

Problema 1

```
int main()
{
    omp_set_dynamic(0);

    #pragma omp parallel num_threads(2)
    {
        num_hilos(1);
        #pragma omp parallel num_threads(3)
        {
            num_hilos(2);
            #pragma omp parallel num_threads(5)
            {
                num_hilos(3);
            }
        }
    }
    return(0);
}
```

```
void num_hilos(int level)
{
    #pragma omp single
    {
        printf("Nivel %d: # hilos en el team: %d\n", level,
            omp_get_num_threads());
    }
}
```

Problema 1

./p1

Nivel 1: # hilos en el team: 2

Nivel 2: # hilos en el team: 1

Nivel 3: # hilos en el team: 1

Nivel 2: # hilos en el team: 1

Nivel 3: # hilos en el team: 1

Problema 1

./p1

Nivel 1: # hilos en el team: 2

Nivel 2: # hilos en el team: 1

Nivel 3: # hilos en el team: 1

Nivel 2: # hilos en el team: 1

Nivel 3: # hilos en el team: 1

Una sola vez level 1 (con 2 hilos).

Dos veces level 2 (con un hilo).

Dos veces level 3 (con un hilo).

Problema 1_2

```
int main()
{
    omp_set_dynamic(0);
    omp_set_nested(1);
    #pragma omp parallel num_threads(2)
    {
        num_hilos(1);
        #pragma omp parallel num_threads(3)
        {
            num_hilos(2);
            #pragma omp parallel num_threads(5)
            {
                num_hilos(3);
            }
        }
    }
    return(0);
}
```

```
void num_hilos(int level)
{
    #pragma omp single
    {
        printf("Nivel %d: # hilos en el team: %d\n",
            level, omp_get_num_threads());
    }
}
```

Problema 1_2

./p1_2

Nivel 1: # hilos en el team: 2

Nivel 2: # hilos en el team: 3

Nivel 2: # hilos en el team: 3

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Problema 1_2

./p1_2

Nivel 1: # hilos en el team: 2

Nivel 2: # hilos en el team: 3

Nivel 2: # hilos en el team: 3

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Una vez level 1 (con dos hilos).

Dos veces level 2 (con 3 hilos).

Seis veces (2*3) veces level 3 (con 5 hilos)

Problema 1_2

./p1_2

Nivel 1: # hilos en el team: 2

Nivel 2: # hilos en el team: 3

Nivel 2: # hilos en el team: 3

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Nivel 3: # hilos en el team: 5

Una vez level 1 (con dos hilos).

Dos veces level 2 (con 3 hilos).

Seis veces (2*3) veces level 3 (con 5 hilos)

El single afecta a la última región paralela.

Problema 2

```
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);    /* line A */
        else
            omp_set_num_threads(6);    /* line B */
        printf("%d: %d %d\n", omp_get_thread_num(),
            omp_get_num_threads(),
            omp_get_max_threads());
    }
}
```

...

Problema 2

```
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);    /* line A */
        else
            omp_set_num_threads(6);    /* line B */
        printf("%d: %d %d\n", omp_get_thread_num(),
            omp_get_num_threads(),
            omp_get_max_threads());
    }
}
```

Fija el #hilos
de la siguiente región

...

Problema 2

```
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);    /* line A */
        else
            omp_set_num_threads(6);    /* line B */
        printf("%d: %d %d\n", omp_get_thread_num(),
            omp_get_num_threads(),
            omp_get_max_threads());
    }
    ...
}
```

./p2
0: 2 4
1: 2 6
...

Problema 2

```
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);    /* line A */
        else
            omp_set_num_threads(6);    /* line B */
        printf("%d: %d %d\n", omp_get_thread_num(),
            omp_get_num_threads(),
            omp_get_max_threads());
    }
}
```

...

```
...
#pragma omp parallel
{
    #pragma omp master
    {
        printf("Inner: yo: %d de %d\n",
            omp_get_thread_num(), omp_get_num_threads());
    }
    omp_set_num_threads(7);    /* line C */
}
...
```

Problema 2

```
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);    /* line A */
        else
            omp_set_num_threads(6);    /* line B */
        printf("%d: %d %d\n", omp_get_thread_num(),
            omp_get_num_threads(),
            omp_get_max_threads());
    }
}
```

...

```
...
#pragma omp parallel
{
    #pragma omp master
    {
        printf("Inner: yo: %d de %d\n",
            omp_get_thread_num(), omp_get_num_threads());
    }
    omp_set_num_threads(7);    /* line C */
}
...
```

¿Cuántos habrá
ahora?

Problema 2

```
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);    /* line A */
        else
            omp_set_num_threads(6);    /* line B */
        printf("%d: %d %d\n", omp_get_thread_num(),
            omp_get_num_threads(),
            omp_get_max_threads());
    }
    ...
}
```

```
...
#pragma omp parallel
{
    #pragma omp master
    {
        printf("Inner: yo: %d de %d\n",
            omp_get_thread_num(), omp_get_num_threads());
    }
    omp_set_num_threads(7);    /* line C */
}
...
```

```
./p2
0: 2 4
1: 2 6
Inner: yo: 0 de 4
Inner: yo: 0 de 6
...
```

Sin efecto, sólo afectaría
A futuros anidamientos

Problema 2

```
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);    /* line A */
        else
            omp_set_num_threads(6);    /* line B */
        printf("%d: %d %d\n", omp_get_thread_num(),
            omp_get_num_threads(),
            omp_get_max_threads());
    }
}
```

```
...
#pragma omp parallel
{
    #pragma omp master
    {
        printf("Inner: yo: %d de %d\n",
            omp_get_thread_num(), omp_get_num_threads());
    }
    omp_set_num_threads(7);    /* line C */
}
#pragma omp parallel
printf("cuentame.\n");
}
```

¿Cuántas veces?

...

Problema 3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int t = (0 == 0); // true value
    int f = (1 == 0); // false value
    #pragma omp parallel if (f) num_threads(4)
    {
        printf ("FALSE: Soy el thread %d\n", omp_get_thread_num());
    }
    #pragma omp parallel if (t) num_threads(4)
    {
        printf ("TRUE : Soy el thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```

Problema 3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int t = (0 == 0); // true value
    int f = (1 == 0); // false value
    #pragma omp parallel if (f) num_threads(4)
    {
        printf ("FALSE: Soy el thread %d\n", omp_get_thread_num());
    }
    #pragma omp parallel if (t) num_threads(4)
    {
        printf ("TRUE : Soy el thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```



¿Cuántas veces?

Problema 3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int t = (0 == 0); // true value
    int f = (1 == 0); // false value
    #pragma omp parallel if (f) num_threads(4)
    {
        printf ("FALSE: Soy el thread %d\n", omp_get_thread_num());
    }
    #pragma omp parallel if (t) num_threads(4)
    {
        printf ("TRUE : Soy el thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```

¿Cuántas veces?

¿Y ésta?

Problema 3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int t = (0 == 0); // true value
    int f = (1 == 0); // false value
    #pragma omp parallel if (f) num_threads(4)
    {
        printf ("FALSE: Soy el thread %d\n", omp_get_thread_num());
    }
    #pragma omp parallel if (t) num_threads(4)
    {
        printf ("TRUE : Soy el thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```

./p3

TRUE : Soy el thread 0
TRUE : Soy el thread 2
TRUE : Soy el thread 1
TRUE : Soy el thread 3

Problema 3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int t = (0 == 0); // true value
    int f = (1 == 0); // false value
    #pragma omp parallel if (f) num_threads(4)
    {
        printf ("FALSE: Soy el thread %d\n", omp_get_thread_num());
    }
    #pragma omp parallel if (t) num_threads(4)
    {
        printf ("TRUE : Soy el thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```

./p3

FALSE: Soy el thread 0

TRUE : Soy el thread 0

TRUE : Soy el thread 2

TRUE : Soy el thread 1

TRUE : Soy el thread 3

Problema 4

```
#include <math.h>
void nowait_example2(int n, float *a, float *b, float *c, float *y, float *z)
{
    int i;
    #pragma omp parallel
    {
        #pragma omp for schedule(static) nowait
        for (i=0; i<n; i++)
            c[i] = (a[i] + b[i]) / 2.0f;

        #pragma omp for schedule(static) nowait
        for (i=0; i<n; i++)
            z[i] = sqrtf(c[i]);

        #pragma omp for schedule(static) nowait
        for (i=1; i<=n; i++)
            y[i] = z[i-1] + a[i];
    }
}
```

Problema 5

```
#pragma omp parallel for num_threads(4) private(j) collapse(2)
for (i = 0; i < 2; i++)
    for (j = 0; j <= i; j++)
        printf("%d %d %d\n", i, j, omp_get_thread_num());
```

Problema 5

```
#pragma omp parallel for num_threads(4) private(j) collapse(2)
for (i = 0; i < 2; i++)
    for (j = 0; j <= i; j++)
        printf("%d %d %d\n", i, j, omp_get_thread_num());
```

```
gcc -o p5 p5.c -fopenmp
```

```
p5.c: In function 'main':
```

```
p5.c:8:17: error: condition expression refers to iteration variable 'i'
```


```
8 |     for (j = 0; j <= i; j++)
  |                   ^
```

Problema 5_2

```
int main (void)
{
    int i,j;
    #pragma omp parallel for num_threads(4) private(j)
    for (i = 0; i < 2; i++)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    return 0;
}
```

Problema 5_2

```
int main (void)
{
    int i,j;
    #pragma omp parallel for num_threads(4) private(j)
    for (i = 0; i < 2; i++)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    return 0;
}
```



!!!OJO!!!

Problema 5_2

```
int main (void)
{
    int i,j;
    #pragma omp parallel for num_threads(4) private(j)
    for (i = 0; i < 2; i++)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    return 0;
}
```

./p5_2
1 0 1
1 1 1
1 2 1
1 3 1
1 4 1
0 0 0
0 1 0
0 2 0
0 3 0
0 4 0

Problema 5_2

```
int main (void)
{
    int i,j;
    #pragma omp parallel for num_threads(4) private(j)
    for (i = 0; i < 2; i++)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    return 0;
}
```

./p5_2

1 0 **1**

1 1 **1**

1 2 **1**

1 3 **1**

1 4 **1**

0 0 **0**

0 1 **0**

0 2 **0**

0 3 **0**

0 4 **0**

Sólo dos hilos corriendo

Problema 5_3

```
int main (void)
{
    int i,j;
    #pragma omp parallel for num_threads(4) private(j) collapse(2)
    for (i = 0; i < 2; i++)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    return 0;
}
```

Problema 5_3

```
int main (void)
{
    int i,j;
    #pragma omp parallel for num_threads(4) private(j) collapse(2)
    for (i = 0; i < 2; i++)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    return 0;
}
```

```
./p5_3
0 3 1
0 4 1
1 0 1
1 3 3
1 4 3
0 0 0
0 1 0
0 2 0
1 1 2
1 2 2
```

Problema 5_3

```
int main (void)
{
    int i,j;
    #pragma omp parallel for num_threads(4) private(j) collapse(2)
    for (i = 0; i < 2; i++)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    return 0;
}
```

```
./p5_3
0 3 1
0 4 1
0 4 1
1 0 1
1 3 3
1 4 3
0 0 0
0 1 0
0 2 0
1 1 2
1 2 2
```

4 hilos corriendo

Problema 5_4

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j;

    #pragma omp parallel for num_threads(2) private(j)
    for (i = 0; i < 2; i++){

        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());

    }
    return 0;
}
```

Problema 5_4

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j;

    #pragma omp parallel for num_threads(2) private(j)
    for (i = 0; i < 2; i++){

        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());

    }
    return 0;
}
```

./p5_4

0 0 0

0 1 0

0 2 0

0 3 0

0 4 0

1 0 0

1 1 0

1 2 0

1 3 0

1 4 0

¿¡¡Sólo un hilo!!?

Problema 5_4

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j;

    #pragma omp parallel for num_threads(2) private(j)
    for (i = 0; i < 2; i++){

        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d\n", i, j, omp_get_thread_num());
    }
    return 0;
}
```

./p5_4

0 0 0

0 1 0

0 2 0

0 3 0

0 4 0

1 0 0

1 1 0

1 2 0

1 3 0

1 4 0

¿¡¡Sólo un hilo!!?

**¿O sólo un hilo en el
parallel interior?**

Problema 5_4_2

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j,yo;

    #pragma omp parallel for num_threads(2) private(j, yo)
    for (i = 0; i < 2; i++){
        yo=omp_get_thread_num();
        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d %d\n", i, j, yo, omp_get_thread_num());
    }
    return 0;
}
```

Problema 5_4_2

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j,yo;

    #pragma omp parallel for num_threads(2) private(j, yo)
    for (i = 0; i < 2; i++){
        yo=omp_get_thread_num();
        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d %d\n", i, j, yo, omp_get_thread_num());
    }
    return 0;
}
```

./p5_4_2

0 0 **0** 0

0 1 **0** 0

0 2 **0** 0

0 3 **0** 0

0 4 **0** 0

1 0 **1** 0

1 1 **1** 0

1 2 **1** 0

1 3 **1** 0

1 4 **1** 0

Ahhh, había dos...

Pero... ¿solo dos???

Problema 5_4_3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j,yo;
    omp_set_nested(1);
    #pragma omp parallel for num_threads(2) private(j, yo)
    for (i = 0; i < 2; i++){
        yo=omp_get_thread_num();
        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d %d\n", i, j, yo, omp_get_thread_num());
    }
    return 0;
}
```

Problema 5_4_3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j,yo;
    omp_set_nested(1);
    #pragma omp parallel for num_threads(2) private(j, yo)
    for (i = 0; i < 2; i++){
        yo=omp_get_thread_num();
        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d %d\n", i, j, yo, omp_get_thread_num());
    }
    return 0;
}
```

./p5_4_3

0 0 0 0

0 1 0 0

0 2 0 0

0 3 0 1

0 4 0 1

1 0 1 0

1 1 1 0

1 2 1 0

1 3 1 1

1 4 1 1

Problema 5_4_3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j,yo;
    omp_set_nested(1);
    #pragma omp parallel for num_threads(2) private(j, yo)
    for (i = 0; i < 2; i++){
        yo=omp_get_thread_num();
        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d %d\n", i, j, yo, omp_get_thread_num());
    }
    return 0;
}
```

./p5_4_3

0 0 **0 0**

0 1 **0 0**

0 2 **0 0**

0 3 **0 1**

0 4 **0 1**

1 0 **1 0**

1 1 **1 0**

1 2 **1 0**

1 3 **1 1**

1 4 **1 1**

¿Cuántos hay ahora?

Problema 5_4_3

```
#include <omp.h>
#include <stdio.h>
int main (void)
{
    int i,j,yo;
    omp_set_nested(1);
    #pragma omp parallel for num_threads(2) private(j, yo)
    for (i = 0; i < 2; i++){
        yo=omp_get_thread_num();
        #pragma omp parallel for num_threads(2)
        for (j = 0; j < 5; j++)
            printf("%d %d %d %d\n", i, j, yo, omp_get_thread_num());
    }
    return 0;
}
```

./p5_4_3

0 0 **0 0**

0 1 **0 0**

0 2 **0 0**

0 3 **0 1**

0 4 **0 1**

1 0 **1 0**

1 1 **1 0**

1 2 **1 0**

1 3 **1 1**

1 4 **1 1**

**¿Mejor o peor que
collapse?**

Problema 5_5

```
#include <stdio.h>
#include <omp.h>
int main (void){
    #pragma omp parallel if(1==0)
    {
        printf ("hilo OUT %d\n", omp_get_thread_num());
        #pragma omp parallel
        printf ("hilo IN %d\n", omp_get_thread_num());
    }
    return 1;
}
```

Problema 5_5

```
#include <stdio.h>
#include <omp.h>
int main (void){
    #pragma omp parallel if(1==0)
    {
        printf ("hilo OUT %d\n", omp_get_thread_num());
        #pragma omp parallel
        printf ("hilo IN %d\n", omp_get_thread_num());
    }
    return 1;
}
```



¿Qué hará?

Problema 5_5

```
#include <stdio.h>
#include <omp.h>
int main (void){
    #pragma omp parallel if(1==0)
    {
        printf ("hilo OUT %d\n", omp_get_thread_num());
        #pragma omp parallel
        printf ("hilo IN %d\n", omp_get_thread_num());
    }
    return 1;
}
```

```
./nested
hilo OUT 0
hilo IN 0
hilo IN 2
hilo IN 1
hilo IN 3
```

Problema 6

```
#pragma omp parallel shared(A) private(B)
{
    #pragma omp task
    {
        int C;
        compute(A, B, C);
    }
}
```

¿Visibilidad de cada variable?

Problema 6

¿Visibilidad de cada variable?

```
#pragma omp parallel shared(A) private(B)
{
    #pragma omp task
    {
        int C;
        compute(A, B, C);
    }
}
```



shared

Problema 6

¿Visibilidad de cada variable?

```
#pragma omp parallel shared(A) private(B)
{
    #pragma omp task
    {
        int C;
        compute(A, B, C);
    }
}
```

shared

firstprivate

Problema 6

¿Visibilidad de cada variable?

```
#pragma omp parallel shared(A) private(B)
{
    #pragma omp task
    {
        int C;
        compute(A, B, C);
    }
}
```

shared

private

firstprivate

Problema 7

¿Cómo paralelizar?

```
#include <stdio.h>
#define N 1000000000
int main(int argc, char** argv){
    int total_Sum;

    total_Sum = 0;

    for(int i = 1; i <= N; i++)
        total_Sum += i;

    printf("Total Sum: %d\n", total_Sum);
    return 0;
}
```

Problema 7_1

```
#include <stdio.h>
#define N 1000000000
int main(int argc, char** argv){
    double total_Sum;
    #pragma omp parallel
    {
        #pragma omp single
        total_Sum = 0;
        #pragma omp for reduction(+:total_Sum)
        for(int i = 1; i <= N; i++)
            total_Sum += i;
    }
    printf("Total Sum: %f\n", total_Sum);
    return 0;
}
```

```
./p6_1
Tiempo Reduction:
6.7820787E-01 s
O...
1.0722280E+00 s
```

Problema 7_2

```
#include <stdio.h>
#define N 1000000000
int main(int argc, char** argv){
    double partial_Sum, total_Sum;
    #pragma omp parallel private(partial_Sum)
    {
        partial_Sum = 0;
        #pragma omp single
        total_Sum = 0;
        #pragma omp for
        for(int i = 1; i <= N; i++)
            partial_Sum += i;
        #pragma omp critical
        total_Sum += partial_Sum;
    }
    printf("Total Sum: %\n", total_Sum);
    return 0;
}
```

./p6_2
Tiempo Critical:
6.0535502E-01 s
O...
1.0161831E+00 s
**¿Mejor o peor que
reduction?**

Problema 7_3

```
#include <stdio.h>
#define N 1000000000
int main(int argc, char** argv){
    double partial_Sum, total_Sum;
    #pragma omp parallel private(partial_Sum)
    {
        partial_Sum = 0;
        #pragma omp single
        total_Sum = 0;
        #pragma omp for
        for(int i = 1; i <= N; i++)
            partial_Sum += i;
        #pragma omp atomic update
        total_Sum += partial_Sum;
    }
    printf("Total Sum: %f\n", total_Sum);
    return 0;
}
```

./p6_3
Tiempo Atomic:
5.7236290E-01 s
O...
1.0028360E+00 s
**¿Mejor o peor que
reduction y critical?**

Problema 7_3

```
#include <stdio.h>
#define N 1000000000
int main(int argc, char** argv){
    double partial_Sum, total_Sum;
    #pragma omp parallel private(partial_Sum)
    {
        partial_Sum = 0;
        #pragma omp single
        total_Sum = 0;
        #pragma omp for
        for(int i = 1; i <= N; i++)
            partial_Sum += i;
        #pragma omp atomic update
        total_Sum += partial_Sum;
    }
    printf("Total Sum: %f\n", total_Sum);
    return 0;
}
```

¿Por qué no hay tanta mejora?

#hilos en conflicto...
Conflictos en critical...
¿Luce atomic?

Problema 7_3

```
#include <stdio.h>
#define N 1000000000
int main(int argc, char** argv){
    double partial_Sum, total_Sum;
    #pragma omp parallel private(partial_Sum)
    {
        partial_Sum = 0;
        #pragma omp single
        total_Sum = 0;
        #pragma omp for
        for(int i = 1; i <= N; i++)
            partial_Sum += i;
        #pragma omp atomic update
        total_Sum += partial_Sum;
    }
    printf("Total Sum: %f\n", total_Sum);
    return 0;
}
```

En general:

Critical, puede contener cualquier cosa. Coste caro en gestión. Si además no tienen nombre, bloquea TODO (ningún hilo puede entrar en ninguna otra región crítica: un candado para todas ellas).

Atomic usa recursos HW específicos. No hay candados. No bloquea otras operaciones atómicas. Sólo soporta algunas operaciones.

Problema 8

```
omp_set_nested(1);
int i = 1;
#pragma omp parallel sections
{
    #pragma omp section
    #pragma omp critical (name)
    #pragma omp parallel
    #pragma omp single
    i++;
}
printf("i=%i\n",i);
```

Problema 8

```
omp_set_nested(1);
int i = 1;
#pragma omp parallel sections
{
    #pragma omp section
    #pragma omp critical (name)
    #pragma omp parallel //Sin este parallel, no compila
    #pragma omp single
    i++;
}
printf("i=%i\n",i);
```

¿Para qué sirve “name”?

Ojo al parallel interior

¿Qué valor tiene la *i* al final?

Problema 8

```
omp_set_nested(1);
int i = 1;
#pragma omp parallel sections
{
    #pragma omp section
    #pragma omp critical (name)
    #pragma omp parallel //Sin este parallel, no compila
    #pragma omp single
    i++;
}
printf("i=%i\n",i);
```

./ ...
i=2

Problema 9

```
#include <omp.h>
#include <stdio.h>
#define NT 4

int main() {
    int section_count = 1;
    omp_set_dynamic(0);
    omp_set_num_threads(NT);
    #pragma omp parallel
        #pragma omp sections firstprivate(section_count)
        {
            [#pragma omp section]
            {
                section_count++;
                printf("S1: section_count %d\n", section_count);
            }
            #pragma omp section
            {
                section_count++;
                printf("S2: section_count %d\n", section_count);
            }
        }
    return 1;
}
```

**¿Qué vemos por
pantalla?**

Problema 9

```
#include <omp.h>
#include <stdio.h>
#define NT 4

int main() {
    int section_count = 1;
    omp_set_dynamic(0);
    omp_set_num_threads(NT);
    #pragma omp parallel
        #pragma omp sections firstprivate(section_count)
        {
            [#pragma omp section]
            {
                section_count++;
                printf("S1: section_count %d\n", section_count);
            }
            #pragma omp section
            {
                section_count++;
                printf("S2: section_count %d\n", section_count);
            }
        }
    return 1;
}
```

./ ...

S1: section_count 2

S2: section_count 2

Problema 10

```
int func1(){  
    printf("soy el hilo %d en func1\n",omp_get_thread_num());  
    return 1;  
}  
int func2(){  
    printf("soy el hilo %d en func2\n",omp_get_thread_num());  
    return 2;  
}  
void func3(int a, int b){  
    printf("soy el hilo %d en func3 y suma=%d\n",omp_get_thread_num(),a+b);  
}  
  
void func(){  
    int a,b;  
    a=func1();  
    b=func2();  
    func3(a,b);  
}
```

Paralelizar

Problema 10

```
void func(){
```

Paralelizar

```
    int a,b;
```

```
    #pragma omp parallel
```

```
    {
```

```
        a=func1();
```

```
        b=func2();
```

```
        func3(a,b);
```

```
    }
```

```
}
```

Problema 10

```
void func(){
```

Paralelizar

```
    int a,b;
```

```
    #pragma omp parallel
```

```
    {
```

```
        #pragma omp single/master
```

```
        {
```

```
            a=func1();
```

```
            b=func2();
```

```
            func3(a,b);
```

```
        }
```

```
    }
```

```
}
```

Problema 10

```
void func(){
```

```
    int a,b;
```

```
    #pragma omp parallel
```

```
    {
```

```
        #pragma omp single/master
```

```
        {
```

```
            #pragma omp task
```

```
            a=func1();
```

```
            #pragma omp task
```

```
            b=func2();
```

```
            #pragma omp task
```

```
            func3(a,b);
```

```
        }
```

```
    }
```

```
}
```

Paralelizar

¿Funciona?

Problema 10

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp single/master
        {
            #pragma omp task
            a=func1();

            #pragma omp task
            b=func2();

            #pragma omp task
            func3(a,b);
        }
    }
}
```

Paralelizar

¿Funciona?

./...

soy el hilo 0 en func3 y suma=0

soy el hilo 1 en func2

soy el hilo 2 en func1

Problema 10

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp single/master
        {
            #pragma omp task
            a=func1();

            #pragma omp task
            b=func2();
            #pragma omp taskwait
            [#pragma omp task ]
            func3(a,b);
        }
    }
}
```

Paralelizar

./ ...
soy el hilo 1 en func2
soy el hilo 2 en func1
soy el hilo 0 en func3 y suma=3

Problema 10_2

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp single/master
        {
            #pragma omp task depend(out:a)
            a=func1();

            #pragma omp task depend(out:b)
            b=func2();
            #pragma omp task depend(in:a,b)
            func3(a,b);
        }
    }
}
```

Paralelizar

Teóricamente debería funcionar igual...

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            a=func1();

            #pragma omp section
            b=func2();
            #pragma omp barrier
            #pragma omp section
            func3(a,b);
        }
    }
}
```

Paralelizar

¿Funcionará?

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            [#pragma omp section]
            a=func1();

            #pragma omp section
            b=func2();
            #pragma omp barrier
            #pragma omp section
            func3(a,b);
        }
    }
}
```

Paralelizar

¿Funcionará?

error: '#pragma omp barrier' may only be used in compound statements before '#pragma'

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            [#pragma omp section]
            a=func1();

            #pragma omp section
            b=func2();
        }
        [#pragma omp barrier]
        func3(a,b);
    }
}
```

Paralelizar

¿Funcionará?

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            [#pragma omp section]
            a=func1();

            #pragma omp section
            b=func2();
        }
        [#pragma omp barrier]
        func3(a,b);
    }
}
```

Paralelizar

¿Funcionará?

./ ...

soy el hilo 2 en func1

soy el hilo 0 en func2

soy el hilo 2 en func3 y suma=3

soy el hilo 3 en func3 y suma=3

soy el hilo 0 en func3 y suma=3

soy el hilo 1 en func3 y suma=3

¿Arreglos?

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            [#pragma omp section]
            a=func1();

            #pragma omp section
            b=func2();
        }
        [#pragma omp barrier]
        func3(a,b);
    }
}
```

Paralelizar

¿Funcionará?

./ ...

soy el hilo 2 en func1

soy el hilo 0 en func2

soy el hilo 2 en func3 y suma=3

soy el hilo 3 en func3 y suma=3

soy el hilo 0 en func3 y suma=3

soy el hilo 1 en func3 y suma=3

¿Arreglos?

Single/master o mover

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            [#pragma omp section]
            a=func1();

            #pragma omp section
            b=func2();
        }
        [#pragma omp barrier]
        #pragma omp single/master
        func3(a,b);
    }
}
```

Paralelizar

¿Funcionará?

Ahora sí:

./ ...

soy el hilo 2 en func1

soy el hilo 0 en func2

soy el hilo 2 en func3 y suma=3

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            [#pragma omp section]
            a=func1();

            #pragma omp section
            b=func2();

        }

    }
    func3(a,b);
}
```

Paralelizar

¿Funcionará?

Problema 10_3

```
void func(){
    int a,b;
    #pragma omp parallel
    {
        #pragma omp sections
        {
            [#pragma omp section]
            a=func1();

            #pragma omp section
            b=func2();

        }

    }
    func3(a,b);
}
```

Paralelizar

¿Funcionará?

Ahora también:

./ ...

soy el hilo 3 en func1

soy el hilo 2 en func2

soy el hilo 0 en func3 y suma=3

Problema 11

```
#include <stdio.h>
#include <omp.h>
```

¿tasks sin single?

```
void print(int i){
    printf("soy el hilo %d con i=%d\n",omp_get_thread_num(),i);
}
```

```
int main(){
    int i;
    #pragma omp parallel for
    for(i = 0; i <10; i++){
        #pragma omp task
        print(i);
    }
    return 1;
}
```


Problema 11

```
#include <stdio.h>
#include <omp.h>

void print(int i){
    printf("soy el hilo %d con i=%d\n",omp_get_thread_num(),i);
}

int main(){
    int i;
    #pragma omp parallel for
    for(i = 0; i <10; i++){
        #pragma omp task
        print(i);
    }
    return 1;
}
```

¿tasks sin single?

```
./ ...
soy el hilo 3 con i=6
soy el hilo 2 con i=1
soy el hilo 2 con i=7
soy el hilo 2 con i=4
soy el hilo 2 con i=8
soy el hilo 2 con i=5
soy el hilo 2 con i=9
soy el hilo 3 con i=3
soy el hilo 0 con i=0
soy el hilo 1 con i=2
```

Problema 12

```
void func()
{
    #pragma omp parallel
    {
        #pragma omp critical
        {
            func1();
            #pragma omp barrier
            func2();
        }
    }
}
```

¿Funciona?

Problema 12_2

```
void func()
{
    #pragma omp parallel
    {
        #pragma omp single
        {
            func1();
            #pragma omp barrier
            func2();
        }
    }
}
```

¿Funciona?

Problema 13

```
#include <stdio.h>
```

¿Qué hace?

```
float x, y;
```

```
#pragma omp threadprivate(x, y)
```

```
int main( )
```

```
{
```

```
    float a;
```

```
    float b;
```

```
    #pragma omp parallel private(a,b)
```

```
    {
```

```
        #pragma omp single copyprivate(a,b,x,y)
```

```
        scanf("%f %f %f %f",&a,&b,&x,&y);
```

```
        printf("%f %f %f %f\n",a, b, x, y);
```

```
    }
```

```
    return 0;
```

```
}
```

Problema 13

```
#include <stdio.h>
```

```
float x, y;
```

```
#pragma omp threadprivate(x, y)
```

```
int main( )
```

```
{
```

```
    float a;
```

```
    float b;
```

```
    #pragma omp parallel private(a,b)
```

```
    {
```

```
        #pragma omp single copyprivate(a,b,x,y)
```

```
        scanf("%f %f %f %f",&a,&b,&x,&y);
```

```
        printf("%f %f %f %f\n",a, b, x, y);
```

```
    }
```

```
    return 0;
```

```
}
```

¿Qué hace?

Sin esto no deja compilar

Problema 13

```
#include <stdio.h>
float x, y;
#pragma omp threadprivate(x, y)
int main( )
{
    float a;
    float b;
    #pragma omp parallel private(a,b)
    {
        #pragma omp single copyprivate(a,b,x,y)
        scanf("%f %f %f %f",&a,&b,&x,&y);
        printf("%f %f %f %f\n",a, b, x, y);
    }
    return 0;
}
```

¿Qué hace?

./ ...

1 2 3 4

1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000

Problema 13

```
#include <stdio.h>
float x, y;
#pragma omp threadprivate(x, y)
int main( )
{
    float a;
    float b;
    #pragma omp parallel private(a,b)
    {
        #pragma omp single copyprivate(a,b,x,y)
        scanf("%f %f %f %f",&a,&b,&x,&y);
        printf("%f %f %f %f\n",a, b, x, y);
    }
    return 0;
}
```

¿Qué hace?

./ ...

1 2 3 4

1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000

**Copyprivate como difusor de
información entre hilos (¡¡a sus
variables privadas!!)**

Problema 13_2

```
#include <stdio.h>
float x, y;
#pragma omp threadprivate(x, y)
int main( )
{
    float a;
    float b;
    #pragma omp parallel private(a,b)
    {
        #pragma omp master copyprivate(a,b,x,y)
        scanf("%f %f %f %f",&a,&b,&x,&y);
        printf("%f %f %f %f\n",a, b, x, y);
    }
    return 0;
}
```

PERO...
¿Qué hace?

Problema 13_2

```
#include <stdio.h>
float x, y;
#pragma omp threadprivate(x, y)
int main( )
{
    float a;
    float b;
    #pragma omp parallel private(a,b)
    {
        #pragma omp master copyprivate(a,b,x,y)
        scanf("%f %f %f %f",&a,&b,&x,&y);
        printf("%f %f %f %f\n",a, b, x, y);
    }
    return 0;
}
```

PERO...

¿Qué hace?

NO COMPILA

Problema 13_2

```
int main() {
    float a, b;
    float *tmp1,*tmp2;
    #pragma omp parallel private(tmp1,tmp2)
    {
        #pragma omp single copyprivate(tmp1,tmp2)
        {
            tmp1=(float*)malloc(sizeof(float));
            tmp2=(float*)malloc(sizeof(float));
        }
        #pragma omp master
        {
            scanf("%f %f",tmp1,tmp2);
            a=*tmp1; b=*tmp2;
        }
        #pragma omp barrier
        printf("%f %f\n",a,b);
        #pragma omp single
        {
            free(tmp1); free(tmp2);
        }
    }
}
```

Truco: variables temporales.
¿Qué hace?

Problema 13_2

```
int main() {
    float a, b;
    float *tmp1,*tmp2;
    #pragma omp parallel private(tmp1,tmp2)
    {
        #pragma omp single copyprivate(tmp1,tmp2)
        {
            tmp1=(float*)malloc(sizeof(float));
            tmp2=(float*)malloc(sizeof(float));
        }
        #pragma omp master
        {
            scanf("%f %f",tmp1,tmp2);
            a=*tmp1; b=*tmp2;
        }
        #pragma omp barrier
        printf("%f %f\n",a,b);
        #pragma omp single
        {
            free(tmp1); free(tmp2);
        }
    }
}
```

Truco: variables temporales.
¿Qué hace?

```
./ ..
1 2
1.000000 2.000000
1.000000 2.000000
1.000000 2.000000
1.000000 2.000000
```

Problema 13_3

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int a = 12345;
#pragma omp threadprivate(a)
int main(){
    omp_set_dynamic(0);
    #pragma omp parallel copyin(a)
    {
        #pragma omp master
        a = 67890;
        #pragma omp barrier
        printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    }
    #pragma omp parallel copyin(a)
    printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    return 1;
}
```

Otra opción: **copyin**
¿Qué hace?

Problema 13_3

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int a = 12345;
#pragma omp threadprivate(a)
int main(){
    omp_set_dynamic(0);
    #pragma omp parallel [copyin(a)]
    {
        #pragma omp master
        a = 67890;
        #pragma omp barrier
        printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    }
    #pragma omp parallel copyin(a)
    printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    return 1;
}
```

Este no hace nada,
pero...

Otra opción: **copyin**
¿Qué hace?

```
./ ...
Hilo 0: a = 67890.
Hilo 2: a = 12345.
Hilo 1: a = 12345.
Hilo 3: a = 12345.
Hilo 2: a = 67890.
Hilo 3: a = 67890.
Hilo 1: a = 67890.
Hilo 0: a = 67890.
```

Problema 13_4

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int a = 12345;
#pragma omp threadprivate(a)
int main(){
    omp_set_dynamic(0);
    a=0;
    #pragma omp parallel copyin(a)
    {
        #pragma omp master
        a = 67890;
        #pragma omp barrier
        printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    }
    #pragma omp parallel copyin(a)
    printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    return 1;
}
```

Otra opción: **copyin**
¿Qué hace?

./ ...

Problema 13_4

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int a = 12345;
#pragma omp threadprivate(a)
int main(){
    omp_set_dynamic(0);
    a=0;
    #pragma omp parallel copyin(a)
    {
        #pragma omp master
        a = 67890;
        #pragma omp barrier
        printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    }
    #pragma omp parallel copyin(a)
    printf("Hilo %d: a = %d.\n", omp_get_thread_num(), a);
    return 1;
}
```

Otra opción: **copyin**
¿Qué hace?

```
./ ...
Hilo 0: a = 67890.
Hilo 3: a = 0.
Hilo 1: a = 0.
Hilo 2: a = 0.
Hilo 2: a = 67890.
Hilo 3: a = 67890.
Hilo 0: a = 67890.
Hilo 1: a = 67890.
```

Problema 14

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

omp_lock_t *new_lock(){
    omp_lock_t *lock_ptr;
    #pragma omp single copyprivate(lock_ptr)
    {
        lock_ptr = (omp_lock_t *) malloc(sizeof(omp_lock_t));
        omp_init_lock( lock_ptr );
    }
    return lock_ptr;
}
```

Aún más complicado (y útil):
Cuando el número de cerrojos no es conocido a priori (llamada a new_lock desde región paralela -p.e. en un parallel for-), pero necesitamos que sean **compartidos** (si no, vaya m... de cerrojos).

Problema 15

```
int main(){  
    double *A, sum;  
    A = (double *)malloc(N*sizeof(double));  
    fill_rand(N, A);  
    sum = Sum_array(N, A);  
}
```

¿Cómo se llama este algoritmo?

Problema 15

```
int main(){
    double *A, sum;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
        }
        #pragma omp section
        {
            sum = Sum_array(N, A);
        }
    }
}
```

¿Cómo se llama este algoritmo???

Problema 15

```
int main(){
    double *A, sum;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
        }
        #pragma omp section
        {
            sum = Sum_array(N, A);
        }
    }
}
```

¡¡Consumidor-Productor!!
¿Hay condiciones de carrera?

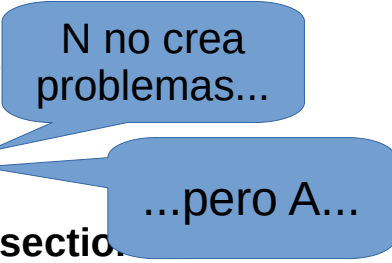
Problema 15

```
int main(){
    double *A, sum;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
        }
        #pragma omp section
        {
            sum = Sum_array(N, A);
        }
    }
}
```

¡¡Consumidor-Productor!!
¿Hay condiciones de carrera?
En otras palabras: ¿hay
escrituras y lecturas
simultáneas sobre una misma
dirección?

Problema 15

```
int main(){
    double *A, sum;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp
        {
            fill_rand(N, A);
        }
        #pragma omp section
        {
            sum = Sum_array(N, A);
        }
    }
}
```



¡¡Consumidor-Productor!!
¿Hay condiciones de carrera?
Sí, sobre el array A

¿Cómo se soluciona?

Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

Espera activa!!

Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

¿Hay condiciones de carrera?

Problema 15

*flag es
compartida*

```
int main(){
    double *A, sum,
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

**¿Hay condiciones de carrera?
Sí (esto es el día de la
marmota)**



Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            #pragma atomic write
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                #pragma atomic read
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

¿Hay condiciones de carrera?

Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            #pragma atomic write
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                #pragma atomic read
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

¿Hay condiciones de carrera?
!!!NO!!!

(pero...)



Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            #pragma atomic write
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                #pragma atomic read
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

¿Hay condiciones de carrera?
!!!NO!!!

(pero...
¿y la coherencia?)



Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            #pragma atomic write
            flag = 1;
        }
        #pragma omp section
        {
            while (1){
                #pragma atomic read
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

Avisar de que
actualice
memoria

¿Hay condiciones de carrera?
!!!NO!!!

Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            #pragma atomic write
            flag = 1;
            #pragma omp flush(flag)
        }
        #pragma omp section
        {
            while (1){
                #pragma atomic read
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

¿Hay condiciones de carrera?
!!!NO!!!

¿Más problemas de
coherencia?

Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            #pragma atomic write
            flag = 1;
            #pragma omp flush(flag)
        }
        #pragma omp section
        {
            while (1){
                #pragma atomic read
                if (flag==1) break;
            }
            sum = Sum_array(N, A);
        }
    }
}
```

¿Y si al leer
aquí no estaba
actualizada?

¿Hay condiciones de carrera?
!!!NO!!!

¿Más problemas de
coherencia?

Problema 15

```
int main(){
    double *A, sum;
    int flag = 0;
    A = (double *)malloc(N*sizeof(double));
    #pragma omp parallel sections
    {
        [#pragma omp section]
        {
            fill_rand(N, A);
            #pragma atomic write
            flag = 1;
            #pragma omp flush(flag)
        }
        #pragma omp section
        {
            while (1){
                #pragma omp flush(flag)
                #pragma atomic read
                if (flag==1) break;
            } ...
        }
    }
}
```

¿Hay condiciones de carrera?
¡¡¡NO!!!

¿Más problemas de coherencia?

¡¡¡Ya no!!!

Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi;
    double sum;
    step = 1.0/(double) num_steps;

    int i;
    double x;

    for (i=0, sum=0.0; i< num_steps; i++) {
        x = (i+0.5)*step;
        sum += 4.0/(1.0+x*x);
    }

    pi = sum * step;
}
```

Resulta que... $\int_0^1 \frac{4.0}{(1+x^2)} = \pi$

**Algoritmo para calcular π
Paralelizar a mano (DyV)**

Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id]=0.0;i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0;i<nthreads;i++) pi += sum[i] * step;
}
```

Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i] * step;
}
```

Hilos:1 3.9778399E-01 s
Hilos:2 5.2525401E-01 s
Hilos:3 5.2193189E-01 s
Hilos:4 4.0789413E-01 s



Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i] * step;
}
```

Hilos:1 3.9778399E-01 s
Hilos:2 5.2525401E-01 s
Hilos:3 5.2193189E-01 s
Hilos:4 4.0789413E-01 s



¿Qué f*ck está pasando???

Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i] * step;
}
```

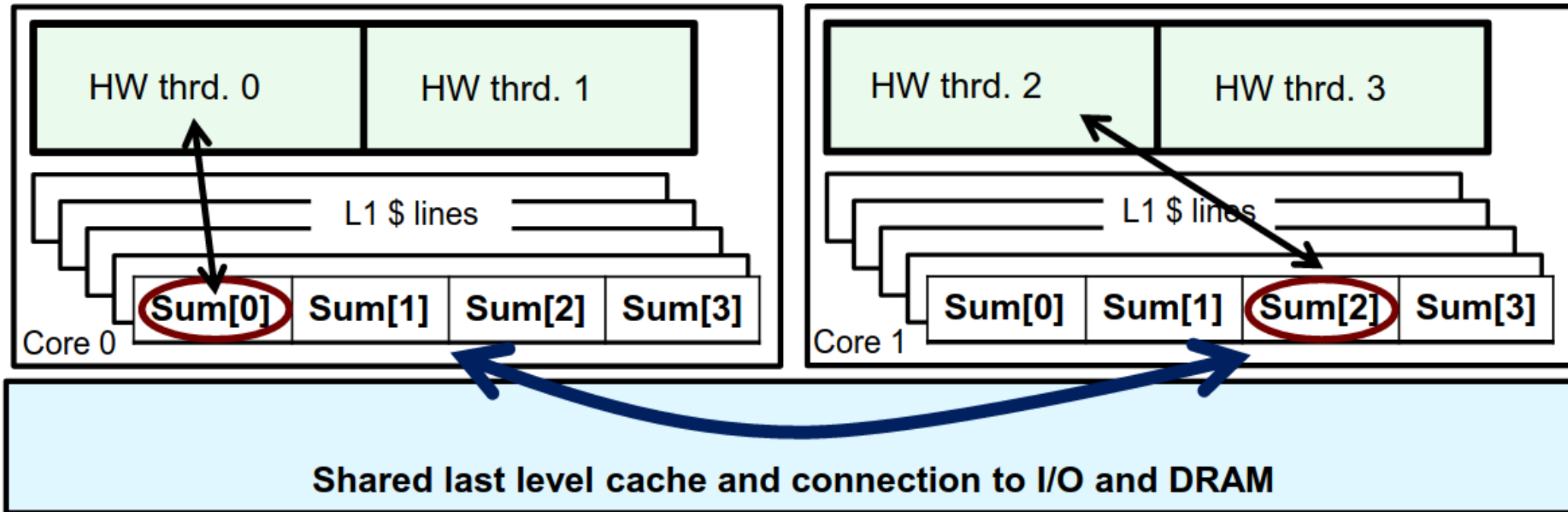
Hilos:1 3.9778399E-01 s
Hilos:2 5.2525401E-01 s
Hilos:3 5.2193189E-01 s
Hilos:4 4.0789413E-01 s



¿Qué f*ck está pasando???

FALSE SHARING

Problema 16



Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i] * step;
}
```

Hilos:1 3.9778399E-01 s
Hilos:2 5.2525401E-01 s
Hilos:3 5.2193189E-01 s
Hilos:4 4.0789413E-01 s

¿Qué hacer?



¿Qué f*ck está pasando???

FALSE SHARING

Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i] * step;
}
```

¿Y si es “compute bound”?

Hilos:1 3.9778399E-01 s
Hilos:2 5.2525401E-01 s
Hilos:3 5.2193189E-01 s
Hilos:4 4.0789413E-01 s



¿Qué f*ck está pasando???

FALSE SHARING

Problema 16

```
static long num_steps = 100000000; double step;
void main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i] * step;
}
```

Hilos:1 3.9778399E-01 s
Hilos:2 5.2525401E-01 s
Hilos:3 5.2193189E-01 s
Hilos:4 4.0789413E-01 s

¿Forzar que las
cachés sean
distintas en cada
hilo?



¿Qué f*ck está pasando???

FALSE SHARING

Problema 16

```
static long num_steps = 100000000; double step, int PAD=8;
```

```
int main ()
```

```
{
```

```
    int i, double pi, nthreads=NUM_THREADS;
```

```
    double sum[NUM_THREADS][PAD];
```

```
    step = 1.0/(double) num_steps;
```

```
    #pragma omp parallel
```

```
    {
```

```
        int i, id;
```

```
        double x;
```

```
        id = omp_get_thread_num();
```

```
        for (i=id, sum[id][0]=0.0; i< num_steps; i=i+nthreads) {
```

```
            x = (i+0.5)*step;
```

```
            sum[id][0] += 4.0/(1.0+x*x);
```

```
        }
```

```
    }
```

```
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i][0] * step;
```

Si las líneas de
caché son de
64B en L1...

Hilos:1

Hilos:2

Hilos:3

Hilos:4



Problema 16

```
static long num_steps = 100000000; double step, int PAD=8;
int main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS][PAD];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id][0]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id][0] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i][0] * step;
```

C almacena
en row-major,
así que...

Hilos:1
Hilos:2
Hilos:3
Hilos:4



Problema 16

```
static long num_steps = 100000000; double step, int PAD=8;
int main ()
{
    int i, double pi, nthreads=NUM_THREADS;
    double sum[NUM_THREADS][PAD];
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        int i, id;
        double x;
        id = omp_get_thread_num();
        for (i=id, sum[id][0]=0.0; i< num_steps; i=i+nthreads) {
            x = (i+0.5)*step;
            sum[id][0] += 4.0/(1.0+x*x);
        }
    }
    for(i=0, pi=0.0; i<nthreads; i++) pi += sum[i][0] * step;
```

Hilos:1 4.0145493E-01 s
Hilos:2 2.1057796E-01 s
Hilos:3 2.5786400E-01 s
Hilos:4 2.1443796E-01 s



**Derrochamos memoria, pero
va MEJOR... ¿Y OpenMP?**

Problema 16

```
static long num_steps = 100000000; double step;
int main ()
{
    int i, double pi[=0], nthreads=NUM_THREADS;
    double sum;
    step = 1.0/(double) num_steps;
    #pragma omp parallel for private(x) reduction(+:pi)
    for (i=0;i< num_steps; i++) {
        x = (i+0.5)*step;
        pi += 4.0/(1.0+x*x);
    }
    pi = pi * step;
```

Hilos:1 3.1362700E-01 s
Hilos:2 1.7782283E-01 s
Hilos:3 1.6215396E-01 s
Hilos:4 1.5540290E-01 s



¡Muy cerca del máximo teórico!

Problema 17

```
#include <stdio.h> #include <omp.h>
#define N 1000
int main(){
    int id,i;
    omp_set_num_threads(N);
    int vect[N];
    #pragma omp parallel private(id)
    {
        id=omp_get_thread_num();
        vect[id]=2*id;
    }
    for(i = 0; i < N; i++)
        printf("%i ",vect[i]);
    return 1;
}
```

Camino de cuda...

¿Es independiente del HW?

Problema 17

```
#include <stdio.h> #include <omp.h>
#define N 1000
int main(){
    int id,i;
    omp_set_num_threads(N);
    int vect[N];
    #pragma omp parallel private(id)
    {
        id=omp_get_thread_num();
        vect[id]=2*id;
    }
    for(i = 0; i < N; i++)
        printf("%i ",vect[i]);
    return 1;
}
```

Camino de cuda...

¿Es independiente del HW?
¿Núm máx de hilos?

Problema 17

```
#include <stdio.h> #include <omp.h>
#define N 1000
int main(){
    int id,i;
    omp_set_num_threads(N);
    int vect[N];
    #pragma omp parallel private(id)
    {
        id=omp_get_thread_num();
        vect[id]=2*id;
    }
    for(i = 0; i < N; i++)
        printf("%i ",vect[i]);
    return 1;
}
```

Camino de cuda...

¿Es independiente del HW?

¿Núm máx de hilos?

Linux dice...:

cat /proc/sys/kernel/threads-max
62461

Pero... no más de 3500 aprox.

Problema 17

```
#include <stdio.h> #include <omp.h>
#define N 1000
int main(){
    int id,i;
    omp_set_num_threads(N);
    int vect[N];
    #pragma omp parallel private(id)
    {
        id=omp_get_thread_num();
        vect[id]=2*id;
    }
    for(i = 0; i < N; i++)
        printf("%i ",vect[i]);
    return 1;
}
```

Camino de cuda...

¿Es independiente del HW?

¿Núm máx de hilos?

Linux dice...:

cat /proc/sys/kernel/threads-max
62461

Pero... no más de 3500 aprox.

¿TIENE SENTIDO????