

Parallel and Distributed Computing

CS3006 (BDS-6A)

Lecture 09

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science, FAST

23 February, 2023

Previous Lecture

- Decomposition Techniques
 - Recursive
 - Data (Input, Output, Intermediate)
 - Exploratory
 - Speculative
 - Hybrid
- Distribution Schemes & Mapping (static, dynamic)
 - Block-distribution (1D, 2D, cyclic, random)
- POSIX threads
- OpenMP: intro & library functions, Hello World Example

First Program: hello world

```
#include <omp.h>
#include <iostream>
using namespace std;
```

```
int main()
```

```
{
```

```
    omp_set_num_threads(4);
```

```
    #pragma omp parallel
```

```
    {
```

```
        int Id = omp_get_thread_num();
```

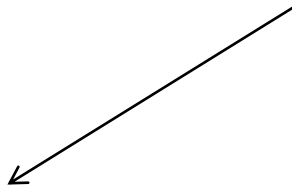
```
        printf ("hello(%d)", Id);
```

```
        printf ("world(%d)\n", Id)
```

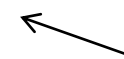
```
    }
```

```
}
```

Runtime function to request a certain number of threads

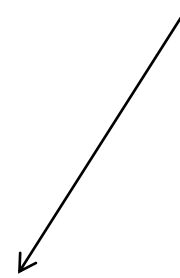


Runtime function returning a thread ID



```
#include <omp.h>
int numT;
int main()
{
    #pragma omp parallel num_threads(4)
    {
        int Id = omp_get_thread_num();
        numT = omp_get_num_threads();
        printf ("hello(%d)", Id);
        printf ("world(%d)\n", Id)
    }
}
```

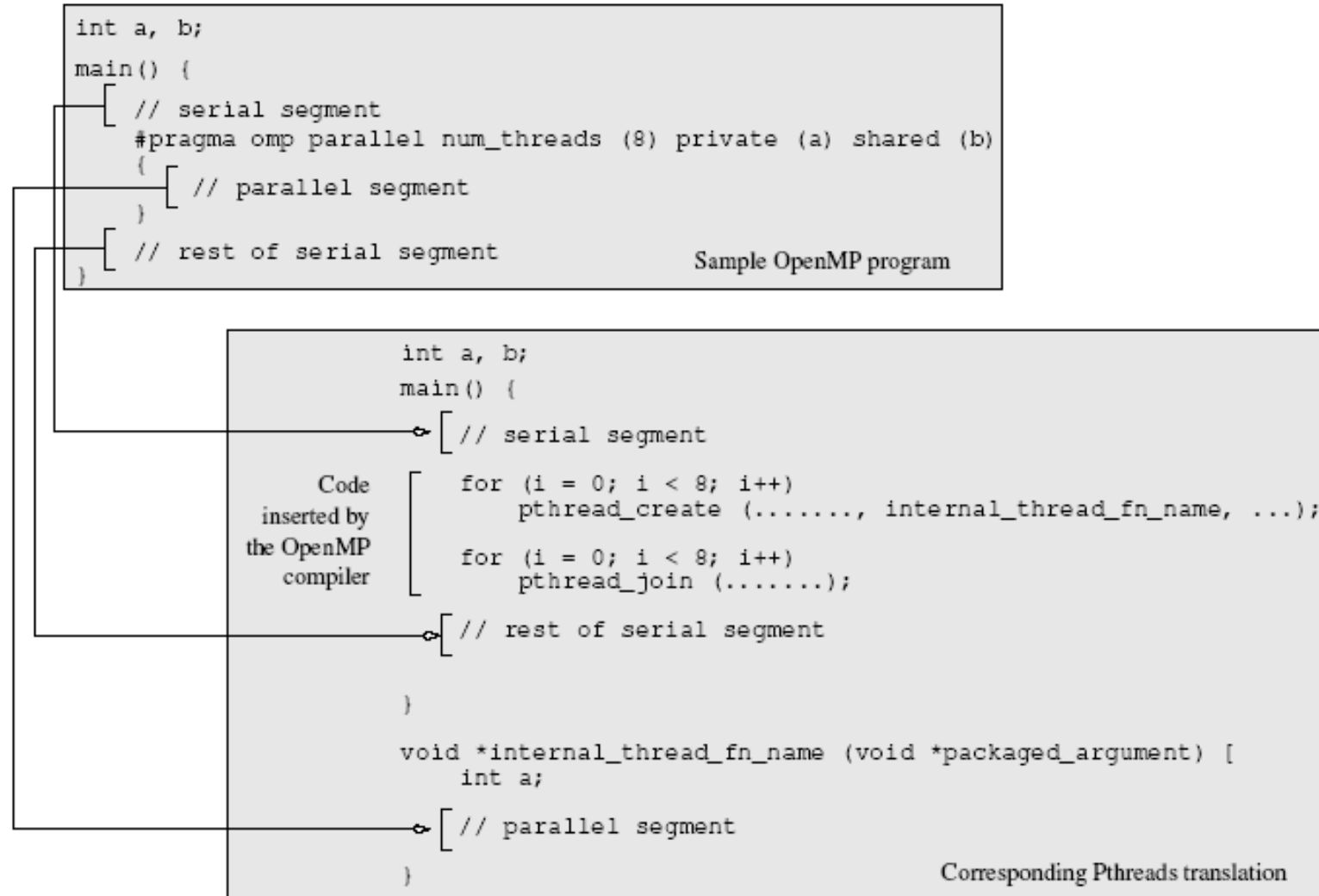
Clause to request a certain number of threads



Runtime function
returning the num of threads actually
created

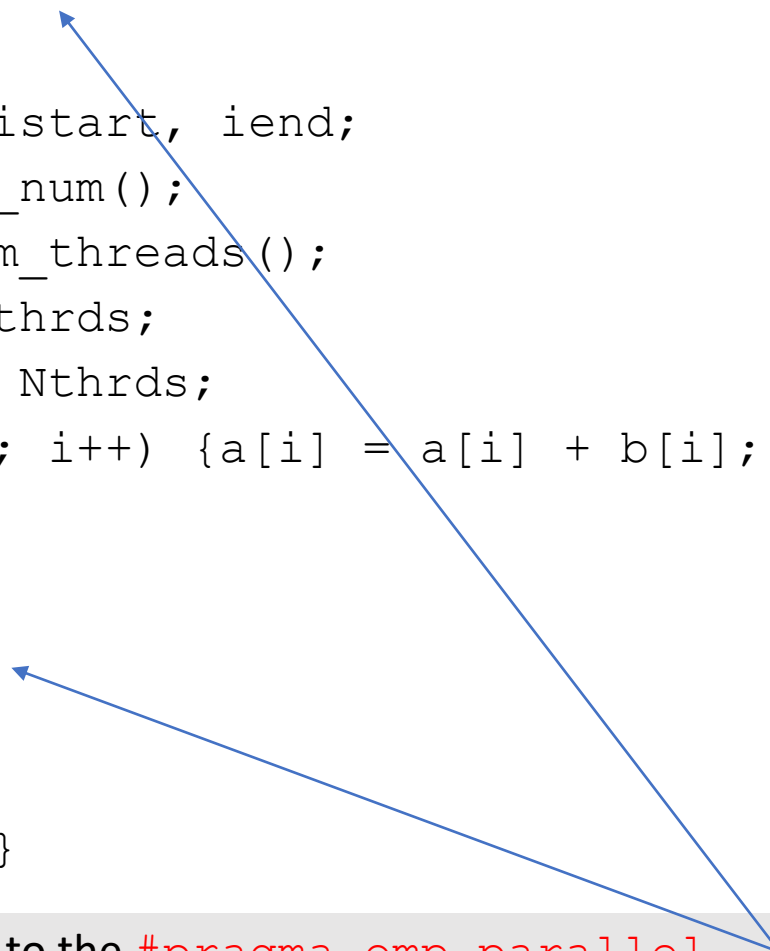


Program Structure



Source: Introduction to Parallel Computing (Karypis and Co.)

Loop work-sharing Construct

- **Sequential Code**
 - `for(i = istart; i < iend; i++) { a[i] = a[i] + b[i]; }`
 - **OpenMP Parallel Region**
 - `#pragma omp parallel`
 - `{`
 - `int id, i, Nthrds, istart, iend;`
 - `id = omp_get_thread_num();`
 - `Nthrds = omp_get_num_threads();`
 - `istart = id * N / Nthrds;`
 - `iend = (id+1) * N / Nthrds;`
 - `for(i=istart; i<iend; i++) {a[i] = a[i] + b[i];}`
 - `}`
 - **OpenMP parallel region and a worksharing `for` construct**
 - `#pragma omp parallel`
 - `#pragma omp for`
 - `for(i=0; i<N; i++) {`
 - `a[i] = a[i] + b[i];}`
- 

Need to add something like `num_threads(4)` to the `#pragma omp parallel`

What is OpenMP?

- OpenMP (Open Multi-Processing) is a popular shared-memory programming API
- OpenMP supports C/C++ and Fortran on a wide variety of architectures
- OpenMP is supported by popular C/C++ compilers, for e.g., LLVM/Clang, GNU GCC, Intel ICC, and IBM XLC

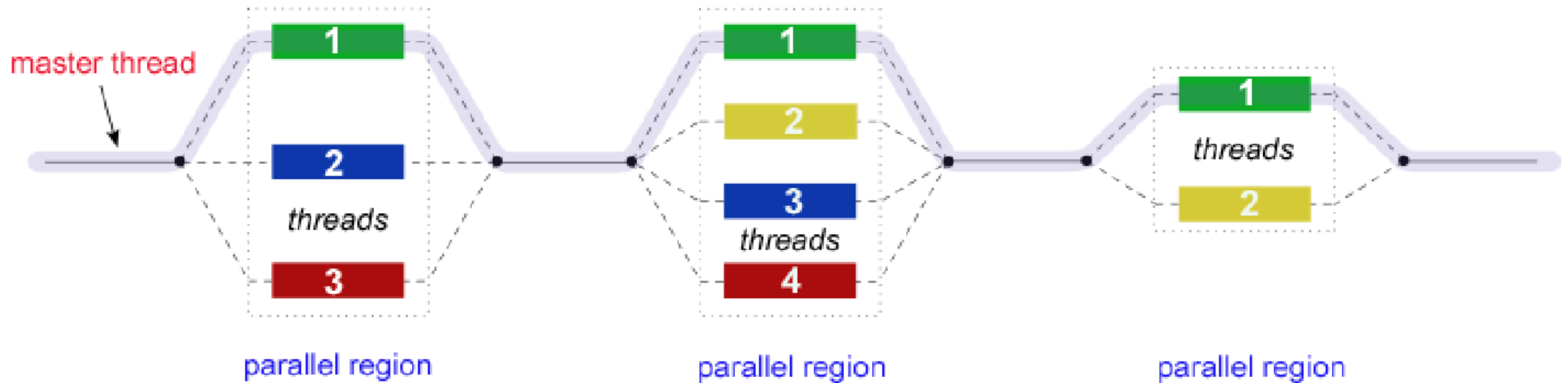
What is OpenMP?

- A directive based parallel programming model
 - OpenMP program is essentially a sequential program augmented with compiler directives to specify parallelism
 - Eases conversion of existing sequential programs

Key Concepts in OpenMP

- Parallel regions where parallel execution occurs via multiple concurrently executing threads
 - Each thread has its own program counter and executes one instruction at a time, similar to sequential program execution
- Shared and private data: shared variables are the means of communicating data between threads
- Synchronization: fundamental means of coordinating execution of concurrent threads
- Mechanism for ***automated work distribution*** across threads

Fork-Join Model of Parallel Execution



Compiling an OpenMP Program

- **Linux and GNU GCC:**
 - `g++ -fopenmp hello-world.cpp`
- **Linux and Clang/LLVM:**
 - `clang++ -fopenmp hello-world.cpp`

Basics to an OpenMP Program

- Each thread has a unique integer “id”; master thread has “id” 0, and other threads have “id” 1, 2, ...
- OpenMP runtime function `omp_get_thread_num()` returns a thread’s unique “id”
- The function `omp_get_num_threads()` returns the total number of executing threads
- The function `omp_set_num_threads(x)` asks for “x” threads to execute in the next parallel region (must be set outside region)

Numerical Integration: Pi program

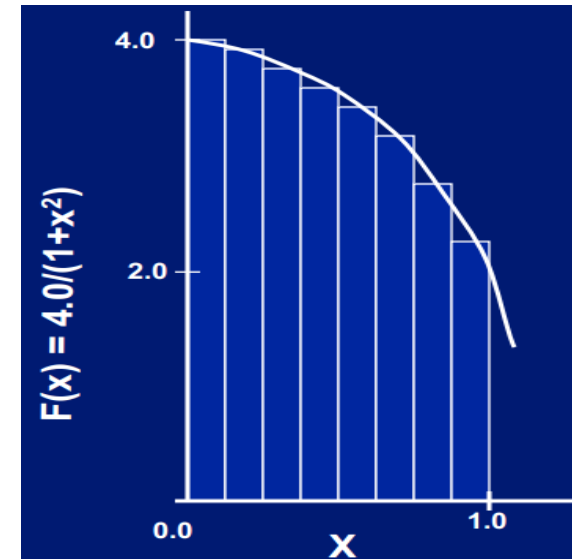
- Mathematically

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

- Integral can be approximated as a sum of rectangles:

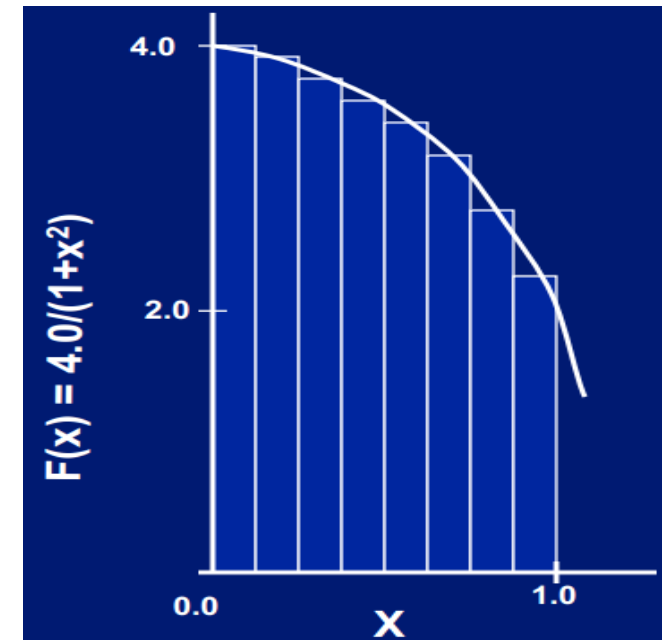
$$\sum_{i=0}^N F(x_i) \Delta x \approx \pi$$

- Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i .



Serial Pi Program

```
static long num_steps = 100000;
double step;
void main ()
{
    int i; double x, pi, sum = 0.0;
    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;
}
```



Parallel Pi program

```
static long num_steps = 100000;
double step;
#define NUM_THREADS 4

int _tmain(int argc, _TCHAR* argv[]) {
    int i, nthreads;
    double pi, sum[NUM_THREADS];
    step = 1.0/(double) num_steps;
    omp_set_num_threads(NUM_THREADS);
#pragma omp parallel
    {
        int i, id, nthrds;
        double x;
        id = omp_get_thread_num();
        nthrds = omp_get_num_threads();
        if(id == 0)
            nthreads = nthrds;
        for (i=id, sum[id]=0.0; i< num_steps; i=i+nthrds){
            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;
    printf("%f\n", pi);
    return 0;
}
```

Synchronization

- **Mutual exclusion:** Only one thread at a time can enter critical section

```
#pragma omp critical [name]
{
    code_block
}
```

- **[name]** optional name that identifies the critical directive

Parallel Pi program using mutual exclusion

```
static long num_steps = 100000;
double step;
#define NUM_THREADS 2

int main(int argc, CHAR* argv[]) {
    double pi;
    step = 1.0/(double) num_steps;
    omp_set_num_threads(NUM_THREADS);
#pragma omp parallel
    {
        int i, id, nthrds;
        double x, sum;
        id = omp_get_thread_num();
        nthrds = omp_get_num_threads();

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

#pragma omp critical
        pi += (sum * step);
    }
    printf("%f\n", pi);
}
```

Assigning Iteration to threads

- Static Scheduling

- Splits the iteration space into equal chunks of size `chunk_size`
- Assign them to threads in round-robin fashion
- When no `chunk_size` is specified, the iteration space is split into as many chunks as there are threads and one chunk is assigned to each thread.
- `#pragma omp for schedule(static, chunk_size)`
- `#pragma omp for schedule(static)`

Assigning Iteration to threads

- Due to a number of reasons, including heterogeneous computing resources, non-uniform processor load, even equally partitioned workloads take widely varying execution times
- **Dynamic Scheduling**
 - Assign them (i.e., iterations) to threads when they become idle
 - Take care of the temporal imbalances resulting from static scheduling
 - If no `chunk_size` is specified, it defaults to a single iteration per chunk
 - `#pragma omp for schedule(dynamic, chunk_size)`
 - `#pragma omp for schedule(static)`

Reduction

- Sometimes there is true dependency between loop iterations which can not be removed.

```
double average=0.0, A[MAX]; int i;  
for (i=0; i< MAX; i++) {  
    average + = A[i];  
}  
average = average/MAX;
```

- We are combining values into a single accumulation variable (average)
- How to share work among threads. Such situation is called Reduction.

OpenMP reduction clause

- **reduction (op : list)**
- Inside a parallel work a local copy of each list variable is made
- List variable is initialized depending on the “op”
(e.g. 0 for “+”)
- Compiler updates each local copy
- Local copies are reduced into a single value and combined with the original global value

```
double ave=0.0, A[MAX]; int i;  
#pragma omp parallel for reduction (+:ave)  
for (i=0; i< MAX; i++) { ave + = A[i]; }  
ave = ave/MAX;
```

References

1. Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing* (Vol. 110). Redwood City, CA: Benjamin/Cummings.
2. Quinn, M. J. Parallel Programming in C with MPI and OpenMP, (2003).
3. <https://www.cse.iitk.ac.in/users/swarnendu/courses/autumn2019-cs698l/OpenMP.pdf>
4. <https://www3.nd.edu/~z xu2/acms60212-40212/Lec-12-OpenMP.pdf>