Announcements

- Use office hours

- If you foresee not being able to complete assignments for a valid reason, email me asap instead of after the deadline
saxpy (single precision a*x+y) example

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

```c
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    z[i] = a * x[i] + y[i];
}
```
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](https://www.openmp.org/spec-html/5.0/openmpsu106.html#x139-5540002.19.4)
private clause

- Each thread has its own copy of the variables in the list
- Private variables are uninitialized when a thread starts
- The value of a private variable is unavailable to the master thread after the parallel region has been executed
default clause

- Determines the data sharing attributes for variables for which this would be implicitly determined otherwise
Anything wrong with this example?

val = 5;

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


Anything wrong with this example?

```c
val = 5;

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

The value of `val` will not be available to threads inside the loop.
Anything wrong with this example?

```c
#pragma omp parallel for private(val)
for (int i = 0; i < n; i++) {
    val = i + 1;
}
printf("%d\n", val);
```
Anything wrong with this example?

```c
#include <stdio.h>

#define N 10

int main() {
    int val;
    #pragma omp parallel for private(val)
    for (int i = 0; i < N; i++) {
        val = i + 1;
    }
    printf("%d\n", val);
    return 0;
}
```

The value of `val` will not be available to the master thread outside the loop.
firstprivate clause

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

```c
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

```c
#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

```c
#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
User-specified loop scheduling

• Schedule clause

  \[ \text{schedule (type[, chunk])} \]

• 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

Calculate the value of $\pi = \int_{0}^{1} \frac{4}{1 + x^2}$

```c
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 \( \pi = \int_{0}^{1} \frac{4}{1 + x^2} \)

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

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

    #pragma omp parallel for firstprivate(h) 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

```c
#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
