High Performance Computing Systems (CMSC714)

Lecture 5: OpenMP

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Reading assignments are on the website:
  - Lead presenter should upload their slides (pdf, <15 minutes) on ELMS
  - Other designated readers should upload a pdf with short summary and 2-3 questions to ELMS
  - Due at 6:00 PM the day before class
- Assignment 1 on MPI is posted and is due on February 22

Summary of last lecture

- Non-blocking point-to-point operations
- Collective operations
- Timing MPI programs
- Other send modes and MPI protocols

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

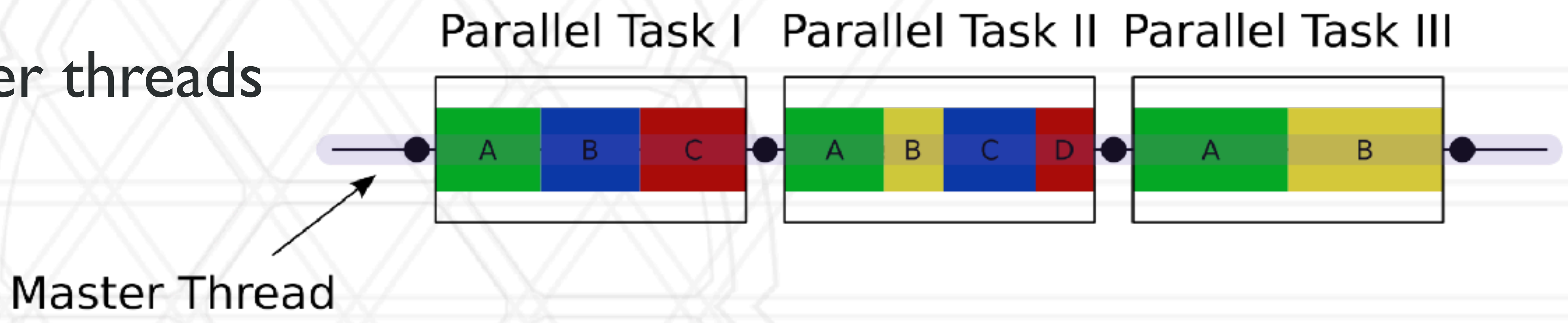
- OpenMP is an example of a shared memory programming model
- Provides on-node parallelization
- Meant for certain kinds of programs/computational kernels
  - That use arrays and loops
- Hopefully easy to implement in parallel with small code changes

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

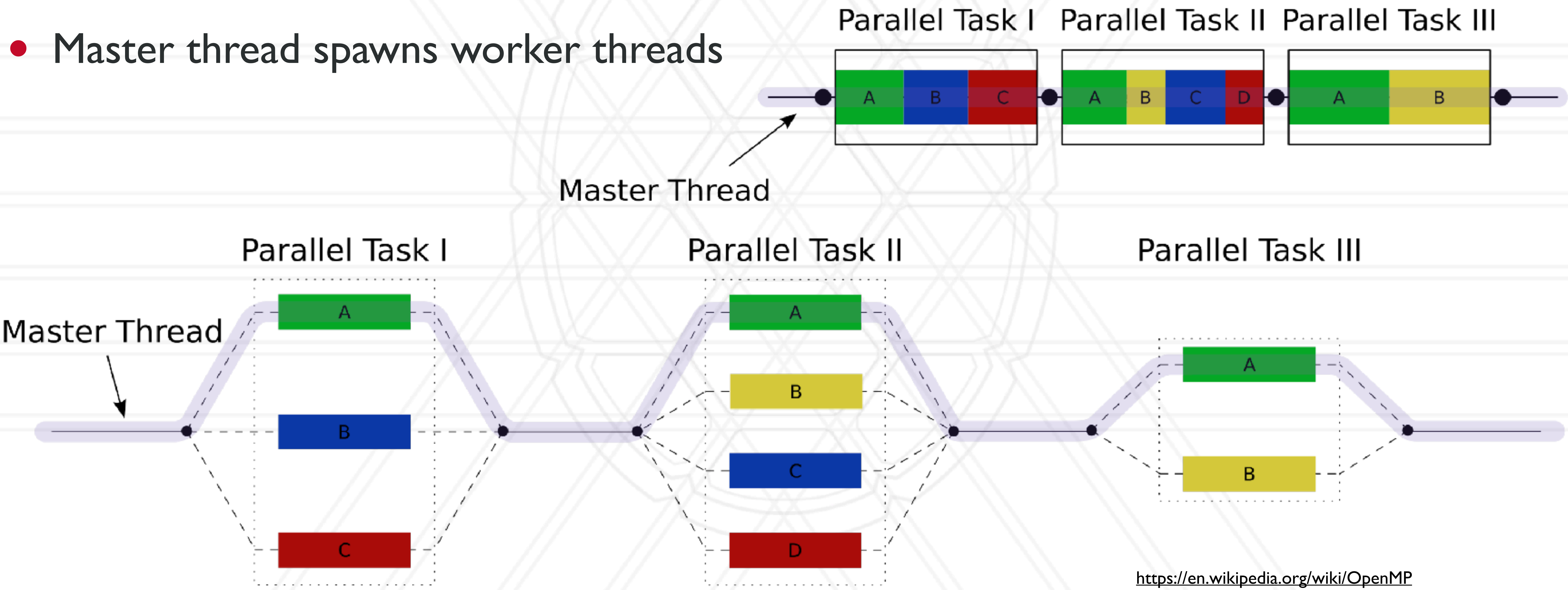
- Single flow of control
- Master thread spawns worker threads



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

- saxpy (single precision a*x+y) example

```
int main(int argc, char **argv)
{
    ...

    for (int i = 0; i < n; i++) {
        z[i] = a * x[i] + y[i];
    }

    ...
}
```

Parallel for example

- saxpy (single precision a*x+y) example

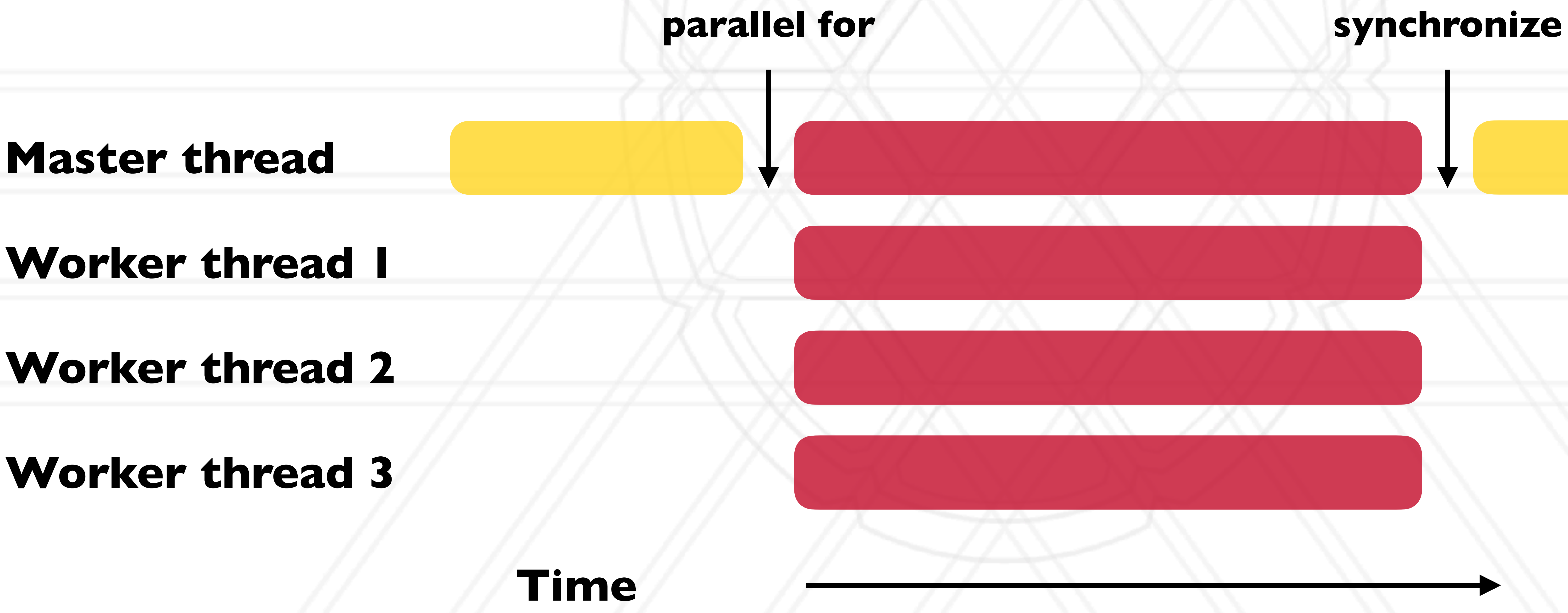
```
int main(int argc, char **argv)
{
    ...

    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        z[i] = a * x[i] + y[i];
    }

    ...
}
```

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

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)

Overriding defaults using clauses

- Specify how data is shared between threads executing a parallel region
- `private(list)`
- `shared(list)`
- `default(shared | none)`
- `reduction(operator: list)`
- `firstprivate(list)`
- `lastprivate(list)`

https://www.openmp.org/spec-html/5.0/openmpsu106.html#x139-5540002.19.4

firstprivate clause

- Initializes each thread's private copy to the value of the master thread's copy

```
val = 5;

#pragma omp parallel for firstprivate(val)
for (int i = 0; i < n; i++) {
    ... = val + 1;
}
```

lastprivate clause

- Writes the value belonging to the thread that executed the last iteration of the loop to the master's copy
- Last iteration determined by sequential order

lastprivate clause

- Writes the value belonging to the thread that executed the last iteration of the loop to the master's copy
- Last iteration determined by sequential order

```
#pragma omp parallel for lastprivate(val)
for (int i = 0; i < n; i++) {
    val = i + 1;
}

printf(“%d\n”, val);
```

reduction(operator: list) clause

- Reduce values across private copies of a variable
- Operators: +, -, *, &, |, ^, &&, ||, max, min

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    val += i;
}

printf("%d\n", val);
```

https://www.openmp.org/spec-html/5.0/openmpsu107.html#x140-5800002.19.5

reduction(operator: list) clause

- Reduce values across private copies of a variable
- Operators: +, -, *, &, |, ^, &&, ||, max, min

```
#pragma omp parallel for reduction(+: val)
for (int i = 0; i < n; i++) {
    val += i;
}

printf("%d\n", val);
```

https://www.openmp.org/spec-html/5.0/openmpsu107.html#x140-5800002.19.5

Loop scheduling

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

User-specified loop scheduling

- Schedule clause

```
schedule (type[, chunk])
```

- type: static, dynamic, guided, runtime
- static: iterations divided as evenly as possible (#iterations/#threads)
  - chunk < #iterations/#threads can be used to interleave threads
- dynamic: assign a chunk size block to each thread
  - When a thread is finished, it retrieves the next block from an internal work queue
  - Default chunk size = 1

Other schedules

- guided: similar to dynamic but start with a large chunk size and gradually decrease it for handling load imbalance between iterations
- auto: scheduling delegated to the compiler
- runtime: use the OMP_SCHEDULE environment variable

https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-scheduling.html

Calculate the value of $\pi = \int_0^1 \frac{4}{1+x^2}$

```
int main(int argc, char *argv[])
{
    ...

    n = 10000;

    h   = 1.0 / (double) n;
    sum = 0.0;

    for (i = 1; i <= n; i += 1) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    ...
}
```

Calculate the value of $\pi = \int_0^1 \frac{4}{1+x^2}$

```
int main(int argc, char *argv[])
{
    ...

    n = 10000;
    h   = 1.0 / (double) n;
    sum = 0.0;

    #pragma omp parallel for private(x) reduction(+: sum)
    for (i = 1; i <= n; i += 1) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    ...
}
```

Parallel region

- All threads execute the structured block

```
#pragma omp parallel [clause [clause] ... ]
    structured block
```

- Number of threads can be specified just like the parallel for directive

Synchronization

- Concurrent access to shared data may result in inconsistencies
- Use mutual exclusion to avoid that
- critical directive
- atomic directive
- Library lock routines

https://software.intel.com/content/www/us/en/develop/documentation/advisor-user-guide/top/appendix/adding-parallelism-to-your-program/replacing-annotations-with-openmp-code/adding-openmp-code-to-synchronize-the-shared-resources.html

Questions?

UNIVERSITY OF MARYLAND

Abhinav Bhatele

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

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