Minho
Advanced
Computing
Center

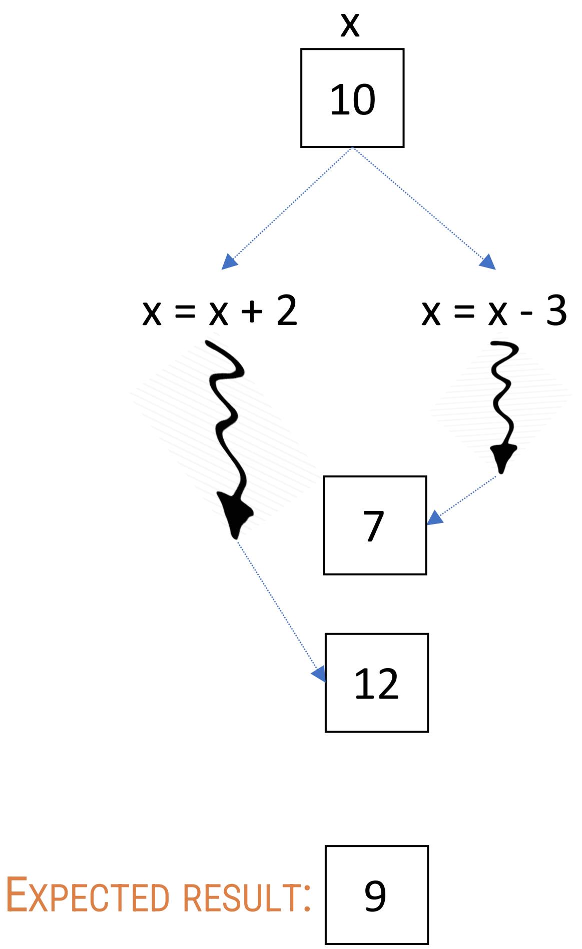
OpenMP

Advanced Concepts

André Pereira
Minho Advanced Computing Center
ampereira@macc.fccn.pt

OpenMP – Race Conditions

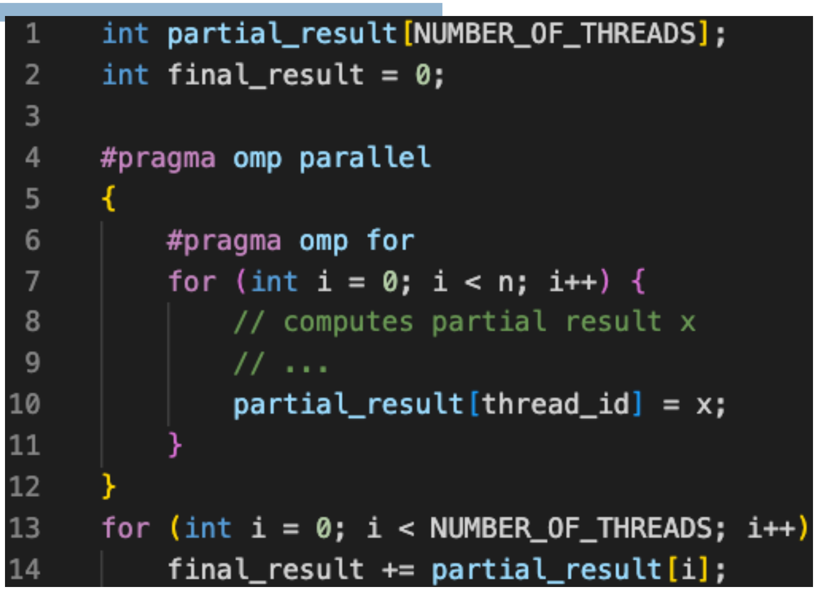
- Inadequate management of concurrent accesses may create wrong results
  - Data races
  - Read-after-write and write-after-read dependencies must be handled by the programmer
- OpenMP provides pragmas to control access to shared data
  - `#pragma omp critical`
- Algorithms and/or data structures may need to be redesigned



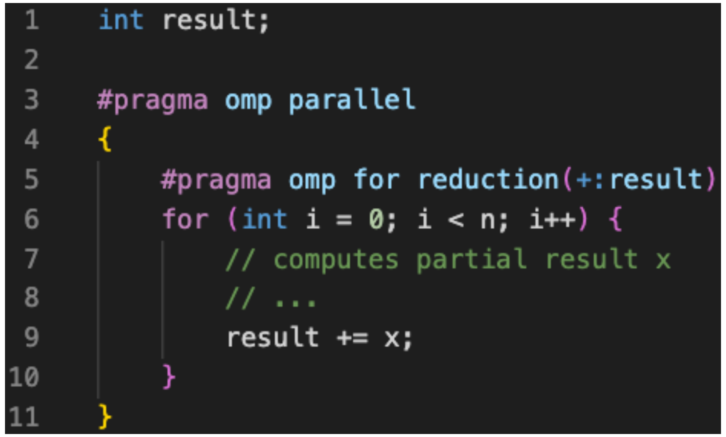
OpenMP – Reduction

Threads often compute a subset of a workload

- Partial results stored privately to each thread
  - Improves performance (less contention to access shared data)
  - Reduces the risk of data races
- Partial results must be merged after the parallel section
  - can be implemented manually
  - `#pragma omp reduction` does this automatically (restrictions apply)

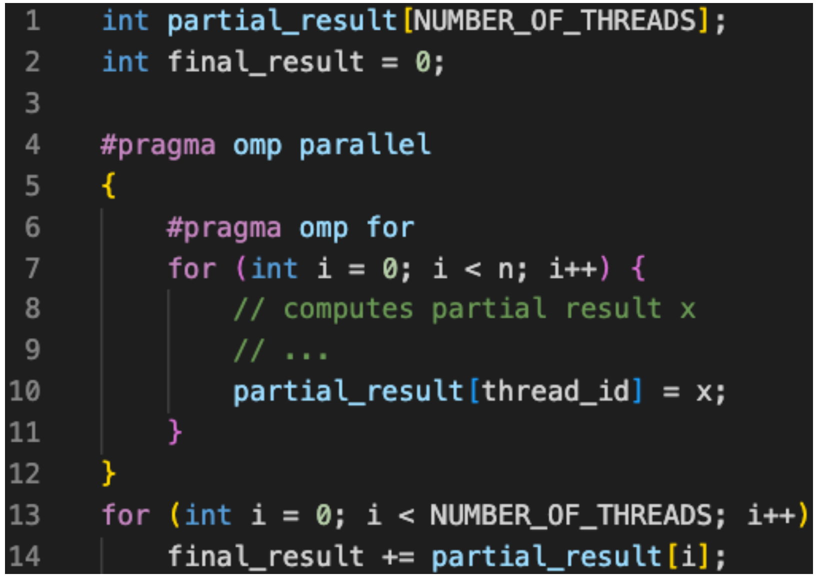


```c
int partial_result[NUMBER_OF_THREADS];
int final_result = 0;

#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < n; i++) {
        // computes partial result x
        // ...
        partial_result[thread_id] = x;
    }
}
for (int i = 0; i < NUMBER_OF_THREADS; i++)
    final_result += partial_result[i];
```

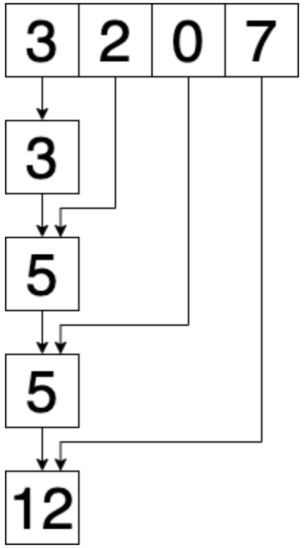


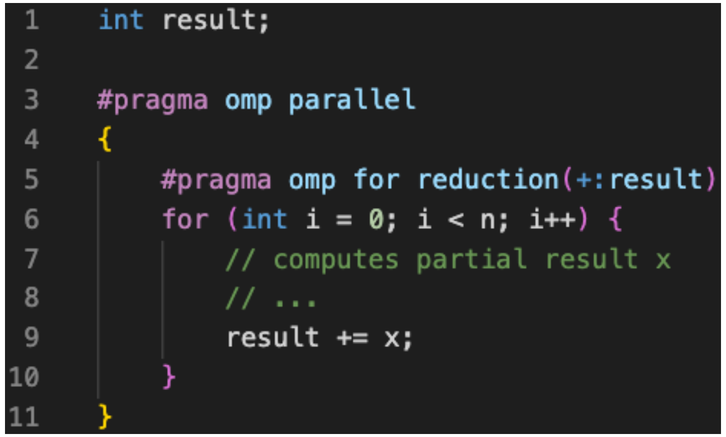
```c
int result;

#pragma omp parallel
{
    #pragma omp for reduction(+:result)
    for (int i = 0; i < n; i++) {
        // computes partial result x
        // ...
        result += x;
    }
}
```

OpenMP – Reduction

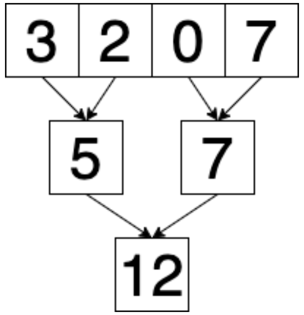


```
int partial_result[NUMBER_OF_THREADS];
int final_result = 0;

#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < n; i++) {
        // computes partial result x
        // ...
        partial_result[thread_id] = x;
    }
}
for (int i = 0; i < NUMBER_OF_THREADS; i++)
    final_result += partial_result[i];
```





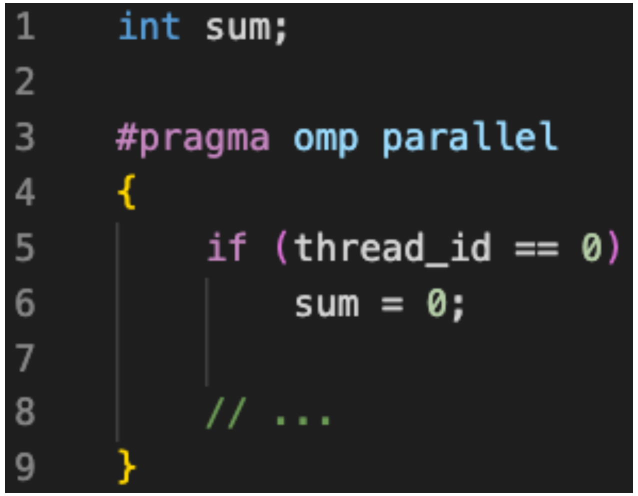
```
int result;

#pragma omp parallel
{
    #pragma omp for reduction(+:result)
    for (int i = 0; i < n; i++) {
        // computes partial result x
        // ...
        result += x;
    }
}
```



OpenMP – Thread Management

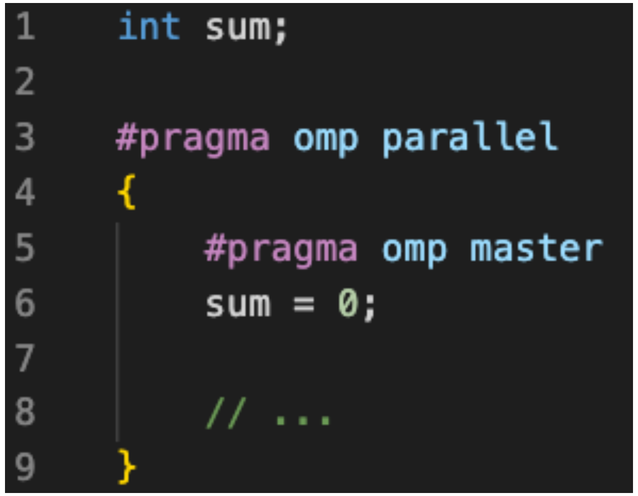
Controlling the sections of code that each thread executes can be achieved through:

- Each thread individual id
- OpenMP pragma directives
    - **Master**: only the master thread executes the code section
    - **Single**: the first thread to arrive executes the code section



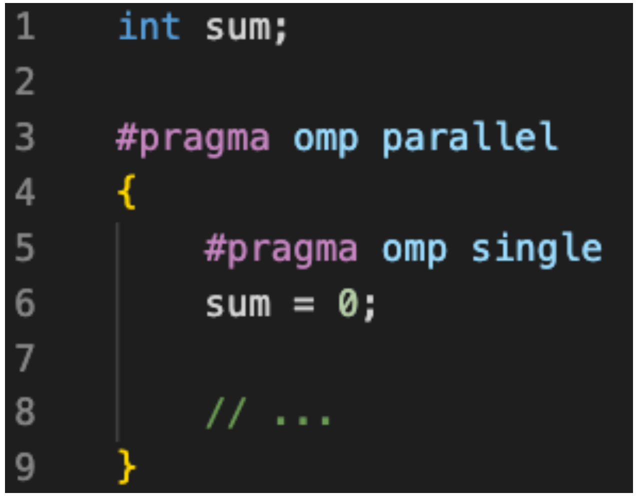
```
int sum;

#pragma omp parallel
{
    if (thread_id == 0)
        sum = 0;

    // ...
}
```

=



```
int sum;

#pragma omp parallel
{
    #pragma omp master
    sum = 0;

    // ...
}
```

≠



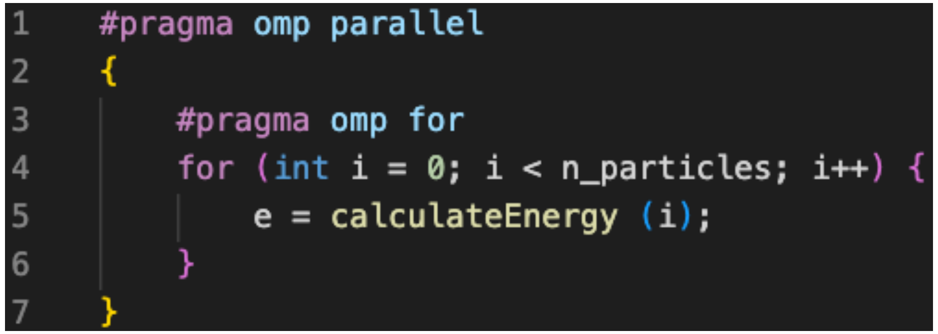
```
int sum;

#pragma omp parallel
{
    #pragma omp single
    sum = 0;

    // ...
}
```

OpenMP – Loop Scheduling

Workloads can be classified as

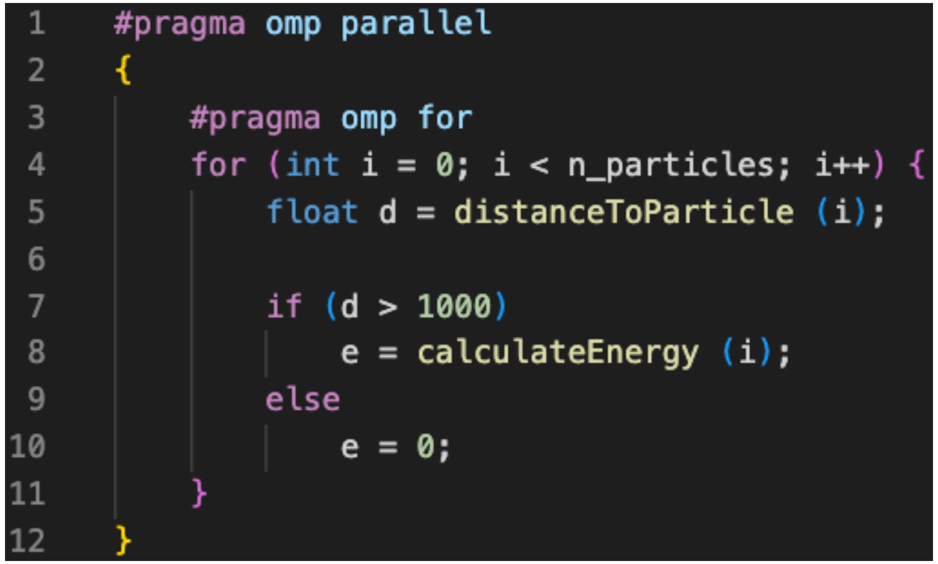
- **Regular**: each loop iteration takes the same amount of time
- **Irregular**: loop iterations take different amount of time

The execution time of a parallel region is the time of the slowest thread



```
#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < n_particles; i++) {
        e = calculateEnergy (i);
    }
}
```

Regular workload



```
#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < n_particles; i++) {
        float d = distanceToParticle (i);

        if (d > 1000)
            e = calculateEnergy (i);
        else
            e = 0;
    }
}
```

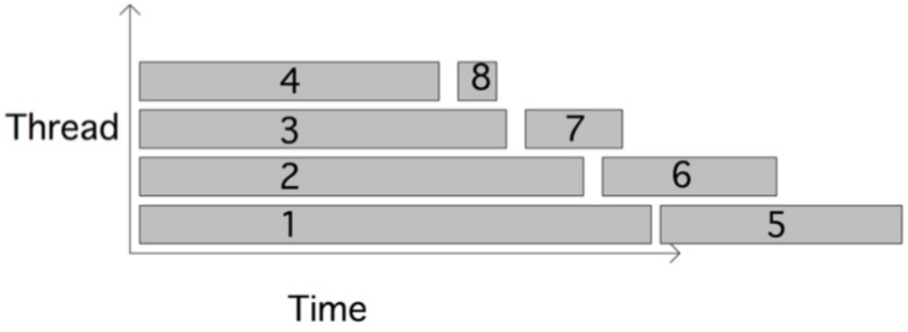
Irregular workload

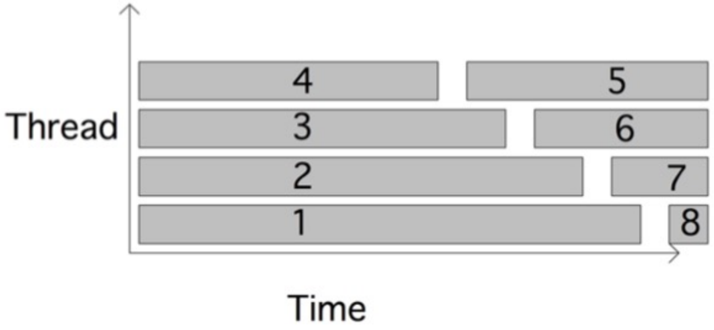
OpenMP – Loop Scheduling

Workloads can be classified as

- Regular: each loop iteration takes the same amount of time
- Irregular: loop iterations take different amount of time

The execution time of a parallel region is the time of the slowest thread



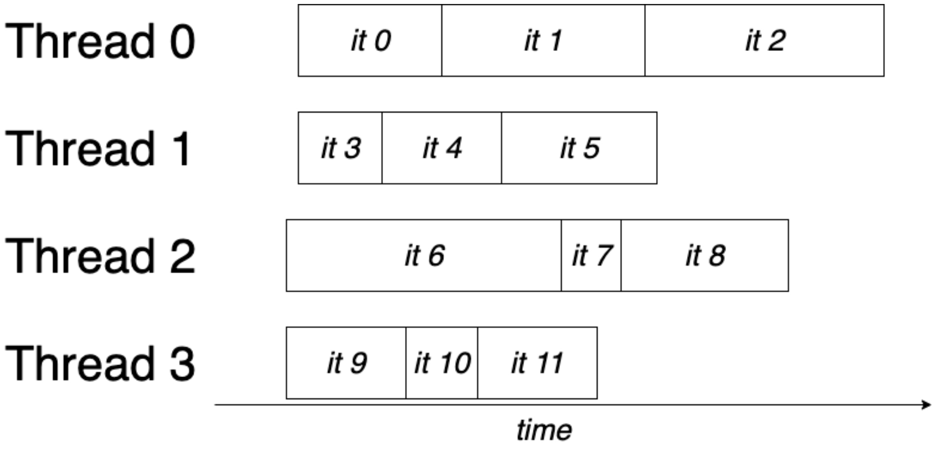


OpenMP – Loop Scheduling

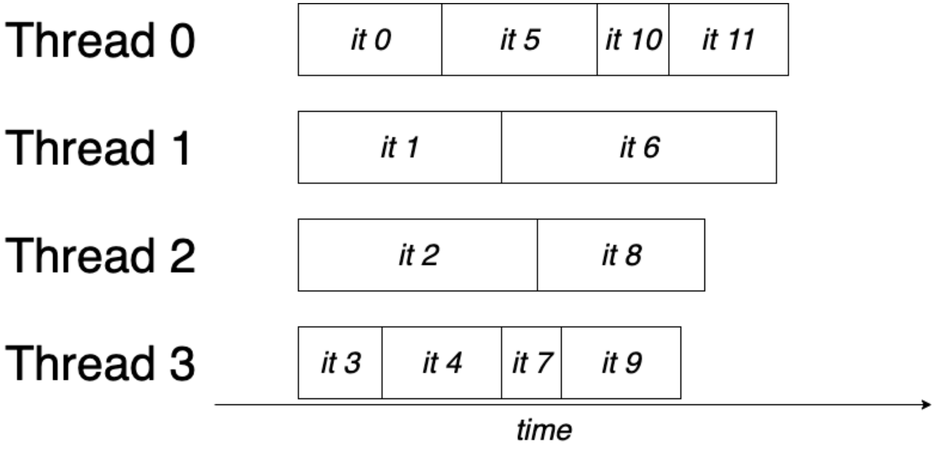
OpenMP loop scheduling options

- static
  - Iterations are assigned equally among threads prior to the parallel section
  - Small overhead
  - Static user-defined chunk size
- dynamic
  - Each thread gets iterations according to their throughput
  - Higher overhead
  - Static user-defined chunk size
- guided
  - Similar to dynamic but adjusts the chunk size during runtime
  - Less overhead than dynamic

STATIC SCHEDULING



DYNAMIC SCHEDULING



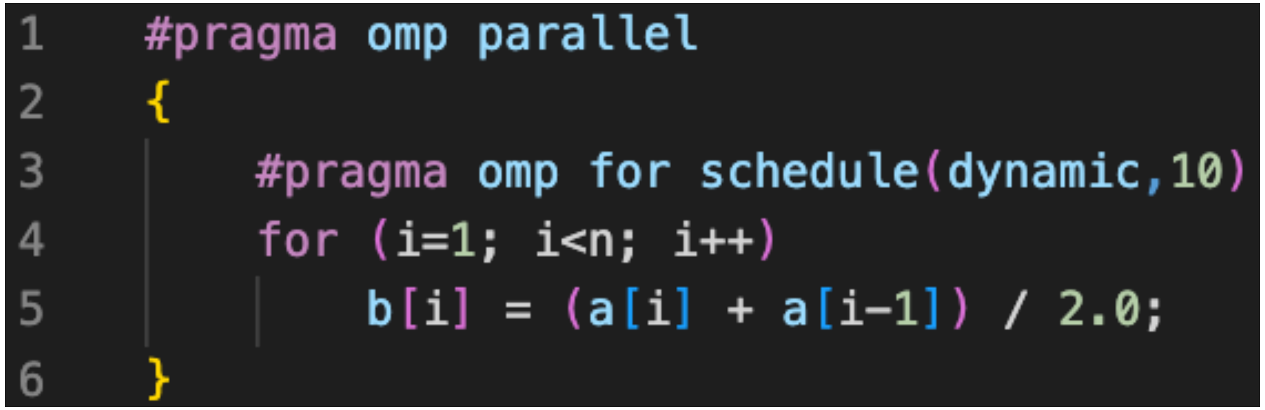
OpenMP – Loop Scheduling

OpenMP loop scheduling options

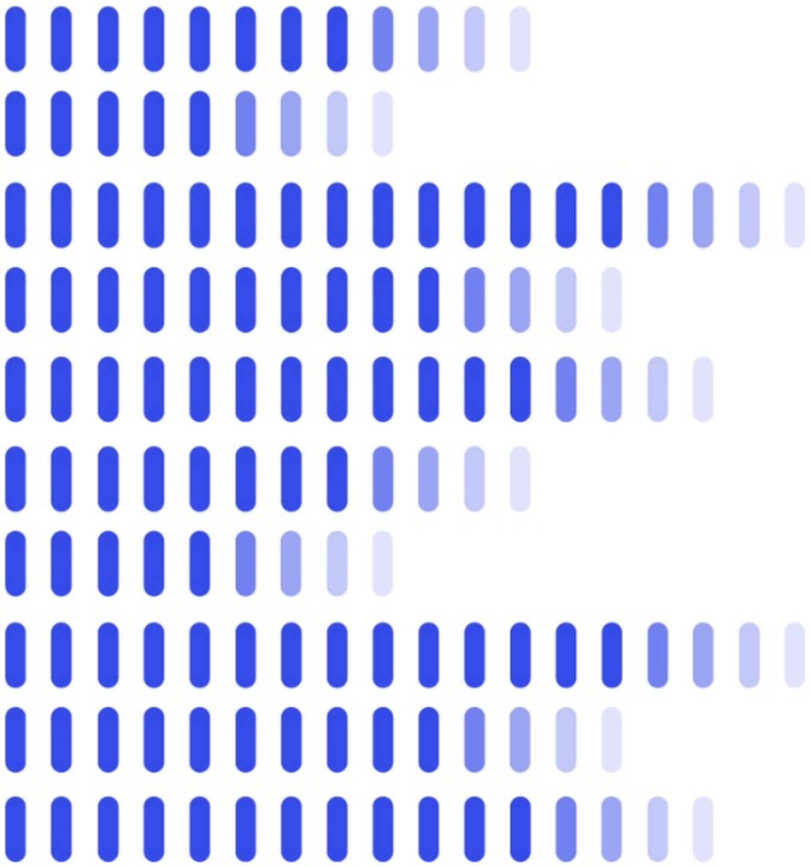
- static
- dynamic
- guided

Chunk size can also be user defined

- Set as a single iteration by default
- Larger chunks require less scheduling overhead
- Smaller chunks are better to schedule irregular workloads



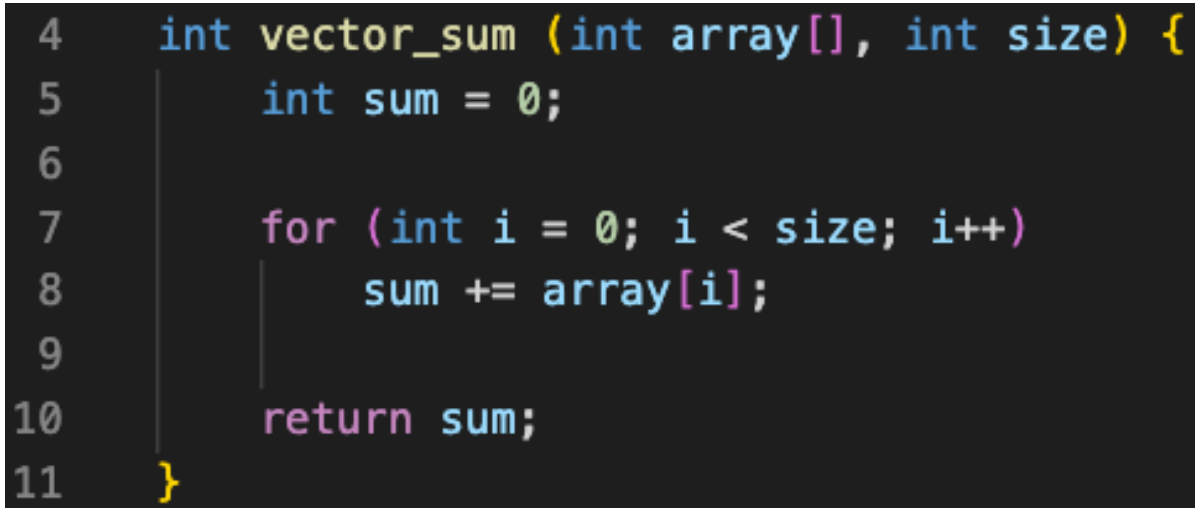
```
#pragma omp parallel
{
    #pragma omp for schedule(dynamic,10)
    for (i=1; i<n; i++)
        b[i] = (a[i] + a[i-1]) / 2.0;
}
```



Lab Session

Vector Sum

1. A simple parallelization
    - Parallelize the code using **`#pragma omp parallel`**
    - Distribute the iterations using a **`#pragma omp for`** directive
    - Use the **`critical`** directive on shared data if necessary
    - Execute and measure the performance of the code

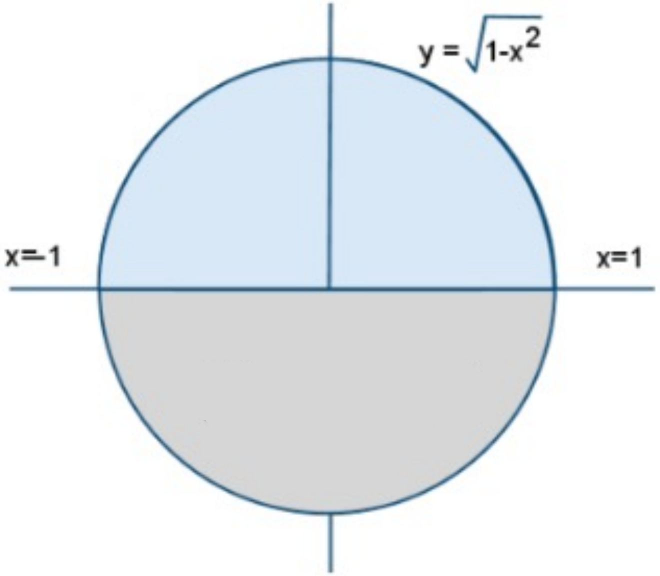


```
int vector_sum (int array[], int size) {
    int sum = 0;

    for (int i = 0; i < size; i++)
        sum += array[i];

    return sum;
}
```

2. Privatize shared data
    - Privatize the accesses to shared data
        - each thread should contain it’s copy of sum
    - Implement a manual merge of the partial results
    - Execute and measure the performance of the code

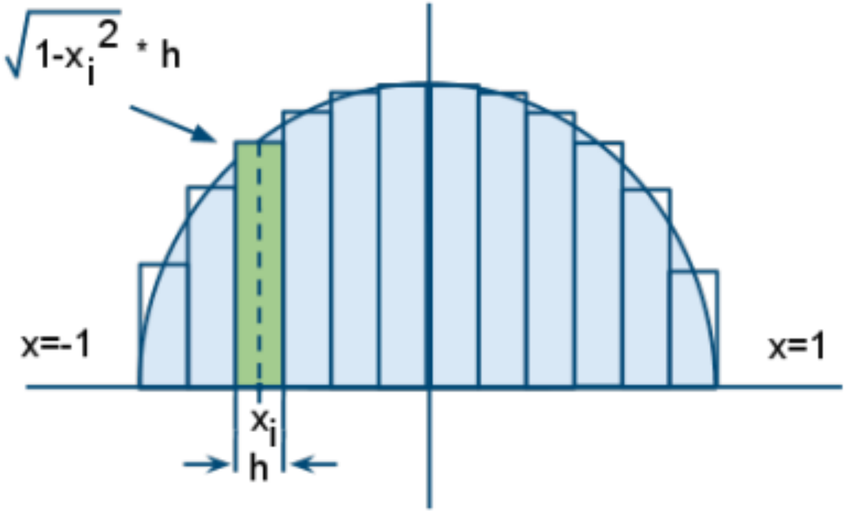
3. Removing implicit synchronizations
    - Replace the manual merge of the results with OpenMP’s reduction directive
    - Execute and measure the performance of the code

Pi Calculation - Integral Approximation

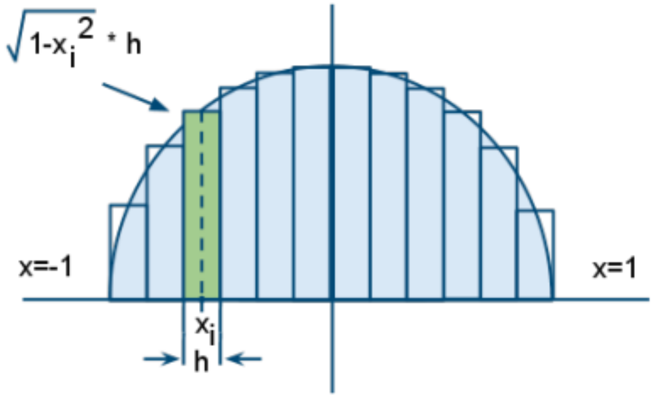


Pi can be calculated through the area of a circle

- The integral of the function of a circle is equivalent
- Integrals can be approximated through iterative methods

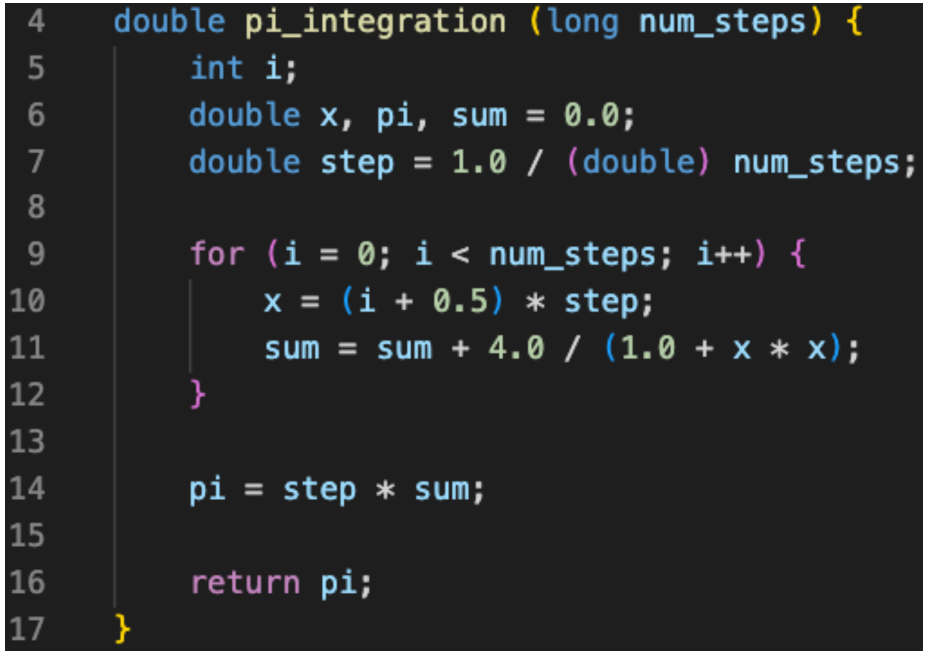


Pi Calculation - Integral Approximation



For loop parallelization

- Parallelize the code using the **`omp parallel`** and **`omp for`** directives
- Privatize variables if needed
- Each thread should compute a partial sum (avoid race conditions!)
- Merge the partial results using OpenMP directives
- Execute the code and measure its performance



```
double pi_integration (long num_steps) {
    int i;
    double x, pi, sum = 0.0;
    double step = 1.0 / (double) num_steps;

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

    pi = step * sum;

    return pi;
}
```

Thank you for attending!

→

hello@macc.fccn.pt

macc.fccn.pt

@minhoacc