# CISC 372: Parallel Computing OpenMP

Stephen F. Siegel

Department of Computer and Information Sciences University of Delaware

・ロト ・日ト ・ヨト ・ヨト ・ヨー うへで

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで

▶ an API for shared memory, multi-threaded programming

- > an API for shared memory, multi-threaded programming
- $\blacktriangleright$  works with C, C++, or Fortran

- > an API for shared memory, multi-threaded programming
- $\blacktriangleright$  works with C, C++, or Fortran
- emphasizes incremental parallelization
  - start with the sequential program
  - add a little bit of parallelism at a time
  - see how it works, change it, add some more, ...
  - contrast with MPI "all or nothing" approach

- > an API for shared memory, multi-threaded programming
- works with C, C++, or Fortran
- emphasizes incremental parallelization
  - start with the sequential program
  - add a little bit of parallelism at a time
  - see how it works, change it, add some more, ...
  - contrast with MPI "all or nothing" approach
- programmer inserts directives and function calls into sequential program
  - in C, a directive is a pragma
  - stands for pragmatic information
  - a general way to pass additional information to the compiler in a form not supported by the C language

- a compiler that does not recognize a kind of pragma can just ignore it
- ▶ #pragma omp ...

- > an API for shared memory, multi-threaded programming
- works with C, C++, or Fortran
- emphasizes incremental parallelization
  - start with the sequential program
  - add a little bit of parallelism at a time
  - see how it works, change it, add some more, ...
  - contrast with MPI "all or nothing" approach
- programmer inserts directives and function calls into sequential program
  - in C, a directive is a pragma
  - stands for pragmatic information
  - a general way to pass additional information to the compiler in a form not supported by the C language

◆□▶ ◆□▶ ◆三▶ ◆三▶ 三三 - つへぐ

- a compiler that does not recognize a kind of pragma can just ignore it
- ▶ #pragma omp ...

S.F. Siegel

- the sequential program remains embedded in the OpenMP version
  - just ignore the pragmas
  - replace function calls with trivial implementations

#### Sequential Dot Product (Chapman et al., Using OpenMP)

```
#include<stdio.h>
int main() {
  double sum, a[256], b[256];
  int status, i, n=256;
  for (i = 0; i < n; i++) {
    a[i] = i * 0.5;
   b[i] = i * 2.0;
 3
  sum = 0:
 for (i = 0; i < n; i++) {
    sum = sum + a[i]*b[i]:
 }
 printf("sum = \%f \n", sum);
3
```

### Dot Proudct in OpenMP (Chapman et al., Using OpenMP)

```
#include<stdio.h>
int main() {
  double sum, a[256], b[256];
  int status, i, n=256;
  for (i = 0; i < n; i++) {
    a[i] = i * 0.5;
   b[i] = i * 2.0;
 3
 sum = 0:
#pragma omp parallel for reduction(+:sum)
 for (i = 0; i < n; i++) {
    sum = sum + a[i]*b[i]:
 }
 printf("sum = \%f \n", sum);
```

most directives are applied to the following structured block S

- most directives are applied to the following structured block S
- S may be almost any kind of statement

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)
  - S must have single point of entry and single point of exit

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)
  - S must have single point of entry and single point of exit
  - may be a loop
    - if enclosing in curly braces would be a structured block

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)
  - S must have single point of entry and single point of exit
  - may be a loop
    - if enclosing in curly braces would be a structured block
  - may be an *if* statement
    - if enclosing in curly braces would be a structured block

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)
  - S must have single point of entry and single point of exit
  - may be a loop
    - if enclosing in curly braces would be a structured block
  - may be an *if* statement
    - if enclosing in curly braces would be a structured block
- ▶ in C, the directives have the form **#pragma** omp ...

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)
  - S must have single point of entry and single point of exit
  - may be a loop
    - if enclosing in curly braces would be a structured block
  - may be an *if* statement
    - if enclosing in curly braces would be a structured block
- ▶ in C, the directives have the form **#pragma** omp ...
- you can put non-newline white space before or after the #

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)
  - S must have single point of entry and single point of exit
  - may be a loop
    - if enclosing in curly braces would be a structured block
  - may be an *if* statement
    - if enclosing in curly braces would be a structured block

- ▶ in C, the directives have the form **#pragma** omp ...
- you can put non-newline white space before or after the #
- the directive terminates with the end of the logical line

- most directives are applied to the following structured block S
- S may be almost any kind of statement
  - a compound statement {...} (this is most common)
  - S must have single point of entry and single point of exit
  - may be a loop
    - if enclosing in curly braces would be a structured block
  - may be an *if* statement
    - if enclosing in curly braces would be a structured block
- ▶ in C, the directives have the form **#pragma** omp ...
- you can put non-newline white space before or after the #
- the directive terminates with the end of the logical line
- Iong directives can be spread over multiple physical lines by ending each physical line but the last with \

5

 in C, these physical lines are merged into one logical line at a very early stage of compilation (before preprocessing) Spreading a logical line over multiple physical lines

```
#pragma omp this is my really big long \
    pragma that keeps going and going and \
    going on and on and on and on and on \
    and on and on
    for (i=0; i<n; i++) {
        ...
}</pre>
```

Beware: You cannot have any white space after the  $\$ . It must be the last character on the physical line.

use gcc or clang, add flag -fopenmp; everything else the same gcc -fopenmp -o dot dot.c ./dot

use gcc or clang, add flag -fopenmp; everything else the same

```
gcc -fopenmp -o dot dot.c
./dot
```

- without the flag -fopenmp
  - header file omp.h will not necessarily be found
  - pragmas will just be ignored; program will be sequential

use gcc or clang, add flag -fopenmp; everything else the same

```
gcc -fopenmp -o dot dot.c
./dot
```

- without the flag -fopenmp
  - header file omp.h will not necessarily be found
  - pragmas will just be ignored; program will be sequential
- Apple users
  - ▶ for reasons that escape me, Apple's version of clang does not have OpenMP support
  - advice: install clang yourself using MacPorts
    - sudo port install clang-9.0, or later
    - then use clang-mp-9.0

use gcc or clang, add flag -fopenmp; everything else the same

```
gcc -fopenmp -o dot dot.c
./dot
```

- without the flag -fopenmp
  - header file omp.h will not necessarily be found
  - pragmas will just be ignored; program will be sequential
- Apple users
  - ▶ for reasons that escape me, Apple's version of clang does not have OpenMP support
  - advice: install clang yourself using MacPorts
    - sudo port install clang-9.0, or later
    - then use clang-mp-9.0
- general: all compilers
  - the preprocessor object macro \_OPENMP is defined iff you are running the compiler with OpenMP support
    - $\blacktriangleright$  value is yyyymm, where yyyy is the year of the Standard supported, and mm is the month
    - permits things like #ifdef \_OPENMP ... #else ...

▲□▶ ▲圖▶ ▲国▶ ▲国▶ - 国 - のへで

```
#pragma omp parallel [clauses]
S
```

```
#pragma omp parallel [clauses]
S
```

▶ a program begins execution with one thread

```
#pragma omp parallel [clauses]
S
```

- a program begins execution with one thread
- executing a parallel directive creates a parallel region

▲□▶ ▲□▶ ▲□▶ ▲□▶ ▲□ ● のへで

```
#pragma omp parallel [clauses]
S
```

- a program begins execution with one thread
- executing a parallel directive creates a parallel region
- when control enters the region, a team of threads is created
  - the team includes the original thread, known as the master thread

```
#pragma omp parallel [clauses]
S
```

- a program begins execution with one thread
- executing a parallel directive creates a parallel region
- when control enters the region, a team of threads is created
  - the team includes the original thread, known as the master thread
- all of the threads in the team execute the statement S concurrently

```
#pragma omp parallel [clauses]
S
```

- a program begins execution with one thread
- executing a parallel directive creates a parallel region
- when control enters the region, a team of threads is created
  - the team includes the original thread, known as the master thread
- all of the threads in the team execute the statement S concurrently
- S is typically a big compound statement

```
#pragma omp parallel [clauses]
S
```

- a program begins execution with one thread
- executing a parallel directive creates a parallel region
- when control enters the region, a team of threads is created
  - the team includes the original thread, known as the master thread
- all of the threads in the team execute the statement S concurrently
- S is typically a big compound statement
- additional directives inside S control how threads in the team behave

```
#pragma omp parallel [clauses]
S
```

- a program begins execution with one thread
- executing a parallel directive creates a parallel region
- when control enters the region, a team of threads is created
  - the team includes the original thread, known as the master thread
- > all of the threads in the team execute the statement S concurrently
- S is typically a big compound statement
- > additional directives inside S control how threads in the team behave
- at the end of S there is an implicit barrier
  - all threads join up at this point
  - all threads other than the master essentially disappear

```
#pragma omp parallel [clauses]
S
```

- a program begins execution with one thread
- executing a parallel directive creates a parallel region
- when control enters the region, a team of threads is created
  - the team includes the original thread, known as the master thread
- > all of the threads in the team execute the statement S concurrently
- S is typically a big compound statement
- > additional directives inside S control how threads in the team behave
- at the end of S there is an implicit barrier
  - all threads join up at this point
  - all threads other than the master essentially disappear
- the master continues execution

#### hello1.c: parallel directive example

```
#include <stdio.h>
int main () {
    printf("I am the master.\n"); // just the master
#pragma omp parallel
    {
        printf("Hello, world.\n"); // all threads
    } /* end of parallel region */
    printf("Goodbye, world.\n"); // just the master
}
```

#### hello1.c: parallel directive example

```
#include <stdio.h>
int main () {
    printf("I am the master.\n"); // just the master
#pragma omp parallel
    {
        printf("Hello, world.\n"); // all threads
    } /* end of parallel region */
    printf("Goodbye, world.\n"); // just the master
}
```

```
omp$ cc -fopenmp hello1.c
omp$ ./a.out
I am the master.
Hello, world.
Hello, world.
Goodbye, world.
omp$
```

9

S.F. Siegel  $\diamond$  CISC 372: Parallel Computing  $\diamond$  OpenMP

need to #include <omp.h>

- need to #include <omp.h>
- int omp\_get\_num\_threads()

returns the number of threads in the team in the current region

- need to #include <omp.h>
- int omp\_get\_num\_threads()
  - returns the number of threads in the team in the current region
- int omp\_get\_thread\_num()
  - returns the ID of the calling thread
  - threads within a team are numbered  $0, 1, \ldots$
  - master thread is always thread 0

▲□▶▲□▶▲□▶▲□▶ □ クタウ

- need to #include <omp.h>
- int omp\_get\_num\_threads()
  - returns the number of threads in the team in the current region
- int omp\_get\_thread\_num()
  - returns the ID of the calling thread
  - threads within a team are numbered 0, 1, ...
  - master thread is always thread 0
- omp\_get\_wtime()
  - returns the wall clock time (like MPI\_Wtime)