CSE502: Foundations of Parallel Programming

Lecture 17: Introduction to OpenMP

Vivek Kumar

Computer Science and Engineering

IIIT Delhi

vivekk@iiitd.ac.in

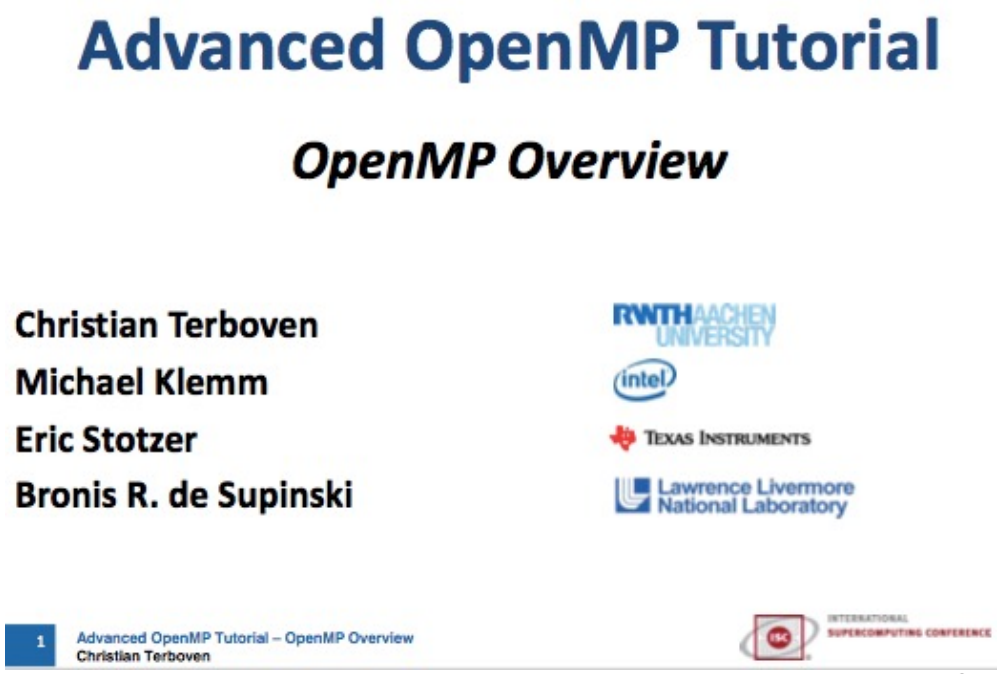
Today's Class

- Introduction to OpenMP
  - Work-sharing constructs in OpenMP

Acknowledgements: Slides heavily borrowed from following two sources:

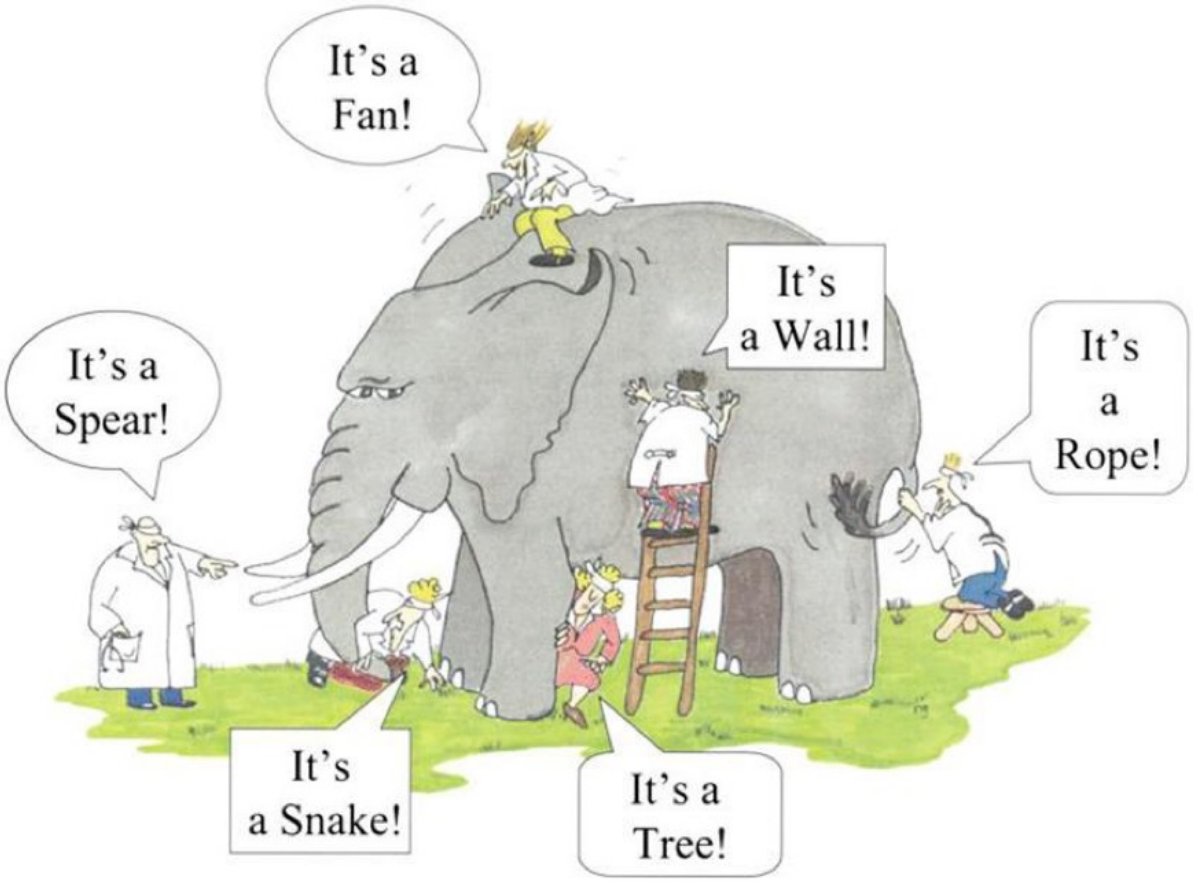
a) ECE563, Purdue University, Dr. Seung-Jai Min

b) COMP422, Rice University, Dr. Vivek Sarkar



What is OpenMP?

- De-facto standard Application Programming Interface (API) to write shared memory parallel applications in C, C++, and Fortran
- Consists of
  - Compiler Directives
  - Runtime routines
  - and Environment variables



OpenMP* Overview:

C$OMP FLUSH

#pragma omp critical

C$OMP THREADPRIVATE(/ABC/)

CALL OMP_SET_NUM_THREADS(10)

C$OM

C$OM

C$(

C

#p

ED

OpenMP: An API for Writing Multithreaded Applications

- A set of compiler directives and library routines for parallel application programmers
- Greatly simplifies writing multi-threaded (MT) programs in Fortran, C and C++
- Standardizes last 20 years of SMP practice

C$OMP PARALLEL COPYIN(/blk/)

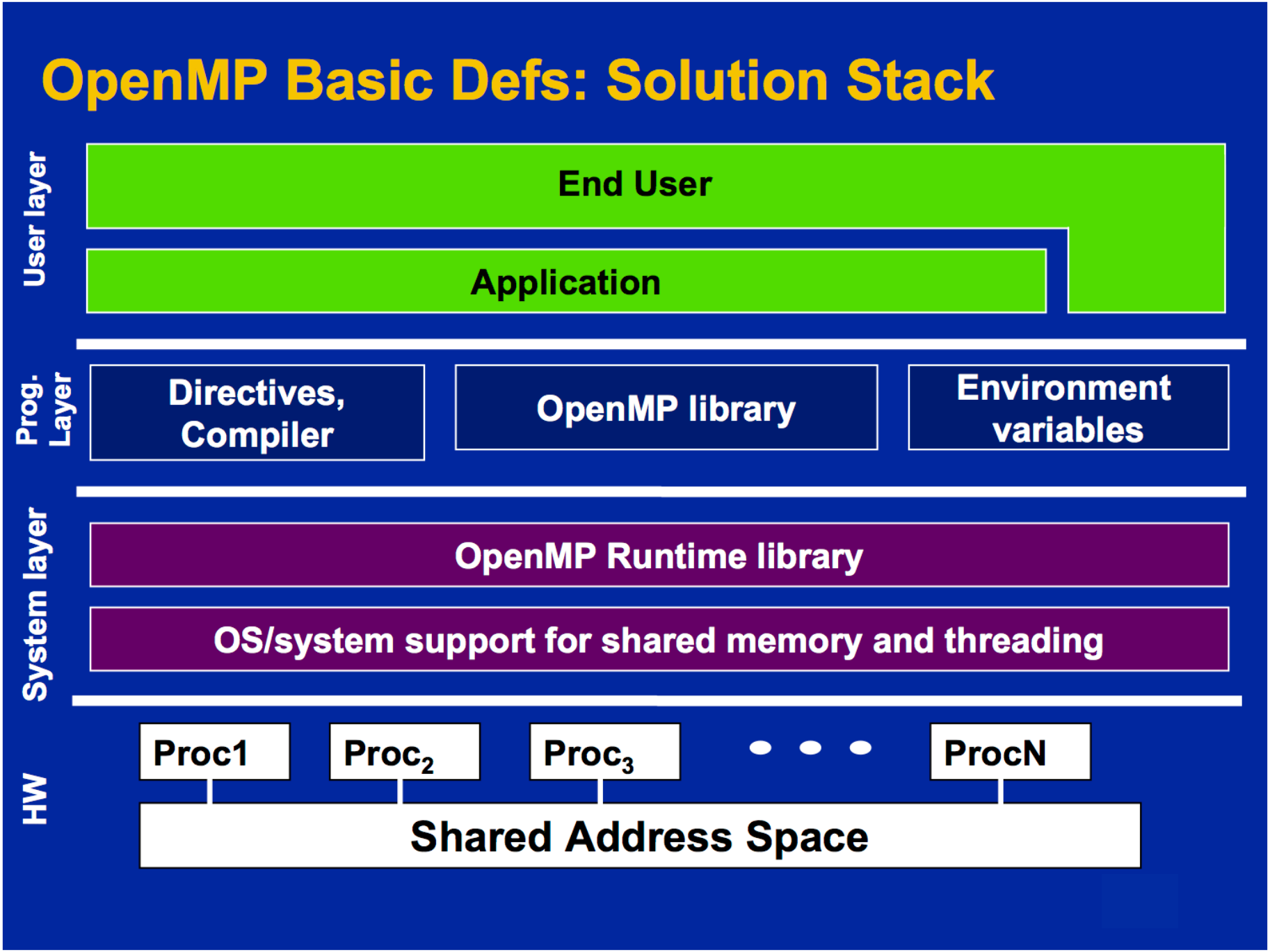
C$OMP DO lastprivate(XX)

Nthrds = OMP_GET_NUM_PROCS()

omp_set_lock(lck)

* The name "OpenMP" is the property of the OpenMP Architecture Review Board.

OpenMP Basic Defs: Solution Stack



OpenMP core syntax

- Most of the constructs in OpenMP are compiler directives.

  `#pragma omp construct [clause [clause]  ]`

  - Example

    `#pragma omp parallel num_threads(4)`

- Function prototypes and types in the file:

  `#include <omp.h>`

- Most OpenMP* constructs apply to a "structured block".
  - Structured block: a block of one or more statements with one point of entry at the top and one point of exit at the bottom.
  - It's OK to have an exit() within the structured block.

Exercise 1, Part A: Hello world

Verify that your environment works

- **Write a program that prints “hello world”.**

```
void main()
{


    int ID = 0;

    printf(“ hello(%d) ”, ID);
    printf(“ world(%d) \n”, ID);

}
```

Exercise 1, Part B: Hello world

Verify that your OpenMP environment works

- Write a multithreaded program that prints “hello world”.

```
#include “omp.h”
void main()
{
 #pragma omp parallel
 {
    int ID = 0;
    printf(“ hello(%d) ”, ID);
    printf(“ world(%d) \n”, ID);
 }
}
```

Switches for compiling and linking

-fopenmp gcc

-mp pgi

/Qopenmp intel

Exercise 1: Solution
A multi-threaded "Hello world" program

- **Write a multithreaded program where each thread prints "hello world".**

```
#include "omp.h"
void main()
{
#pragma omp parallel
 {
    int ID = omp_get_thread_num();
    printf(" hello(%d) ", ID);
    printf(" world(%d) \n", ID);
  }
}
```

OpenMP include file

Parallel region with default number of threads

End of the Parallel region

Runtime library function to return a thread ID.

Sample Output:

hello(1) hello(0) world(1)

world(0)

hello (3) hello(2) world(3)

world(2)

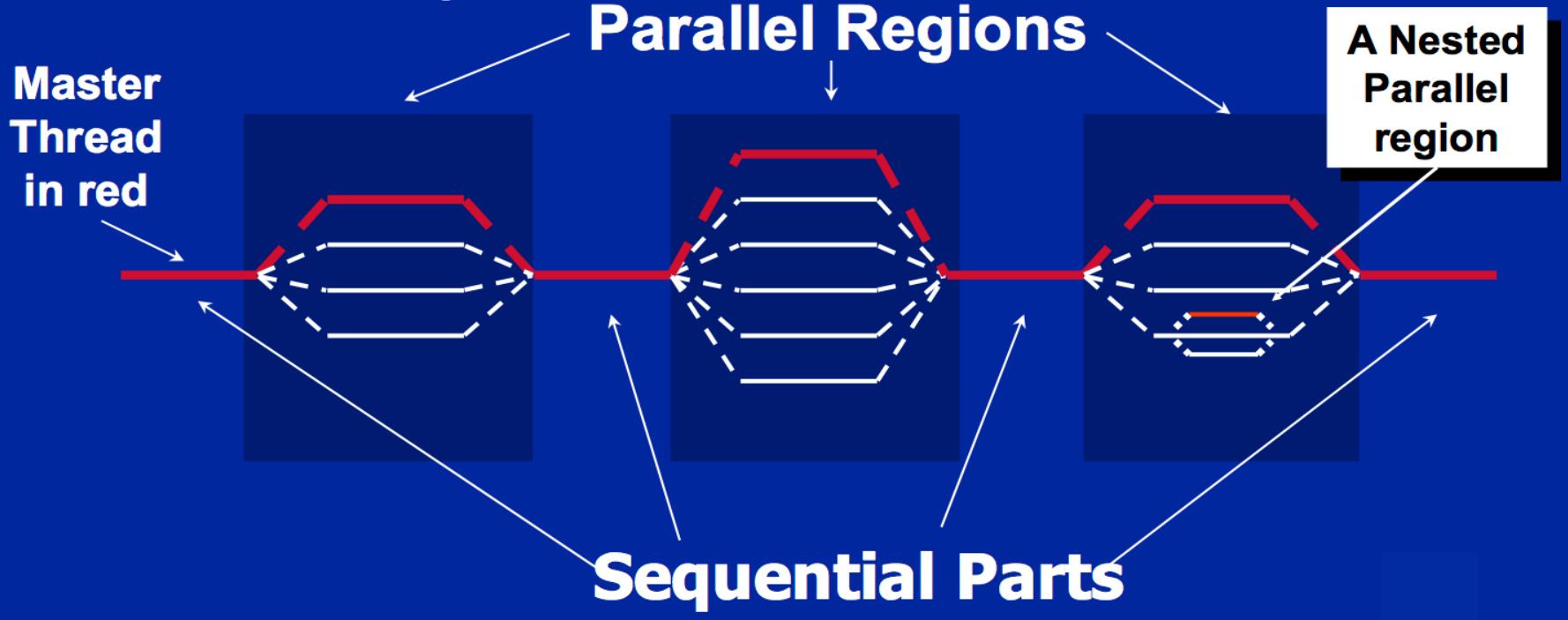
OpenMP Overview:
How do threads interact?

- OpenMP is a multi-threading, shared address model.
  - Threads communicate by sharing variables.
- Unintended sharing of data causes race conditions:
  - race condition: when the program's outcome changes as the threads are scheduled differently.
- To control race conditions:
  - Use synchronization to protect data conflicts.
- Synchronization is expensive so:
  - Change how data is accessed to minimize the need for synchronization.

OpenMP Programming Model:

Fork-Join Parallelism:

- Master thread spawns a team of threads as needed.
- Parallelism added incrementally until performance goals are met: i.e. the sequential program evolves into a parallel program.



Thread Creation: Parallel Regions

- You create threads in OpenMP* with the parallel construct.
- For example, To create a 4 thread Parallel region:

```
double A[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
        int ID = omp_get_thread_num();
        pooh(ID,A);
}
```

Runtime function to request a certain number of threads

Each thread executes a copy of the code within the structured block

Runtime function returning a thread ID

- Each thread calls `pooh(ID,A)` for `ID = 0 to 3`

Thread Creation: Parallel Regions

- You create threads in OpenMP* with the parallel construct.
- For example, To create a 4 thread Parallel region:

```
double A[1000];

#pragma omp parallel num_threads(4)
{
        int ID = omp_get_thread_num();
        pooh(ID,A);
}
```

clause to request a certain number of threads

Each thread executes a copy of the code within the structured block

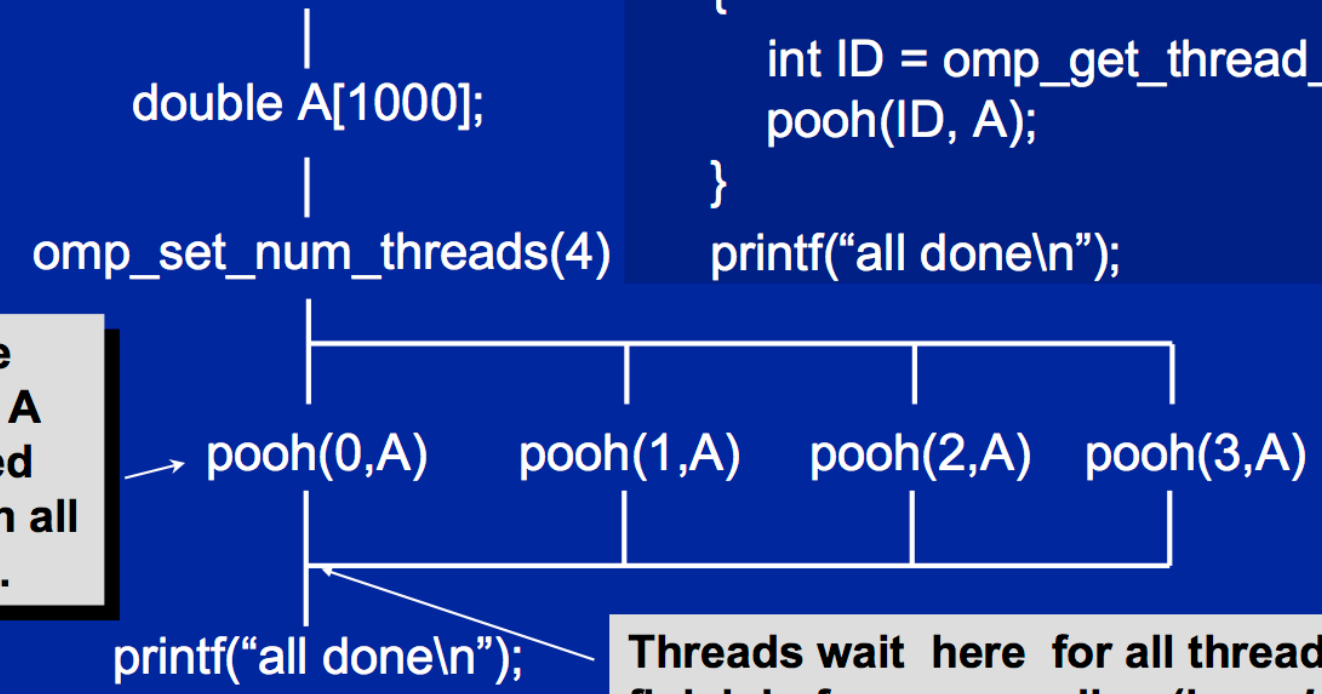
Runtime function returning a thread ID

- Each thread calls `pooh(ID,A)` for `ID = 0 to 3`

Thread Creation: Parallel Regions example

- **Each thread executes the same code redundantly.**

```
double A[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
    int ID = omp_get_thread_num();
    pooh(ID, A);
}
printf("all done\n");
```



A single copy of A is shared between all threads.

Threads wait here for all threads to finish before proceeding (i.e. a *barrier*)

* The name "OpenMP" is the property of the OpenMP Architecture Review Board

How is OpenMP typically used?

- OpenMP is usually used to parallelize loops:
  - Find your most time consuming loops.
  - Split them up between threads.

Sequential Program

```
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i]
  }
}
```

Parallel Program

```
#include "omp.h"
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
#pragma omp parallel for
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i];
  }
}
```

How is OpenMP typically used? (Cont.)

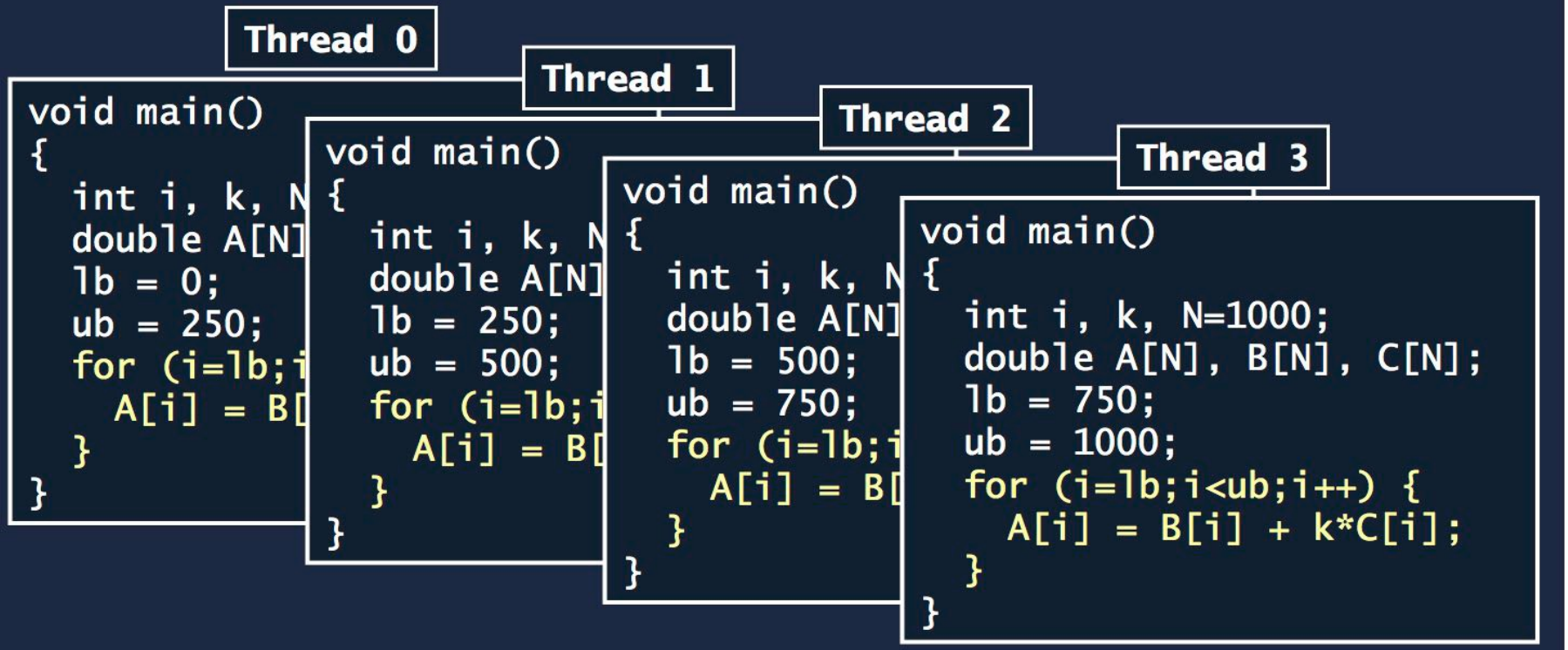
- Single Program Multiple Data (SPMD)

Parallel Program

```
#include "omp.h"
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
#pragma omp parallel for
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i];
  }
}
```

How is OpenMP typically used? (Cont.)

- Single Program Multiple Data (SPMD)



Next Class

- OpenMP work-sharing pargmas

Reading Material

- OpenMP tutorial from LLNL
  – https://computing.llnl.gov/tutorials/openMP/