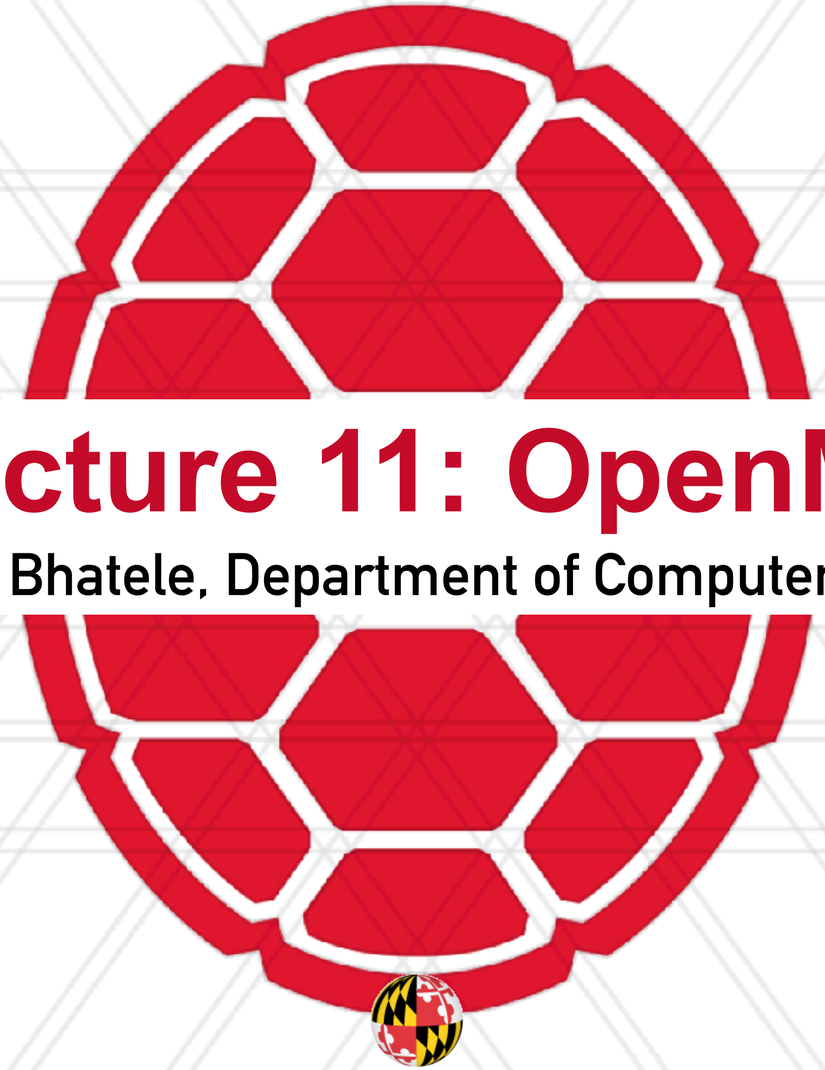
Introduction to Parallel Computing (CMSC498X / CMSC818X)



Lecture 11: OpenMP

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Assignment 2 has been posted
- Deadline: October 19, 11:59 pm AoE

Shared memory programming

- All entities (threads) have access to the entire address space
- Threads "communicate" or exchange data by sharing variables
- User has to manage data conflicts

OpenMP

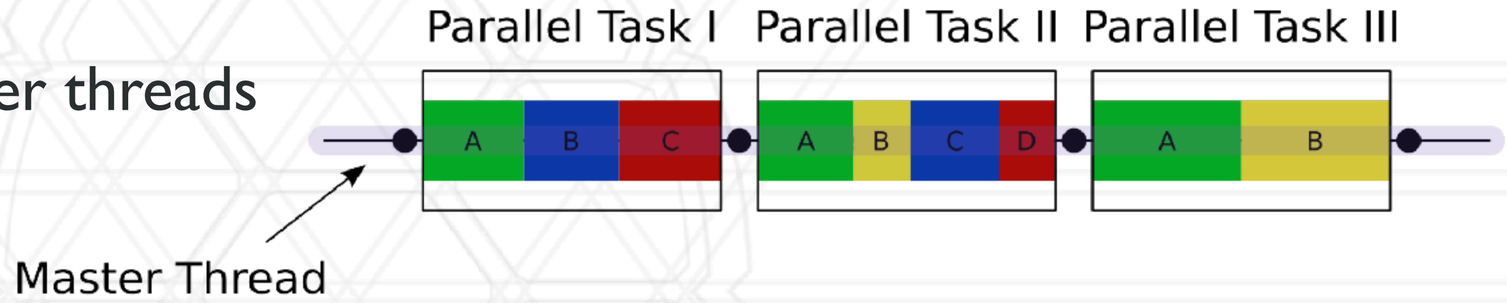
- OpenMP is an example of a shared memory programming model
- Provides on-node parallelization
- Meant for certain kinds of programs/computational kernels
  - That use arrays and loops
- Hopefully easy to implement in parallel with small code changes

OpenMP

- OpenMP is a language extension that enables parallelizing C/C++/Fortran code
- Programmer uses compiler directives and library routines to indicate parallel regions in the code
- Compiler converts code to multi-threaded code
- Fork/join model of parallelism

Fork-join parallelism

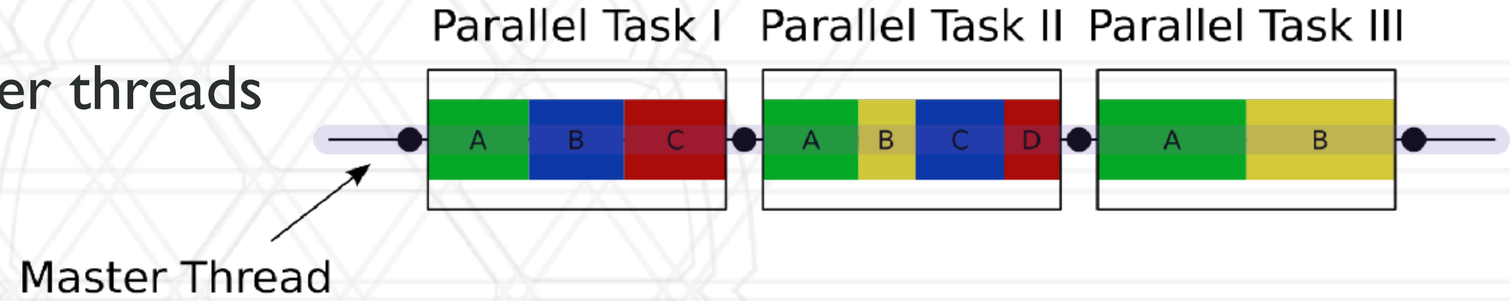
- Single flow of control
- Master thread spawns worker threads

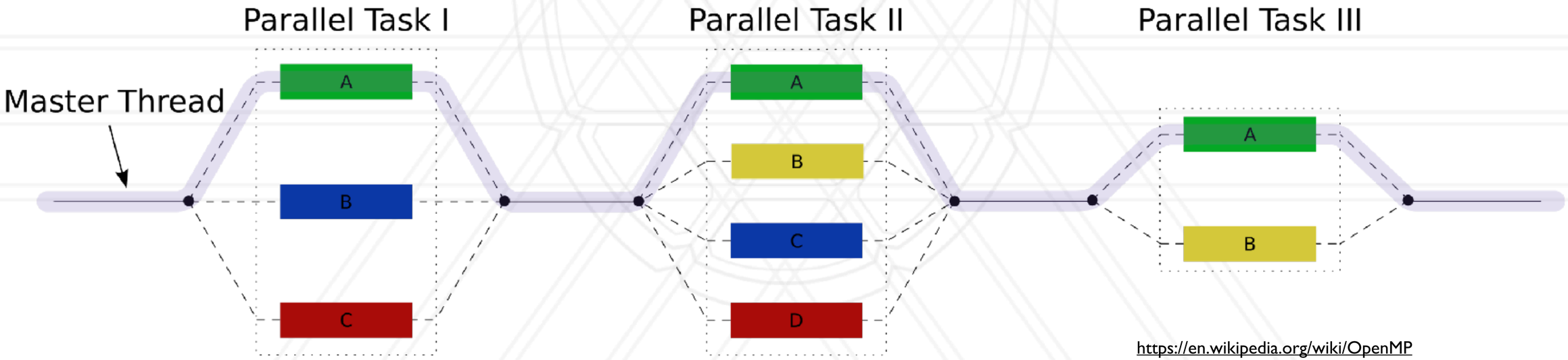


https://en.wikipedia.org/wiki/OpenMP

Fork-join parallelism

- Single flow of control
- Master thread spawns worker threads





https://en.wikipedia.org/wiki/OpenMP

Race conditions when threads interact

- Unintended sharing of variables can lead to race conditions
- Race condition: program outcome depends on the scheduling order of threads
- How can we prevent data races?
  - Use synchronization
  - Change how data is stored

OpenMP pragmas

- Pragma: a compiler directive in C or C++
- Mechanism to communicate with the compiler
- Compiler may ignore pragmas

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

Hello World in OpenMP

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

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

- Compiling: `gcc -fopenmp hello.c -o hello`
- Setting number of threads: `export OMP_NUM_THREADS=2`

Parallel for

- Directs the compiler that the immediately following for loop should be executed in parallel

```
#pragma omp parallel for [clause [clause] ... ]
for (i = init; test_expression; increment_expression) {
    ...
    do work
    ...
}
```

Parallel for example

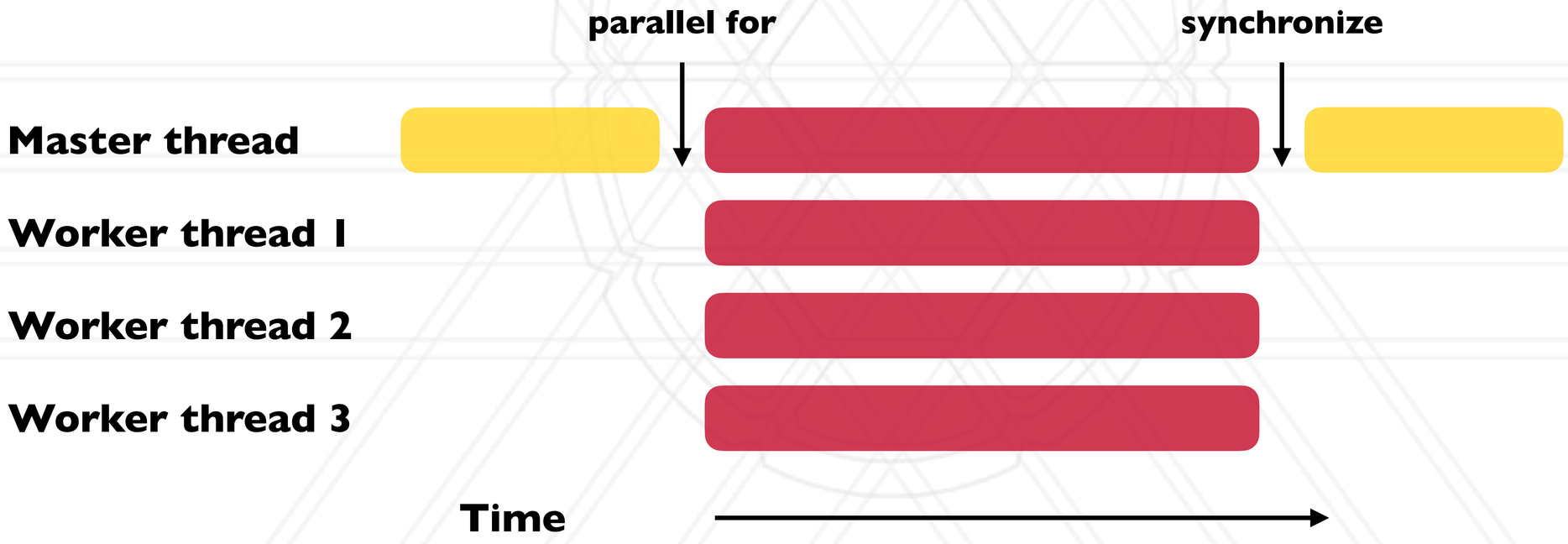
```
int main(int argc, char **argv)
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
    }

    return 0;
}
```

Parallel for execution

- Master thread creates worker threads
- All threads divide iterations of the loop among themselves



Number of threads

- Use environment variable

```
export OMP_NUM_THREADS=X
```

- Use omp_set_num_threads(int num_threads)
  - Set the number of OpenMP threads to be used in parallel regions
- `int omp_get_num_procs(void);`
  - Returns the number of available processors
  - Can be used to decide the number of threads to create

Loop scheduling

- Assignment of loop iterations to different worker threads
- Default schedule tries to balance iterations among threads
- User-specified schedules are also available

Data sharing defaults

- Most variables are shared by default
- Global variables are shared
- Exception: loop index variables are private by default
- Stack variables in function calls from parallel regions are also private to each thread (thread-private)

UNIVERSITY OF
MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu