COSC 462

OpenMP Basics: Directive Syntax

Piotr Luszczek

Example: Largest Value (OpenMP)

#pragma's are compiler directives and are **not** like preprocessor directives such as #include or #ifdef

OpenMP created a namespace: omp

OpenMP allows parallel processing inside "parallel" regions

```
#pragma omp parallel for reduction(+:sum)
for (int i=0; i<N; ++i)
  sum += X[i];
```

Reductions need a target variable

OpenMP takes care of dividing problem size N between threads.

OpenMP #pragma's are most often applied to loops

OpenMP has fast built-in reductions based on arithmetic, bitwise, and logical operators

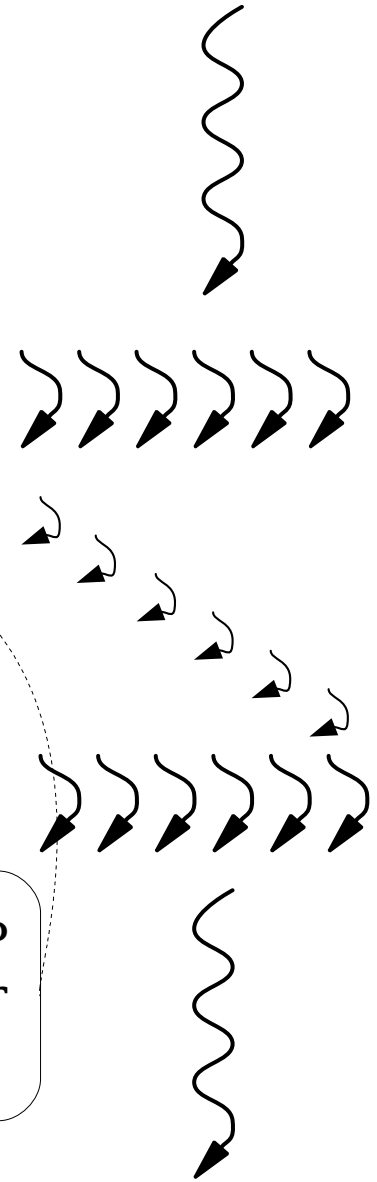
Separate OpenMP Pragma's

#pragma parallel
marks a parallel region

```
#pragma omp parallel
{ // begin of parallel region

#pragma omp for reduction(+:sum)
for (int i=0; i<N; ++i)
  sum += X[i];

} // end of parallel region
```



There are a few OpenMP pragma's that may occur inside a parallel region.

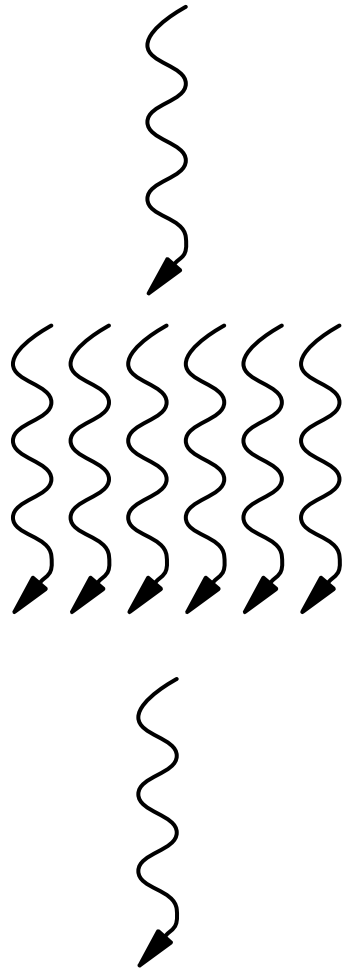
Running Multiple Threads

Parallel regions don't have to use other OpenMP pragmas.

```
#pragma omp parallel
{ // begin of parallel region

    printf( "Hello world!\n" );

} // end of parallel region
```



```
Hello world!
Hello world!
Hello world!
...
```

The output may be **scrambled** but each thread will print once.

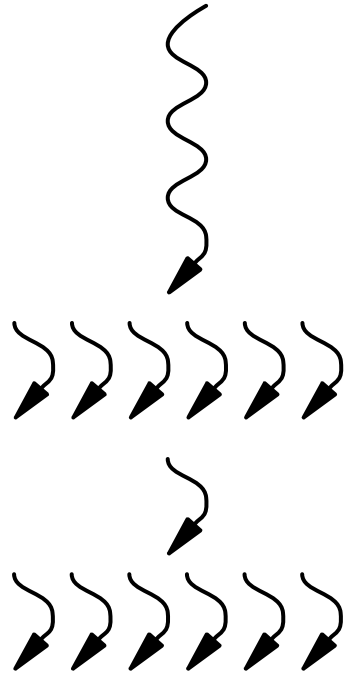
Dealing with I/O: One Thread for I/O

```
#pragma omp parallel
{ // begin of parallel region

    #pragma omp single
    printf( “Hello world!\n” );

} // end of parallel region
```

```
Hello world!
```



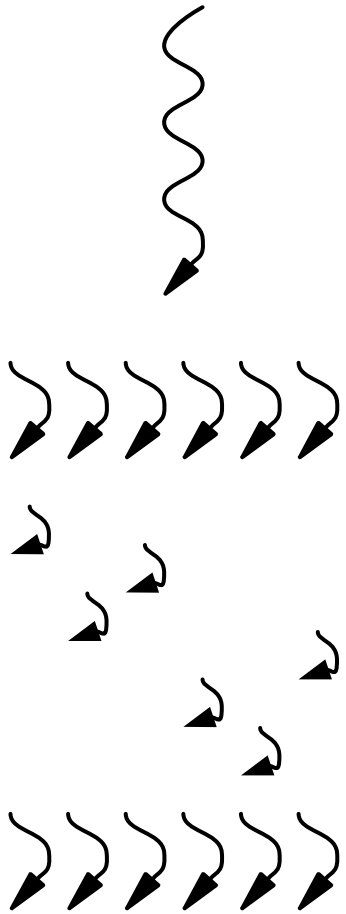
The output will not be **scrambled** and a single thread will print once at some point in time.

Dealing with I/O: Mutual Exclusion

```
#pragma omp parallel
{ // begin of parallel region

    #pragma omp critical
    {
    printf( “Hello world!\n” );
    fflush( stdout );
    }

} // end of parallel region
```



```
Hello world!
Hello world!
Hello world!
...
```

The output will not be **scrambled** and each thread will print once in some order.

OpenMP Implementation Outline

- OpenMP runtime library is implemented using low-level primitives
  - POSIX threads
  - WinThreads
- OpenMP-aware compiler assists in inserting additional code that invokes the OpenMP runtime library
  - At start of the program: initialize the library
    - Just like MPI_Init() initializes MPI
- Every parallel region needs additional code from the compiler, either:
  - Call a separate function with the code inside the region
  - Bring all the threads into the function through long_jump() set_jump()

Two Possible Implementations

```
#pragma parallel
```

- Generated code with separate function:

```
local_args.N = N;
parallelRegion01(
   localArgs);
```

- Generated function:

```
static void
parallelRegion01(
struct Arg local_args)
{
   for(i=0;
      i<localArgs.N/T;
      ++i) ...
_omp_sum(local_sum,
&sum);
}
```

- Generated code with set_jump():

```
if (main_thread)
set_jmp(parallelReg01)
;

for (i=0;i<N/T;++i) {
  local_sum += X[i];
}
_omp_sum(local_sum,
&sum);

if (! main_thread)
long_jmp(threadsWait);
```