

Universidad del Valle de Guatemala  
Facultad de Ingeniería  
Departamento de Ciencias de la Computación  
Computación Paralela



## Hoja de trabajo 1

Repositorio: <https://github.com/JuanPineda115/Paralela-1>

Diego Crespo 19541  
Ale Gudiel 19232  
Juan Pablo Pineda 19087  
Gabriel Quiroz 19255  
José Pablo Ponce 19092

<b>Instrucciones</b>	<b>3</b>
<b>1. Programación con Memoria Paralela_OpenMP</b>	<b>4</b>
Programa secuencial	4
Mediciones de tiempo	5
Secciones críticas	5
Operaciones de reducción	6
Loops paralelos	7
<b>2. Loops paralelos y planificación en OpenMP</b>	<b>9</b>
Riemann2 vs Secuencial	9
OMP_trap3.C vs Trapezoidal con FOR paralelo	9
Comparación omp_trap3.c vs secuencial vs reducción	10
Mediciones de tiempo	10
ProAx.C	11
Programa Secuencial	11
prodAx_for_scope.c	13
Mediciones de tiempo	13
prodAx_for_schedule.c	14
Mediciones de tiempo	15
<b>3. Planificación y Constructos de Sincronización</b>	<b>16</b>
Mediciones de tiempo	16

# Instrucciones

Resuelva los ejercicios hechos en clase, haciendo las modificaciones usando OpenMP para crear soluciones paralelas de un programa secuencial. Deje evidencia de todo su trabajo y realice los cálculos de speedup en donde se le indique. Recuerde que para calcular el speedup de un programa usando tiempo, es necesario que realice varias mediciones y use una de las siguientes: tiempo máximo o tiempo promedio.

En todos los incisos debe dejar copia de lo siguiente:

- Función(es) principal(es) del código. La parte que contiene el cálculo
  - Tarea más importante. No incluya encabezados ni operaciones de interacción con el usuario.
  - Bloque de código modificado
- Captura de pantalla de la salida (x1)
- Mediciones de tiempo (min. 5)
- Métrica de desempeño: speedup, efficiency

# 1. Programación con Memoria Paralela\_OpenMP

## Programa secuencial

Modificaciones del código

```
double f(double x);
double trapezoides(double a, double b, int n);
int main(int argc, char* argv[]) {
    double integral;
    double a=A, b=B;
    int n=N;
    double h;
    if(argc > 1) {
        a = strtol(argv[1], NULL, 10);
        b = strtol(argv[2], NULL, 10);
    }
    integral = trapezoides(a,b,n);
    return 0;
}

double trapezoides(double a, double b, int n) {
    double integral, h;
    int k;
    h = (b-a)/n;
    integral = (f(a) + f(b)) / 2.0;
    for(k = 1; k <= n-1; k++) {
        integral += f(a + k*h);
    }
    integral = integral * h;
    return integral;
}

double f(double x) {
    double return_val;
    return_val = x*x;
    return return_val;
}
```

## Captura de pantalla

```
(hoopah@LAPTOP-N1RRGE1L) - [/mnt/c/Users/juanp/UVG/Progra/Paralela/Paralela-1/riemann]
$ time ./seq
Con n = 10000000 trapezoides, nuestra aproximacion
de la integral de 1.000000 a 40.000000 es = 21333.0000000017

real    0m0.033s
user    0m0.030s
sys     0m0.000s
```

## Mediciones de tiempo

	secuencial	riemann2	riemann3	riemann4
	0.031	0.003	9.932	0.004
	0.031	0.003	9.963	0.014
	0.03	0.002	9.883	0.005
	0.029	0.003	10.057	0.002
	0.031	0.004	9.969	0.016
<b>promedio</b>	0.0304	0.003	9.9608	0.0082
<b>speedup</b>	N/A	10.13333333	0.00305196	3.70731707
<b>eficiencia</b>	N/A	0.63333333	0.00019075	0.23170732

## Secciones críticas

### Modificaciones de código

```
double trapezoids(double a, double b, int n, double
*suma_global) {
    double integral, h, x, local_sum;
    double local_a, local_b;
    int i, local_n, k;
    h = (b-a)/n;
    integral = (f(a) + f(b)) / 2.0;
    #pragma omp parallel private(local_sum, local_a, local_b,
local_n, i)
    {
        int thread_ID = omp_get_thread_num();
        int thread_count = omp_get_num_threads();
        local_n = n/thread_count;
        local_a = a + thread_ID*local_n*h;
        local_b = local_a + local_n*h;
```

```

        printf("\nThread %d: Local n = %d, local a = %f, local b
= %f \n", thread_ID, local_n, local_a, local_b);
        local_sum = 0.0;
        for(i = 1; i <= local_n; i++) {
            x = local_a + i*h;
            local_sum += f(x);
        }
        #pragma omp critical
        *suma_global += local_sum;
        printf("\nSuma global en el thread %d = %f\n", thread_ID,
*suma_global);
    }
    integral += *suma_global;
    integral = integral * h;
    return integral;
}

```

Captura de pantalla

```

(hoopah@LAPTOP-N1RRGE1L)-[/mnt/c/Users/juanp/UNG/Progra/Paralela/Paralela-1/riemann]
$ time ./critical
With n = 32755 trapezoids, our approximation
of the integral from 1.000000 to 40.000000 is = 21328.1208647698

real    0m0.007s
user    0m0.009s
sys     0m0.012s

```

## Operaciones de reducción

Modificaciones de código

```

double trapezoides(double a, double b, int n) {
    double integral, h;
    int k;

    //---- Ancho de cada trapezoide
    h = (b-a)/n;
    //---- Valor inicial de la integral (valores extremos)
    integral = (f(a) + f(b)) / 2.0;

    #pragma omp parallel for reduction(+: integral)
    for(k = 1; k <= n-1; k++) {
        integral += f(a + k*h);
    }
}

```

```

    integral = integral * h;
    return integral;
}

```

Captura de pantalla

```

(hoopah@LAPTOP-N1RRGE1L)-[/mnt/c/Users/juanp/UVG/Progra/Paralela/Paralela-1/riemann]
$ time ./parallel
Con n = 32545 trapezoides, nuestra aproximacion
de la integral de 1.000000 a 40.000000 es = 21333.0000093341

real    0m0.006s
user    0m0.004s
sys     0m0.000s

```

## Loops paralelos

Modificaciones de código

```

double trapezoids(double a, double b, int n) {
    double integral, h, x, local_sum;
    double local_a, local_b;
    int i, local_n, k;

    //---- Width of each trapezoid
    h = (b-a)/n;

    //---- Initial value of integral (end point terms)
    integral = (f(a) + f(b)) / 2.0;

    #pragma omp parallel for private(local_sum, local_a,
local_b, local_n, i) reduction(+: integral)
    for(k = 1; k <= n-1; k++) {
        // Get local thread ID and thread count
        int thread_ID = omp_get_thread_num();
        int thread_count = omp_get_num_threads();

        //---- Local values for the thread
        local_n = n/thread_count;
        local_a = a + thread_ID*local_n*h;
        local_b = local_a + local_n*h;
        printf("\nThread %d: Local n = %d, local a = %f, local b
= %f \n", thread_ID, local_n, local_a, local_b);
        local_sum = 0.0;
        for(i = 1; i <= local_n; i++) {

```

```
        x = local_a + i*h;
        local_sum += f(x);
    }
    integral += local_sum;
}
integral = integral * h;
return integral;
}
```

Captura de pantalla

```
(hoopah@LAPTOP-N1RRGE1L)-[/mnt/c/Users/juanp/UVG/Progra/Paralela/Paralela-1/riemann]
$ time ./locals
With n = 32571 trapezoids, our approximation
of the integral from 1.000000 to 40.000000 is = 43376919.0155225992

real    0m0.652s
user    0m9.875s
sys     0m0.010s
```



## 2. Loops paralelos y planificación en OpenMP

### Riemann2 vs Secuencial

#### 1. Mayor número de trapezoides

Captura de pantalla

Riemann2	Secuencial
<pre>os@debian:~/Desktop/H1\$ time ./programa2 10 20 100 Con n = 100 trapezoides, nuestra aproximacion de la integral de 10.000000 a 20.000000 es = 2333.3500000000  real    0m0.001s user    0m0.000s sys     0m0.000s os@debian:~/Desktop/H1\$</pre>	<pre>sys    0m0.000s os@debian:~/Desktop/H1\$ time ./programa 10 20 50 Con n = 50 trapezoides, nuestra aproximacion de la integral de 10.000000 a 20.000000 es = 2333.4000000000  real    0m0.001s user    0m0.000s sys     0m0.000s</pre>

#### 2. Rango a-b mayor:

Captura de pantalla

Riemann2	Secuencial
<pre>os@debian:~/Desktop/H1\$ time ./programa2 10 20 100 Con n = 100 trapezoides, nuestra aproximacion de la integral de 10.000000 a 20.000000 es = 2333.3500000000  real    0m0.001s user    0m0.000s sys     0m0.000s</pre>	<pre>os@debian:~/Desktop/H1\$ time ./programa 10 15 100 Con n = 50 trapezoides, nuestra aproximacion de la integral de 10.000000 a 15.000000 es = 791.6750000000  real    0m0.001s user    0m0.000s sys     0m0.000s</pre>

#### 3. $f(x) = 2^x$ .

Bloque de código modificado

```
double f(double x) {
    return pow(2, x);
}
```

Captura de pantalla

Riemann2	Secuencial
<pre>Con n = 100 trapezoides, nuestra aproximacion de la integral de 10.000000 a 20.000000 es = 2333.3500000000  real    0m0.001s user    0m0.000s sys     0m0.000s</pre>	<pre>Con n = 50 trapezoides, nuestra aproximacion de la integral de 10.000000 a 20.000000 es = 2333.4000000000  real    0m0.001s user    0m0.000s sys     0m0.000s</pre>

### OMP\_trap3.C vs Trapezoidal con FOR paralelo

La diferencia entre ambos códigos es que el primero utiliza una función llamada "riemann" para calcular la aproximación de la integral mediante la regla del trapecio, mientras que el segundo utiliza la función "Trap" para realizar el mismo cálculo. Además, el Trapezoidal con FOR paralelo incluye directamente las instrucciones de cálculo de la aproximación, mientras que en el OMP\_trap3.c se utilizan variables auxiliares. También, el Trapezoidal con FOR paralelo utiliza la

directiva de compilación "#pragma omp" para realizar el for paralelo, mientras que en el OMP\_trap3.c no se utiliza esta directiva.

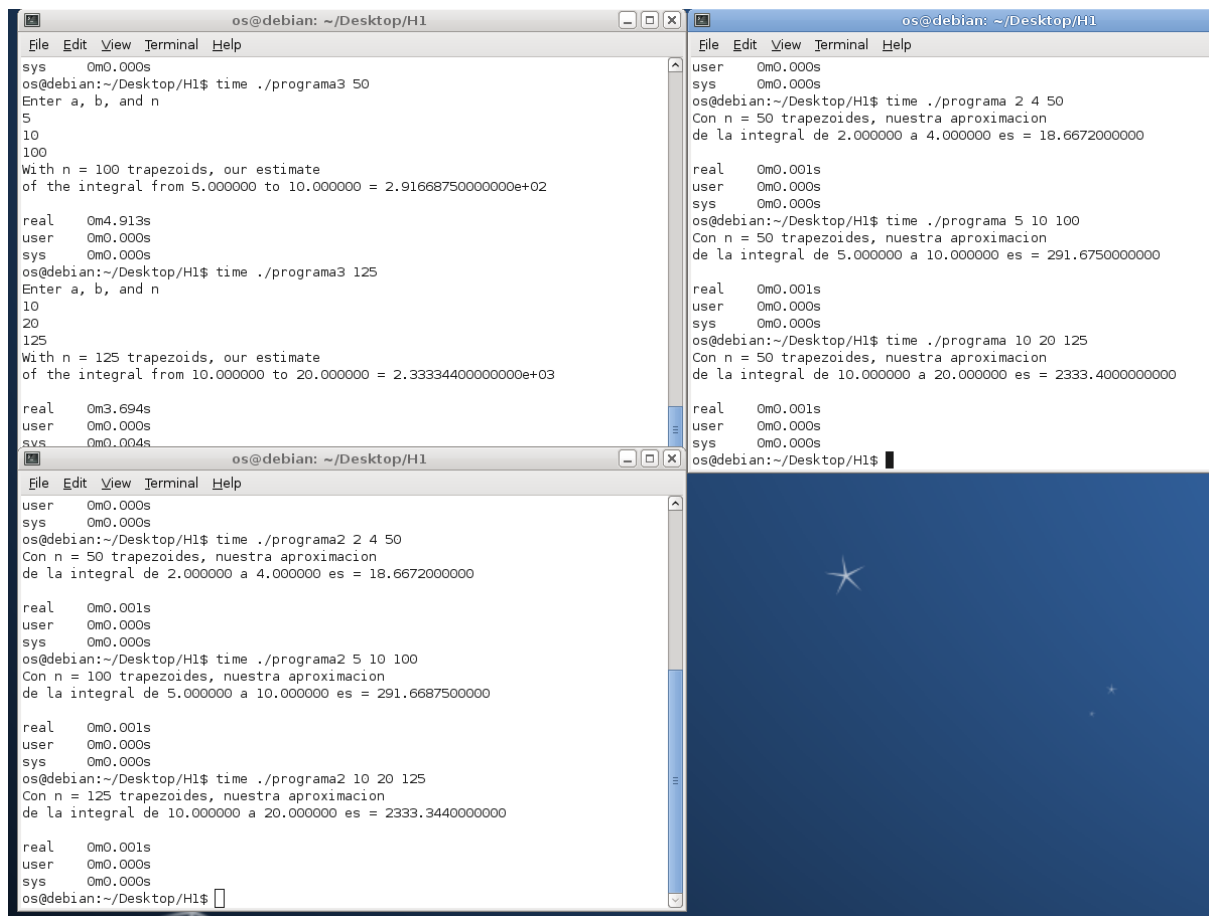
- El Trapezoidal con FOR paralelo no utiliza la función Trap() como el OMP\_trap3.C. En su lugar, el cálculo del método del trapecio se realiza directamente en la función main().
- En el Trapezoidal con FOR paralelo, se calcula el valor de h antes del bucle for principal. En el OMP\_trap3.C, el valor de h se calcula dentro de la función Trap().
- En el Trapezoidal con FOR paralelo, el cálculo de approx se realiza en dos líneas separadas: primero se calcula  $(f(a) + f(b))/2.0$ , y luego se agrega el resultado del bucle for a esta variable. En el OMP\_trap3.C, el cálculo de approx se realiza en una sola línea dentro de la función Trap().
- El Trapezoidal con FOR paralelo utiliza la directiva #pragma omp parallel for para paralelizar el bucle for. En el OMP\_trap3.C, la paralelización se realiza dentro de la función Trap() usando la misma directiva #pragma omp parallel for.
- El Trapezoidal con FOR paralelo utiliza la cláusula reduction(+: approx) para asegurarse de que la variable approx se actualice correctamente cuando se realiza la suma paralela. El OMP\_trap3.C no utiliza esta cláusula, pero logra el mismo efecto agregando los resultados de los subintervalos en una variable local my\_approx antes de agregarla a la variable approx global al final del bucle for.

## Comparación omp\_trap3.c vs secuencial vs reducción

Mediciones de tiempo

	secuencial	Reduccion	OMP3
	3.554	0.01	0.01
	4.426	0.02	0.01
	4.91	0.01	0.01
	3.69	0.02	0.01
	3.685	0.02	0.01
<b>promedio</b>	3.888	0,016	0.01
<b>speedup</b>	N/A	24.3	38.99
<b>eficiencia</b>	N/A	21.10	9.72

## Capturas de Pantalla



## ProAx.C

### Programa Secuencial

```
void prodAx(int m, int n, double * restrict A, double * restrict x,
double * restrict b){

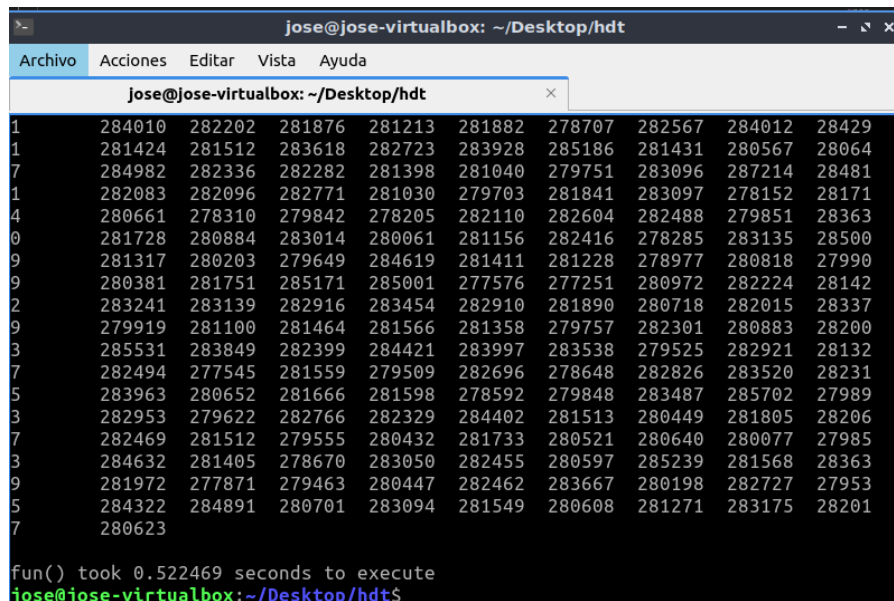
    int i, j;

    {
        for(i=0; i<m; i++){
            b[i]=0.0;

            for(j=0; j<n; j++){
                b[i] += A[i*n + j] * x[j];
            }
        }
    }
}

} //----prodAx----
```

## Captura de pantalla



```
jose@jose-virtualbox: ~/Desktop/hdt
Archivo Acciones Editar Vista Ayuda
jose@jose-virtualbox: ~/Desktop/hdt
1 284010 282202 281876 281213 281882 278707 282567 284012 28429
1 281424 281512 283618 282723 283928 285186 281431 280567 28064
7 284982 282336 282282 281398 281040 279751 283096 287214 28481
1 282083 282096 282771 281030 279703 281841 283097 278152 28171
4 280661 278310 279842 278205 282110 282604 282488 279851 28363
0 281728 280884 283014 280061 281156 282416 278285 283135 28500
9 281317 280203 279649 284619 281411 281228 278977 280818 27990
9 280381 281751 285171 285001 277576 277251 280972 282224 28142
2 283241 283139 282916 283454 282910 281890 280718 282015 28337
9 279919 281100 281464 281566 281358 279757 282301 280883 28200
3 285531 283849 282399 284421 283997 283538 279525 282921 28132
7 282494 277545 281559 279509 282696 278648 282826 283520 28231
5 283963 280652 281666 281598 278592 279848 283487 285702 27989
3 282953 279622 282766 282329 284402 281513 280449 281805 28206
7 282469 281512 279555 280432 281733 280521 280640 280077 27985
3 284632 281405 278670 283050 282455 280597 285239 281568 28363
9 281972 277871 279463 280447 282462 283667 280198 282727 27953
5 284322 284891 280701 283094 281549 280608 281271 283175 28201
7 280623

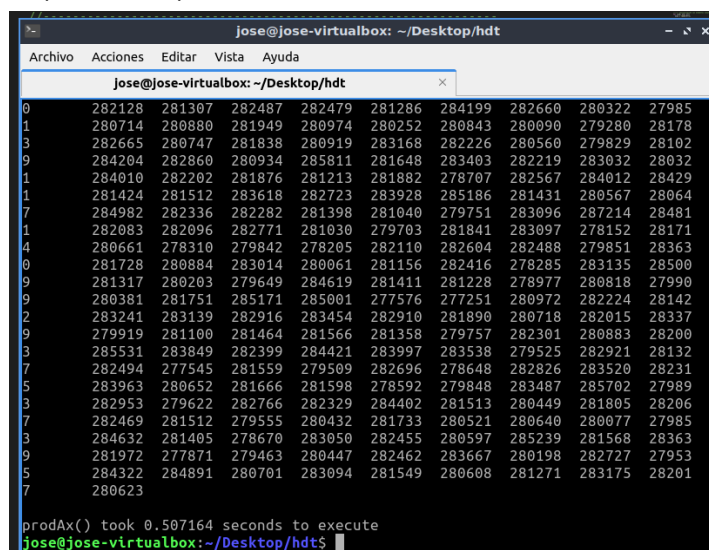
fun() took 0.522469 seconds to execute
jose@jose-virtualbox:~/Desktop/hdt$
```

## Bloque de código modificado

```
void prodAx(int m, int n, double* A, double* x, double* b) {
    int i, j;

    #pragma omp parallel for shared(m,n,A,x,b) private(i,j)
    for(i = 0; i < m; i++) {
        b[i] = 0.0; //inicialización elemento i del vec.
        for(j = 0; j < n; j++) {
            b[i] += A[i*n + j] * x[j]; //producto punto
        }
    } /*---Fin de parallel for---*/
}
```

## Captura de pantalla



```
jose@jose-virtualbox: ~/Desktop/hdt
Archivo Acciones Editar Vista Ayuda
jose@jose-virtualbox: ~/Desktop/hdt
0 282128 281307 282487 282479 281286 284199 282660 280322 27985
1 280714 280880 281949 280974 280252 280843 280090 279280 28178
3 282665 280747 281838 280919 283168 282226 280560 279829 28102
9 284204 282860 280934 285811 281648 283403 282219 283032 28032
1 284010 282202 281876 281213 281882 278707 282567 284012 28429
1 281424 281512 283618 282723 283928 285186 281431 280567 28064
7 284982 282336 282282 281398 281040 279751 283096 287214 28481
1 282083 282096 282771 281030 279703 281841 283097 278152 28171
4 280661 278310 279842 278205 282110 282604 282488 279851 28363
0 281728 280884 283014 280061 281156 282416 278285 283135 28500
9 281317 280203 279649 284619 281411 281228 278977 280818 27990
9 280381 281751 285171 285001 277576 277251 280972 282224 28142
2 283241 283139 282916 283454 282910 281890 280718 282015 28337
9 279919 281100 281464 281566 281358 279757 282301 280883 28200
3 285531 283849 282399 284421 283997 283538 279525 282921 28132
7 282494 277545 281559 279509 282696 278648 282826 283520 28231
5 283963 280652 281666 281598 278592 279848 283487 285702 27989
3 282953 279622 282766 282329 284402 281513 280449 281805 28206
7 282469 281512 279555 280432 281733 280521 280640 280077 27985
3 284632 281405 278670 283050 282455 280597 285239 281568 28363
9 281972 277871 279463 280447 282462 283667 280198 282727 27953
5 284322 284891 280701 283094 281549 280608 281271 283175 28201
7 280623

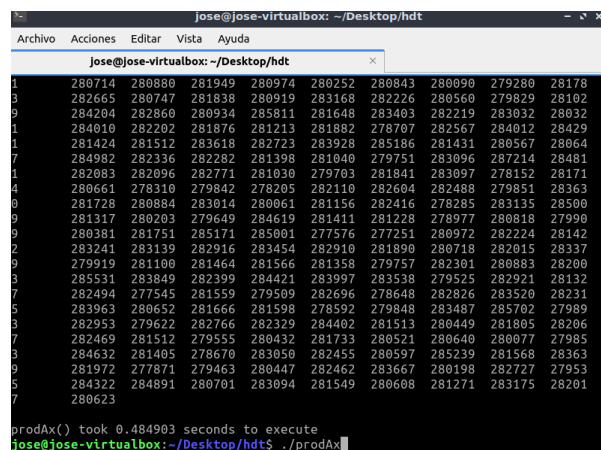
prodAx() took 0.507164 seconds to execute
jose@jose-virtualbox:~/Desktop/hdt$
```

## prodAx\_for\_scope.c

Bloque de código modificado

```
void prodAx(int m, int n, double* restrict A, double* restrict x, double* restrict
b) {
    int i, j;

    #pragma omp parallel for shared(m,n,A,x,b) private(i,j)
    for (i = 0; i < m; i++) {
        double temp = 0.0;
        for (j = 0; j < n; j++) {
            temp += A[i*n + j] * x[j];
        }
        b[i] = temp;
    }
}
```



Mediciones de tiempo

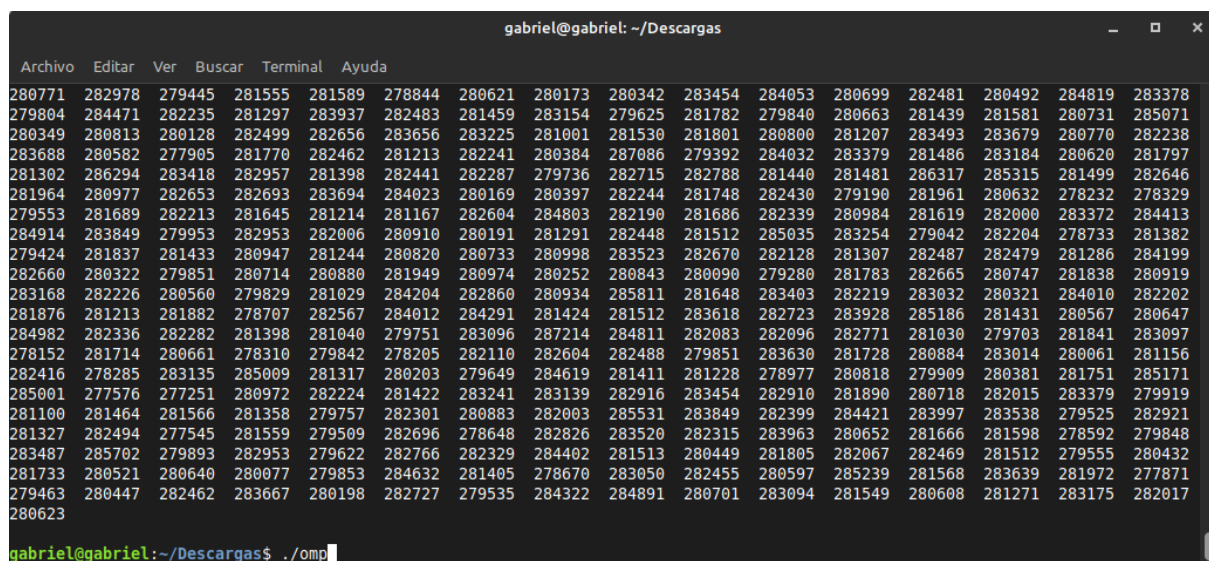
	secuencial	prodAx_paralelo.c	prodAx_for_scope.c
	0,579767	0,507164	0,484903
	0,560366	0,507003	0,494905
	0,542766	0,555774	0,485904
	0,549609	0,497777	0,507567
	0,497053	0,515177	0,495006
<b>promedio</b>	0,545912	0,516579	0,493657
<b>speedup</b>	N/A	1,05678357	1,105853254
<b>eficiencia</b>	N/A	0,2641958926	0,2764633136

Al intentar calcular el producto para una matriz de 30000 x 30000 se produce el siguiente resultado, esto debido a la limitación de la memoria de la computadora.

```
jose@jose-virtualbox:~/Desktop/hdt$ time ./prodAx
Ingrese las dimensiones m y n de la matriz: 30000 30000
memory allocation for A: Cannot allocate memory
Initializing matrix A and vector x
Violación de segmento ('core' generado)

real    0m3.655s
user    0m0.007s
sys     0m0.000s
jose@jose-virtualbox:~/Desktop/hdt$
```

## prodAx\_for\_schedule.c



```
gabriel@gabriel: ~/Descargas
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
280771  282978  279445  281555  281589  278844  280621  280173  280342  283454  284053  280699  282481  280492  284819  283378
279804  284471  282235  281297  283937  282483  281459  283154  279625  281782  279840  280663  281439  281581  280731  285071
280349  280813  280128  282499  282656  283656  283225  281001  281530  281801  280800  281207  283493  283679  280770  282238
283688  280582  277905  281770  282462  281213  282241  280384  287086  279392  284032  283379  281486  283184  280620  281797
281302  286294  283418  282957  281398  282441  282287  279736  282715  282788  281440  281481  286317  285315  281499  282646
281964  280977  282653  282693  283694  284023  280169  280397  282244  281748  282430  279190  281961  280632  278232  278329
279553  281689  282213  281645  281214  281167  282604  284803  282190  281686  282339  280984  281619  282000  283372  284413
284914  283849  279953  282953  282006  280910  280191  281291  282448  281512  285035  283254  279042  282204  278733  281382
279424  281837  281433  280947  281244  280820  280733  280998  283523  282670  282128  281307  282487  282479  281286  284199
282660  280322  279851  280714  280880  281949  280974  280252  280843  280090  279280  281783  282665  280747  281838  280919
283168  282226  280560  279829  281029  284204  282860  280934  285811  281648  283403  282219  283032  280321  284010  282202
281876  281213  281882  278707  282567  284012  284291  281424  281512  283618  282723  283928  285186  281431  280567  280647
284982  282336  282282  281398  281040  279751  283096  287214  284811  282083  282096  282771  281030  279703  281841  283097
278152  281714  280661  278310  279842  278205  282110  282604  282488  279851  283630  281728  280884  283014  280061  281156
282416  278285  283135  285009  281317  280203  279649  284619  281411  281228  278977  280818  279909  280381  281751  285171
285001  277576  277251  280972  282224  281422  283241  283139  282916  283454  282910  281890  280718  282015  283379  279919
281100  281464  281566  281358  279757  282301  280883  282003  285531  283849  282399  284421  283997  283538  279525  282921
281327  282494  277545  281559  279509  282696  278648  282826  283520  282315  283963  280652  281666  281598  278592  279848
283487  285702  279893  282953  279622  282766  282329  284402  281513  280449  281805  282067  282469  281512  279555  280432
281733  280521  280640  280077  279853  284632  281405  278670  283050  282455  280597  285239  281568  283639  281972  277871
279463  280447  282462  283667  280198  282727  279535  284322  284891  280701  283094  281549  280608  281271  283175  282017
280623
gabriel@gabriel:~/Descargas$ ./omp
```

Bloque de código modificado

```
void prodAx(int m, int n, double* A, double* x, double* b) {
    int i, j;

    #pragma omp parallel for shared(m,n,A,x,b) private(i,j)\
    schedule[static,1000]
    for(i = 0; i < m; i++) {
        b[i] = 0.0; //inicialización elemento i del vec.
        for(j = 0; j < n; j++) {
            b[i] += A[i*n + j] * x[j]; //producto punto
        }
    }
    /*---Fin de parallel for---*/
}
```

## Mediciones de tiempo

			Paralelo	
	Secuencial	Static	Dynamic	Guided
10	0,579767			0,84409
100	0,560366			0,83504
1000	0,542766	0,83560	0,82529	0,83971
10000	0,549609	0,40175	0,37755	
100000	0,497053	0,39709	0,37692	
<b>promedio</b>	0,54591	0,54481	0,52659	0,83961
<b>speedup</b>	N/A	1,00202	1,03670	0,65020
<b>eficiencia</b>	N/A	0,12525	0,25917	0,08127

### 3. Planificación y Constructos de Sincronización

```
gabriel@gabriel: ~/Descargas
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
281733  280921  280040  280077  279853  284032  281403  278070  283030  282433  280397  282239  281308  283039  281972  277871
279463  280447  282462  283667  280198  282727  279535  284322  284891  280701  283094  281549  280608  281271  283175  282017
280623

gabriel@gabriel:~/Descargas$ ./omp
Ingrese las dimensiones m y n de la matriz: 10000
10000
Initializing matrix A and vector x
Calculando el producto Ax para m = 10000 n = 10000
prodAx_for_schedule took 0.367018 seconds to execute

b:
283875  282672  283033  277840  285073  281096  281779  277544  282999  280122  281647  280538  280005  282526  285850
280273  282321  279371  279854  279293  283424  281618  280428  285605  283337  281230  283551  283687  279583  280496  282492
279950  281710  282638  281201  284781  278964  280289  282048  282982  282130  279943  280673  281986  282392  281696  284502
286769  282398  284688  282615  282270  279439  285090  282369  280992  281574  283062  279807  283807  279751  282843  283466
282704  280787  282671  280216  278401  279708  282100  281754  284156  284894  282467  281089  283415  279317  280149  283109
283690  280270  283807  281665  282280  281222  281413  277929  284122  279883  284692  279214  281790  285130  281330  280315
280225  284355  280513  283709  282952  281681  283096  283138  280788  282449  277970  282461  278356  279663  284776  280552
284457  281550  282253  280912  279899  280111  281962  283074  281206  280511  282057  283586  281512  283011  283798  280831
281979  281365  278276  283660  280954  283752  279317  281485  279152  282437  282579  282632  283592  282308  283293  283549
279269  283135  283749  282808  282477  280035  279350  279516  282302  279611  282691  282380  284850  281465  283148  280862
283507  283233  280299  279778  281965  282344  279907  281094  284031  280758  283052  279540  281371  285029  281241  279926
282444  278978  280365  281320  281397  281299  280678  282960  279551  281580  284879  281220  283049  280311  281549  280760
```

Bloque de código modificado

```
void prodAx(int m, int n, double* A, double* x, double* b) {
    int i, j;
    omp_lock_t lock; //declarar objeto de mutex

    omp_init_lock(&lock); //inicializar mutex

    #pragma omp parallel for shared(m,n,A,x,b) private(i,j) schedule(dynamic,10000)
    for(i = 0; i < m; i++) {
        double temp = 0.0; //inicialización elemento i del vec.
        for(j = 0; j < n; j++) {
            temp += A[i*n + j] * x[j]; //producto punto
        }
        omp_set_lock(&lock); //adquirir mutex
        b[i] = temp; //actualizar variable compartida
        omp_unset_lock(&lock); //liberar mutex
    }
    omp_destroy_lock(&lock); //destruir mutex
    /*---Fin de parallel for---*/
}
```

Mediciones de tiempo

	secuencial	prodAx_for_schedule_mutex.c
	0,579767	0,374503
	0,560366	0,367617
	0,542766	0,367018
	0,549609	0,366018
	0,497053	0,371018
promedio	0,545912	0,3692348
speedup	N/A	1,478496068



eficiencia	N/A	0,1848120085
------------	-----	--------------