PROGRAMACIÓN AVANZADA

Sincronización en OpenMP

SOLUCIÓN AL EJERCICIO DE LA CLASE ANTERIOR

AUC SECUENCIAL

```
#include <stdio.h>

int main(void)
{
    const int numSteps =
    100000000;
    const int highX = 10;
    const double deltaX = 1.0 *
    highX / numSteps;

    double auc = 0;
```

```
for (int i = 0; i < numSteps;
i++)
{
    double leftX = (i - 1) * deltaX;
    double rightX = (i + 1) *
    deltaX;
    double b1 = sqrt(leftX<0 ? 0 :
    leftX);
    double b2 = sqrt(rightX);
    auc += 0.5*(b1 + b2) * deltaX;
}
printf("The integral is: %f",
auc); }
```

SOLUCIÓN AL EJERCICIO DE LA CLASE ANTERIOR

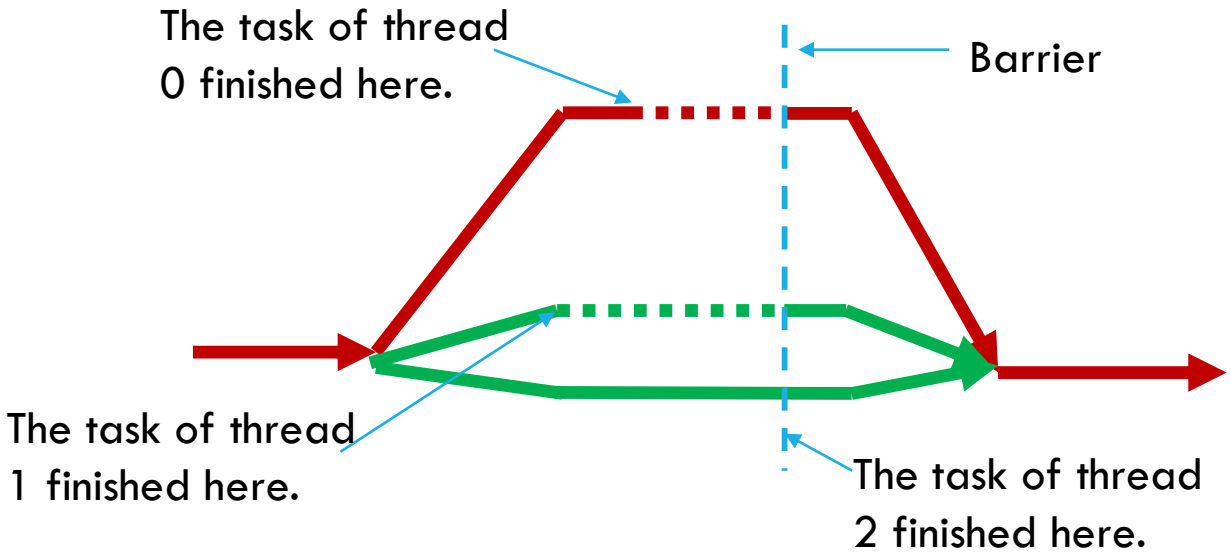
AUC PARALELO

```
#include <omp.h>
#include <stdio.h>
#define NUM_THREADS 2
int main(void){
    const int numSteps = 1000000;
    const int highX = 10;
    const double deltaX = 1.0 * highX
    /   numSteps;
    int nThreads = 0;
    double sum[NUM_THREADS];
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    {
        const int id =
        omp_get_thread_num();
        const int currentNThreads =
            omp_get_num_threads();
    if (id == 0)
        nThreads = currentNThreads;
     sum[id] = 0.0;
     for (int i = 0; i < numSteps; i +=
currentNThreads){
         double leftX = (i-1) * deltaX;
         double rightX = (i+1) * deltaX;
         double b1 = sqrt(leftX < 0 ? 0 :
leftX);
         double b2 = sqrt(rightX);
         sum[id] += 0.5 * (b1 + b2) *
deltaX;
     }
}
double auc = 0;
for (int i = 0; i < nThreads; i++)
    auc += sum[i];
printf("The integral is: %f", auc); }
```

SINCRONIZACIÓN

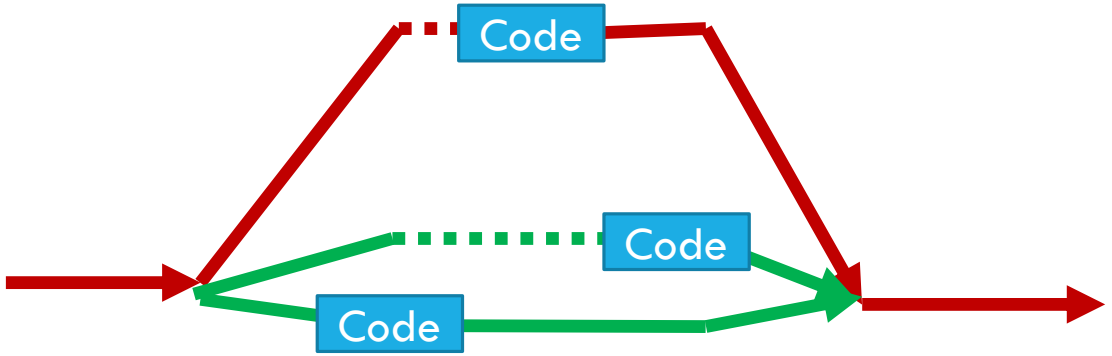
Barrera:

- Cada hilo espera en la barrera hasta que lleguen todos los hilos.

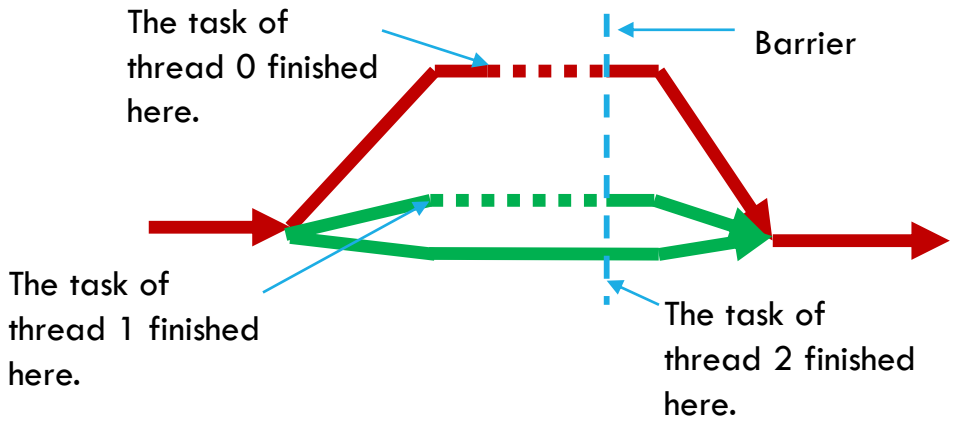


Exclusión mutua

- Defina un bloque de código que solo puede ejecutar un hilo a la vez.



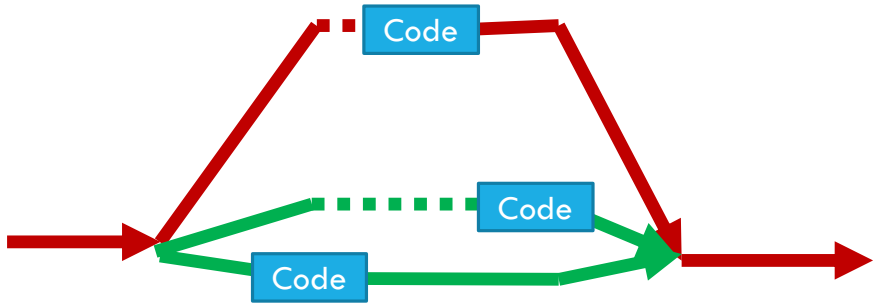
EJEMPLO DE BARRERA




```
#pragma omp parallel
{
    const int id = omp_get_thread_num();
    someArray[id] = veryTimeConsumingFunction(id);

#pragma omp barrier

    otherArray[id] = otherVeryTimeConsumingFunction(id, someArray);
}
```

EXCLUSIÓN MUTUA: CONSTRUCCIÓN CRÍTICA

```
double sum = 0.0; int i;
#pragma omp parallel
{
    const int id = omp_get_thread_num();
    const int currentNThreads = omp_get_num_threads();
    for (i = id; i< numSteps; i+= currentNThreads)
    {
        double currentResult = veryTimeConsumingFunction(id);

#pragma omp critical
        sum += consume(currentResult);
    }
}
```



Si la operación a realizar en la construcción crítica es una de las siguientes x binop= expr, x ++, x--, ++ x, --x; entonces, usa la construcción atómica.

EJERCICIO:

Usando algún elemento de sincronización, o ambos, modifique el código para calcular AUC de manera que se elimine el arreglo con las sumas, y todas las sumas se vayan acumulando en una sola variable, de tipo **float**.

TAREA 5

MUCHAS GRACIAS