CISC 372: Parallel Computing
OpenMP

Stephen F. Siegel

Department of Computer and Information Sciences
University of Delaware

OpenMP Overview

- 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 ...`
- 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;
  }
  sum = 0;
  for (i = 0; i < n; i++) {
    sum = sum + a[i]*b[i];
  }
  printf("sum = %f \n", sum);
}
```

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;
  }
  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);
}
```

Basic Syntactic Concepts

- 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
- long directives can be spread over multiple physical lines by ending each physical line but the last with `\`
  - 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++) {
  ...
}
```

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

Compiling and runnning an OpenMP program

- 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
    - value is $yyyymm$, where $yyyy$ is the year of the Standard supported, and mm is the month
    - permits things like `#ifdef _OPENMP ... #else ...`

The `parallel` directive

```
#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
}
```

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

Basic OMP functions

- 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
- `omp_get_wtime()`
  - returns the wall clock time (like `MPI_Wtime`)