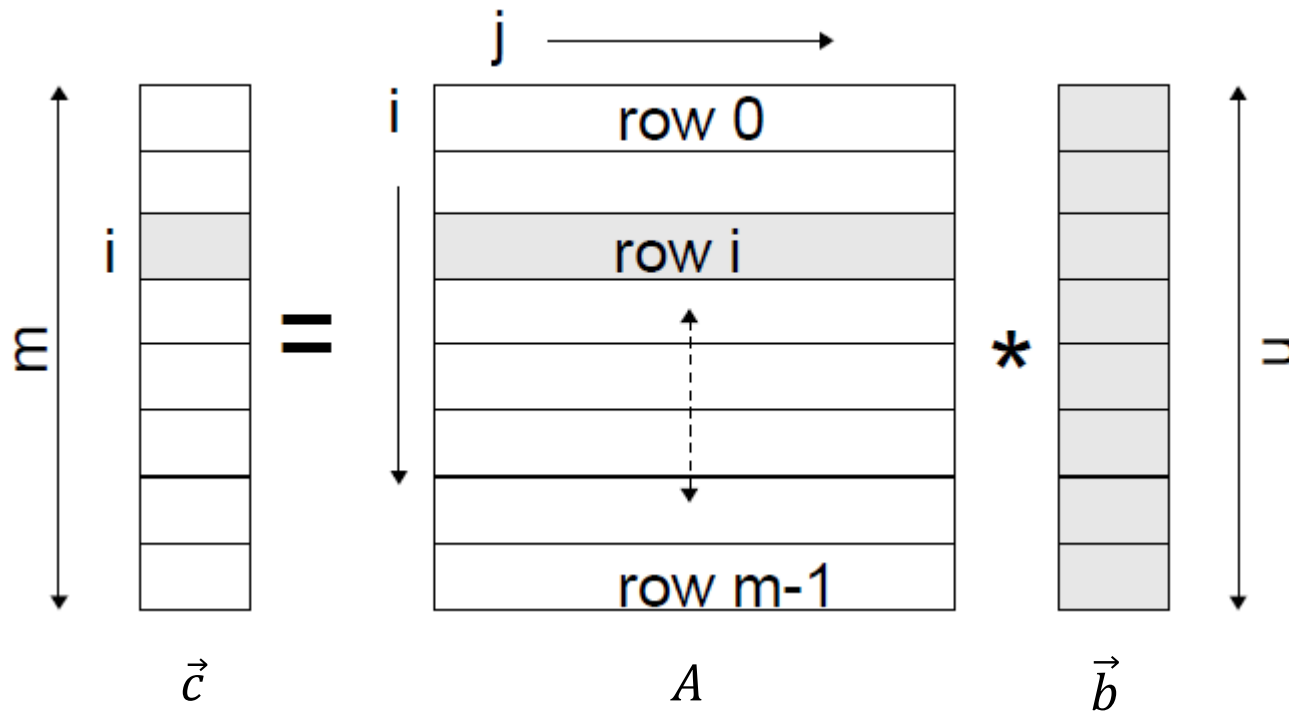
VStudio



Linux

```
g++ -fopenmp SumaVectores.cpp -o SumaVectores
gcc -fopenmp SumaVectores.cpp -o SumaVectores
```

MULTIPLICACIÓN MATRIZ-VECTOR



$$c_i = \sum_{j=1}^n A_{ij} * b_j \quad i = 1, \dots, n$$

MULT_MATRIZ_VECTOR (PARTE_1)

```
10  #include <iostream>
11  #include <stdio.h>
12  #include <stdlib.h>
13  #include <time.h>
14
15  void mostrar_datos(double *A, double *b, long int m, long int n);
16  void mult_mat_x_vector(double *c, double *A, double *b, long int m, long int n);
17  void mostrar_resultado(double* c, long int n);
18
22  int main(int argc, char *argv[]) {
23      double *c, *A, *b;
24      //int m = 10;
25      //int n = 15;
26      long int m = 10000;
27      long int n = 50000;
28      double t0, t1, time; //Medir tiempo de procesamiento
29
30      srand(0);
31      //Crear memoria
32      c = (double *)malloc(m*sizeof(double));
33      A = (double *)malloc(m*n * sizeof(double));
34      b = (double *)malloc(n * sizeof(double));
35      double bytes_reservados = m*n * sizeof(double) + m * sizeof(double) + n * sizeof(double);
36      printf("\nBytes reservados: %lf\n", bytes_reservados);
37      printf("GBytes reservados: %lf\n", bytes_reservados/(1024*1024*1024));
```

MULT_MATRIZ_VECTOR (PARTE_2)

```
39  t0 = clock();
40  //Llenando matriz A y vector b
41  for (long int i = 0; i < m; i++) {
42      for (long int j = 0; j < n; j++) {
43          long int idx = i*n + j;
44          //A[idx] = round(((double)rand() / RAND_MAX)*10);
45          A[idx] = 1.0;
46      }
47  }
48  for (long int j = 0; j < n; j++) {
49      //b[j] = round(((double)rand() / RAND_MAX) * 10);
50      b[j] = 2.0;
51  }
52  t1 = clock();
53  time = (t1 - t0) / CLOCKS_PER_SEC;
54  printf("\nTiempo de carga de datos: %lf seconds\n", time);
59  t0 = clock(); //solo vamos a medir el tiempo
60  //Calcular el producto Matriz x Vector
61  mult_mat_x_vector(c, A, b, m, n);
62
63  t1 = clock();
64  time = (t1 - t0) / CLOCKS_PER_SEC;
```

MULT_MATRIZ_VECTOR (PARTE_3)

```
66 //Imprimir resultado
67 mostrar_resultado(c,10);
68
69 printf("\nTiempo de procesamiento Matriz x Vector: %lf seconds\n", time);
70 //Liberar memoria
71 free(c);
72 free(A);
73 free(b);
74
75 return(0);
76 }

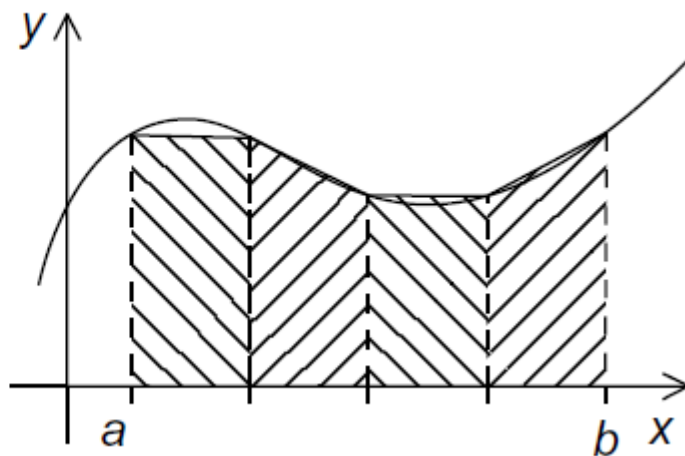
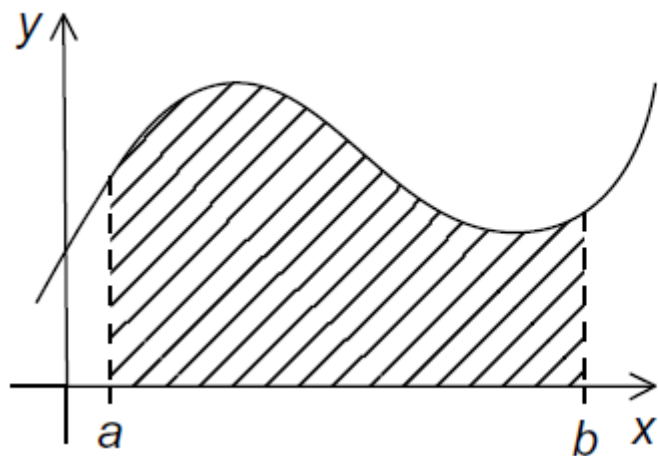
94 void mult_mat_x_vector(double *c, double *A, double *b, long int m, long int n) {
95     long int i, j, idx;
96     // #pragma omp parallel for default(none) shared(c,A,b) private(i,j,idx) firstprivate(m,n)
97     #pragma omp parallel for default(none) shared(c,A,b,m,n) private(i,j,idx)
98     for (i = 0; i < m; i++) {
99         c[i] = 0.0;
100         for (j = 0; j < n; j++) {
101             idx = i*n + j;
102             c[i] += A[idx] * b[j];
103             //c[i] += cos(A[idx]) * sin(b[j]); //Solo para que tarde más tiempo el procesamiento...
104         }
105     }
106 }

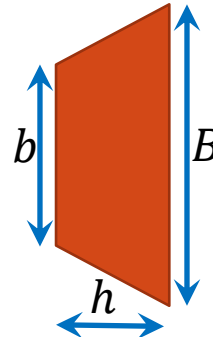
123 void mostrar_resultado(double *c, long int n) {
124     printf("\nVector c_%d,1:\n", n);
125     for (long int i = 0; i < n; i++) {
126         printf("%.2lf ", c[i]);
127     }
128     printf("\n");
129 }
```


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 $x_i = a + ih, \forall i = 0, 1, \dots, n$ entonces una aproximación sería:

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



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


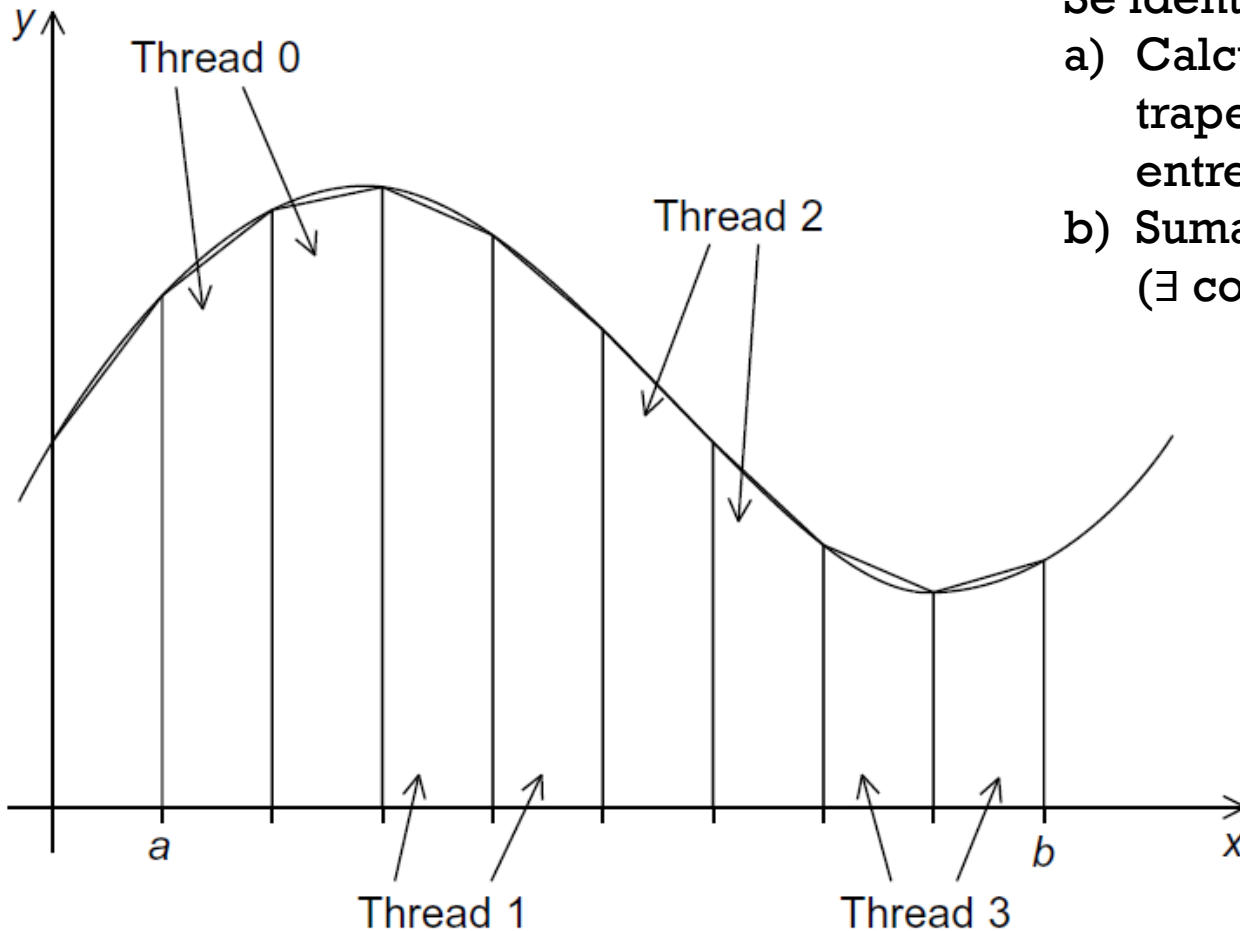
A diagram of a single trapezoid. The top base is labeled b , the bottom base is labeled B , and the height is labeled h . The area is labeled A_T .

CÓDIGO SERIAL

```
21 int main(void){
22     //Variables
23     double t0, t1, time;
24     //int n = 1000000000; //Número de subintervalos //1000000000
25     long long int n = 10000000000; //Número de subintervalos //10000000000
26     double h, a, b;
27     double A_aprox = 0.0; //Área aproximada
28     long long int i;
29     double x_i;
30     int nthreads = 2; //Número de hilos a utilizar
31     //Valores Iniciales
32     a = 1.0;
33     b = 5.0;
34     h = (b - a) / n; //Tamaño del subintervalo o altura el trapecio
35     printf("\nNum_N: %lld", n);
36     /*****/
37     /*Código Serial*/
38     t0 = clock();
39     A_aprox = (Evaluar_f(a) + Evaluar_f(b)) / 2.0;
40     for (i = 1; i <= n - 1; i++){
41         x_i = a + i*h;
42         A_aprox += Evaluar_f(x_i);
43     }
44     A_aprox = h*A_aprox;
45
46     t1 = clock();
47     time = (t1 - t0) / CLOCKS_PER_SEC;
48     printf("\nTiempo Secuencial: %lf seconds", time);
49     //Resultado
50     printf("\nÁrea aprox.: %lf\n", A_aprox);
```

```
73 double Evaluar_f(double x){
74     double f = 0.0;
75     //Función lineal
76     f = x;
77     //Función cuadrática
78     //f = x*x;
79
80     return(f);
81 }
```

REGLA DEL TRAPECIO EN PARALELO



Se identifican dos tipos de Trabajo:

- a) Cálculo de las áreas de cada trapecio (no hay comunicación entre los threads)
- b) Sumar todas las áreas (\exists comunicación)

USANDO OPENMP (VERSIÓN 1)

```
55  /*****  
56  /*Código Paralelo V1*/  
57  //Op_1  
58  t0 = clock();  
59  #pragma omp parallel num_threads(nthreads)  
60  {  
61      AreaTrapezio(a, b, h, n, &A_aprox);  
62  }  
63  A_aprox = h*((Evaluar_f(a) + Evaluar_f(b)) / 2.0 + A_aprox);  
64  //Resultado  
65  t1 = clock();  
66  time = (t1 - t0) / CLOCKS_PER_SEC;  
67  printf("\nTiempo Paralelo V1: %lf seconds", time);  
68  printf("\nÁrea aprox.: %lf\n", A_aprox);  
  
130 void AreaTrapezio(double a, double b, double h, long long int n, double *A_aprox_global){  
131  
132     double x_i, A_aprox_local;  
133     double local_a, local_b;  
134     long long int i, local_n;  
135     int nthreads = omp_get_num_threads(); //Cuantos hilos hay en total  
136     int tid = omp_get_thread_num();      //El id del hilo  
137  
138     local_n = n / nthreads; //Division exacta, a cada hilo se le asigna el mismo  
139                             //numero de subintervalos. Si la división no es exacta,  
140                             //obtendremos una mala aproximación.  
141     local_a = a + tid*local_n*h;  
142     local_b = local_a + local_n*h;  
143     //A_aprox_local = (Evaluar_f(local_a) + Evaluar_f(local_b)) / 2.0;  
144     A_aprox_local = 0.0;  
145     for (i = 1; i <= local_n - 1; i++){  
146         x_i = local_a + i*h;  
147         A_aprox_local += Evaluar_f(x_i);  
148     }  
149     //A_aprox_local = h*A_aprox_local;  
150     #pragma omp critical  
151         *A_aprox_global += A_aprox_local;  
152 }
```

USANDO OPENMP (VERSIÓN 2)

```
37  /*Código Serial*/
38  t0 = clock();
39  A_aprox = (Evaluar_f(a) + Evaluar_f(b)) / 2.0;
40  for (i = 1; i <= n - 1; i++){
41      x_i = a + i*h;
42      A_aprox += Evaluar_f(x_i);
43  }
44  A_aprox = h*A_aprox;
```

```
71  /*****
72  /*Código Paralelo V2*/
73  t0 = clock();
74  //Op_2 Clausulas: Parallel For y Reduction
75  A_aprox = (Evaluar_f(a) + Evaluar_f(b)) / 2.0; //Aunque no inicializamos en 0.0,
76  //dentro de la región paralela cada hilo inicializa su
77  //copia de A_aprox en cero
78  #pragma omp parallel for num_threads(nthreads) reduction(+:A_aprox)
79  for (i = 1; i <= n - 1; i++) {
80      x_i = a + i * h;
81      A_aprox += Evaluar_f(x_i);
82  } //Cuando finaliza la región paralela el reduction agrega al resultado el valor que tenia la variable:
83  //A_aprox= (Evaluar_f(a) + Evaluar_f(b)) / 2.0 + A_aprox
84  A_aprox = h*A_aprox;
85  //Resultado
86  t1 = clock();
87  time = (t1 - t0) / CLOCKS_PER_SEC;
88  printf("\nTiempo Paralelo V2: %lf seconds", time);
89  printf("\nÁrea aprox.: %lf\n", A_aprox);
90  /*****
```

Operador	Valor Inicial
+	0
*	1
-	0
&	~0
	0
^	0
&&	1
	0

Inicialización de variables usando **reduction**

USANDO OPENMP (VERSIÓN 3)

```
91  /*****/
92  /*Código Paralelo V3*/
93  t0 = clock();
94  //Op_3 Clausulas: Parallel For, Reduction private y firstprivate
95  A_aprox = 0.0; //Ojo: Ahora estamos inicializando en 0.0
96  #pragma omp parallel for num_threads(nthreads) reduction(+:A_aprox) private(x_i,i) firstprivate(a,h)
97  for (i = 1; i <= n - 1; i++) {
98      x_i = a + i * h;
99      A_aprox += Evaluar_f(x_i);
100  }
101  A_aprox = h * ((Evaluar_f(a) + Evaluar_f(b)) / 2.0 + A_aprox);
102
103  //Resultado
104  t1 = clock();
105  time = (t1 - t0) / CLOCKS_PER_SEC;
106  printf("\nTiempo Paralelo V3: %lf seconds", time);
107  printf("\nÁrea aprox.: %lf\n", A_aprox);
108  /*****/
```

ESTIMACIÓN DE PI: π

- Una forma de obtener PI es usando la Serie de Leibniz:

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

```
10 #include <stdio.h>
11 #include <iostream>
12 #include <time.h>
13 //OpenMP
14 #include <omp.h>
15
16 int main(void){
17     //Variables
18     double t0, t1, time;
19     long long int n = 1000000000; //Número muy grande de la sumatoria 1000000000
20     long long int k;
21     double pi_aprox;
22     double factor = 1.0;
23     double sum = 0.0;
24     int nthreads = 2;
25     printf("\nNum_n: %lld\n", n);
26     /******/
27     /*Código Serial (Serie de Leibniz)*/
28     t0 = clock();
29     for (k = 0; k < n; k++) {
30         sum += factor / (2 * k + 1);
31         factor = -factor;
32     }
33     pi_aprox = 4.0 * sum;
34     t1 = clock();
35     time = (t1 - t0) / CLOCKS_PER_SEC;
36     printf("\nTiempo Secuencial: %lf seconds", time);
37     printf("\nEl valor de PI es: %.08lf\n", pi_aprox);
```

Código Serial, compilamos y ejecutamos:

D:\franco_alien\Documents\CIMAT_MERIDA

El valor de PI es: 3.141593
Presione una tecla para continuar . . .

ESTIMACIÓN DE PI: π

■ Código paralelo:

```
38  /*****/
39  //Op_1 (Serie de Leibniz): Resultados no esperados, Ojo: factor es compartida
40  t0 = clock();
41  sum = 0.0;
42  factor = 1.0;
43  #pragma omp parallel for num_threads(nthreads) reduction(+:sum)
44  for (k = 0; k < n; k++) {
45      sum += factor / (2 * k + 1);
46      factor = -factor;
47  }
48  pi_aprox = 4.0 * sum;
49  t1 = clock();
50  time = (t1 - t0) / CLOCKS_PER_SEC;
51  printf("\nTiempo Paralelo V1 (Incorrecto): %lf seconds", time);
52  printf("\nEl valor de PI es: %.08lf\n", pi_aprox);
53  /*****/
```

Compilamos y ejecutamos
varias veces:

D:\franco_alien\Documents\CIMAT_MERIDA

```
El valor de PI es: 3.141767
Presione una tecla para continuar . . .
```

D:\franco_alien\Documents\CIMAT_MERIDA

```
El valor de PI es: 3.141916
Presione una tecla para continuar . . .
```

Cómputo Paralelo, Francisco J. Hernández-López

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

ESTIMACIÓN DE PI: π

- Código paralelo:

```
54  /*****
55  //Op_2 (Serie de Leibniz)
56  t0 = clock();
57  sum = 0.0;
58  factor = 1.0;
59  #pragma omp parallel for num_threads(nthreads) reduction(+:sum)\
60                                     private(k,factor)
61  for (k = 0; k < n; k++){
62      factor = (k % 2 == 0) ? 1.0 : -1.0;
63      sum += factor / (2 * k + 1);
64      //factor = -factor;
65  }
66  pi_aprox = 4.0*sum;
67  t1 = clock();
68  time = (t1 - t0) / CLOCKS_PER_SEC;
69  printf("\nTiempo Paralelo V2: %lf seconds", time);
70  printf("\nEl valor de PI es: %.08lf\n", pi_aprox);
71
72  /*****/
```

Compilamos y ejecutamos varias veces:

D:\franco_alien\Documents\CIMAT_MERIDA

El valor de PI es: 3.141593
Presione una tecla para continuar . . .

D:\franco_alien\Documents\CIMAT_MERIDA

El valor de PI es: 3.141593
Presione una tecla para continuar . . .

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

OTRA FORMA DE CALCULAR π USANDO OPENMP

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

```
72  /******  
73  //Op_3 Usando sumas de áreas de rectángulos, otra forma podría ser  
74  //      usando la regla del trapecio.  
75  t0 = clock();  
76  double a = 0.0;  
77  double b = 1.0;  
78  double h = (b - a) / n; //Tamaño de los subintervalos  
79  double A_aprox = 0.0;  
80  double x_i;  
81  
82  #pragma omp parallel for num_threads(nthreads) reduction(+:A_aprox)\  
83                                     private(k,x_i) firstprivate(a,h)  
84  for (k = 0; k <= n; k++){  
85      x_i = a + k*h;  
86      A_aprox += 1.0 / (1 + x_i*x_i);  
87  }  
88  A_aprox = h*A_aprox;  
89  pi_aprox = 4*A_aprox;  
90  t1 = clock();  
91  time = (t1 - t0) / CLOCKS_PER_SEC;  
92  printf("\nTiempo Paralelo V3: %lf seconds", time);  
93  printf("\nEl valor de PI es: %.08lf\n", pi_aprox);  
94  /******
```

GRACIAS POR SU ATENCIÓN

Francisco J. Hernández-López

fcoj23@cimat.mx

WebPage:

www.cimat.mx/~fcoj23

