COMP 4007: Parallel Processing and Computer Architecture

Tutorial 1:
Programming with OpenMP

TA: Wenxiang Lin (869259303@qq.com)

Contents

- Part 1: Programming environment setup
  - UNIX Basics / vi editor

- Part 2: Quick start on C/C++ programming
  - Demo / Compile / Debug / Run

- Part 3: Programming with OpenMP
  - Demo / Compile / Debug / Run

Part 1:
Programming environment setup

UNIX Basics / vi editor

UNIX Basics

- Basic command
    - ls
        - "ls -a" : show hidden files or directories
        - "ls -l" : list in long listing format
        - "ls -al"
    - cd
        - "cd ~" or "cd ": change to home directory
        - "cd .." : change to parent directory
    - mkdir, rmdir, mv
    - rm
        - "rm -r <directory>" : remove the contents of <directorie> recursively
    - cp: copy a file
        - "cp sourcefile destination"
    - tab: complete filename or
        - "cd + tab"

UNIX Basics

- Practice
  - Create a comp4007 directory for lab1 under home directory

    cd ~

    mkdir comp4007

    cd comp4007

    mkdir lab01

    cd lab01

UNIX Basics

- Cat
  - cat "filename" : display content of a file
  - cat > "filename" : create and append content to a file
  - cat >> "filename" : append content at the end of a file
- Practice
  - cat main.c // show you the content of main.c
  - cat > helloworld.txt (enter)
    - type "Hello World" (Enter) **(Ctrl + D)**
  - cat helloworld.txt

UNIX Basics

- Get help information to see how to use UNIX command
    - rm --help
    - cp --help | more
    - cat –help

Vi editor – Starting vi

- Vi - a text base editor under Unix
- Starting vi
  - vi "filename" – Start at line 1 of file
  - vi +n "filename" – start at line n of file
  - vi + "filename" – start at last line of file
  - vi –r "filename" – recover file after a system crash
- Two modes in vi
  - Insertion mode : press "i" or "I" enter this mode
  - Command mode : press "Esc" enter this mode

Vi editor – Saving files and leaving vi

- Saving files
  - :e "filename" – edit other file
  - :w – save current editing file
  - :w "filename" – save as file (vi will not switch to edit the saved file afterward)
  - :w! "filename" – save as existing file
- Leaving vi
  - :q – quit vi
  - :wq – save file and quit vi
  - :q! – quit vi without saving
- Copy : 'yy'
- Paste: 'p'
- Cut: 'cc'

Vi editor – commands

- Moving cursor
- Inserting text
- Changing and replacing text
- Deleting text
- Markers
- Search and replace
- ……

Vi editor – Practice

- Practice
  - Open helloworld.txt
    - vi helloworld.txt (In command mode at the beginning)
    - (press "i" to enter insertion mode, you can edit the file now)
    - (press 'Esc' to enter command mode)
    - (press 'dd' to delete current line)
    - (press 'ndd' to delete n line below current line)
    - (press ':wq' in command mode to leave vi and save the file)

Part 2:
Quick start on C/C++ programming

Demo / Compile / Debug / Run

[Demo 1] Hello World

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

[Case study 1] Hello World

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

Step 1 Include headers if needed

#include <helper_1>

#include <helper_2>

…

<math.h>	Common mathematics functions
<setjmp.h>	Nonlocal jumps
<signal.h>	Signal handling
<stdalign.h> (since C11)	alignas and alignof convenience macros
<stdarg.h>	Variable arguments
<stdatomic.h> (since C11)	Atomic types
<stdbool.h> (since C99)	Boolean type
<stddef.h>	Common macro definitions
<stdint.h> (since C99)	Fixed-width integer types
<stdio.h>	Input/output
<stdlib.h>	General utilities: memory management, program utilities, string conversions, random numbers
<stdnoreturn.h> (since C11)	noreturn convenience macros
<string.h>	String handling

[2] C header files https://en.cppreference.com/w/c/header

[Case study 1] Hello World

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

Step 2 Craft your own innovations

Fundamental tips on syntax:

- Use semicolon ; to end a clause

- Use brackets ( ) to wrap a list of function arguments

- Use curly brackets { } to warp a code block

- Function signature:

  - type_qualifier of the return value

  - function name

  - a list of arguments

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Things you still need to know

1. What **data type**[3] you can use and when to declare a type

- the type `void`
- basic types
  - the type `char`
  - signed integer types
    - standard: `signed char`, `short`, `int`, `long`, `long long` (since C99)
    - extended: implementation defined, e.g. `__int128`
  - unsigned integer types
    - standard: `_Bool` (since C99), `unsigned char`, `unsigned short`, `unsigned int`, `unsigned long`, `unsigned long long` (since C99)
    - extended: implementation-defined, e.g. `__uint128`
  - floating types
    - real floating types: `float`, `double`, `long double`
    - complex types: `float _Complex`, `double _Complex`, `long double _Complex`
    - imaginary types: `float _Imaginary`, `double _Imaginary`, `long double _Imaginary`
- enumerated types
- derived types
  - array types
  - structure types
  - union types
  - function types
  - pointer types
  - atomic types

[3] https://en.cppreference.com/w/c/language/type

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Things you still need to know

1. What **data type** you can use and when to declare a type

2. What **operators** you can have

   - and their precedence and associativity[4]

3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	

[4] https://en.cppreference.com/w/c/language/operator_precedence

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Things you still need to know

1. What **data type** you can use and when to declare a type

2. What **operators** you can have

    *   and their precedence and associativity

3. What are the **scope (lifetime) and visibility** of your variables

    *   on code block basis; shadow principle

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Things you still need to know

1. What **data type** you can use and when to declare a type

2. What **operators** you can have

   - and their precedence and associativity

3. What are the **scope (lifetime) and visibility** of your variables

   - on code block basis; shadow principle

4. How to **format** a string for printf()[5]

   other examples:

```c
char ch = 'A';
printf("%c\n", ch);
```

A

```c
float a = 12.67;
printf("%f\n", a);
printf("%e\n", a);
```

12.670000
1.267000e+01

[5] http://www.cplusplus.com/reference/cstdio/printf/

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Of course, you also need to know about **pointers**—the spirit of C

- why pointers: arguments are passed by **value**

- what is a pointer: the **address** (in virtual memory) of a variable

- related syntax:
    - * (indirection/dereference opeartor):
        - ➢ when used in declaring a variable
            - indicating that the variable is a pointer

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Of course, you also need to know about **pointers**—the spirit of C

- why pointers: arguments are passed by **value**

- what is a pointer: the **address** (in virtual memory) of a variable

- related syntax:
  - * (indirection/dereference opeartor):
    - ➢ when used in declaring a variable
      - indicating that the variable is a pointer
    - ➢ when used as operators in an expression
      - referring to the value pointed by the pointer

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Of course, you also need to know about **pointers**—the spirit of C

- why pointers: arguments are passed by **value**

- what is a pointer: the **address** (in virtual memory) of a variable

- related syntax:
    - * (indirection/dereference opeartor):
        - ➤ when used in declaring a variable
            - indicating that the variable is a pointer
        - ➤ when used as operators in an expression
            - referring to the value pointed by the pointer

[Case study 2] Swapping Two Integers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Of course, you also need to know about **pointers**—the spirit of C

- why pointers: arguments are passed by **value**

- what is a pointer: the **address** (in virtual memory) of a variable

- related syntax:
  - * (indirection/dereference opeartor):
    - ➢ when used in declaring a variable
      - indicating that the variable is a pointer
    - ➢ when used as operators in an expression
      - referring to the value pointed by the pointer

  & (address-of opeartor):
    - fetching the address of a variable (resulting in a pointer)

[Demo 2] Swapping Two Intergers

```c
#include <stdio.h>

int outer_a = 7;
int b = 1;

void swapInt(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int a = outer_a;
    int b = 100 + 200 / 10 - 3 * 10;
    printf("Before: %d %d\n", a, b);
    swapInt(&a, &b);
    printf("After : %d %d\n", a, b);
    return 0;
}
```

Compile with GCC / G++

- Check your gcc / g++ version: gcc --version (-v) / g++ --version (-v)
- Basic option: gcc [source code files] –o [output file]

    e.g. gcc hello.c –o hello

- Specify external libraries (-l) to link

    e.g. gcc foo.c –o foo –l math

        "math" refers to the C math library, which gcc does not link to automatically

        it is used when we our code needs to includes <math.h>

- Generating information for debugging (-g)

    e.g. gcc buggy.c –o buggy –g

        (so that later you can execute "gdb swap")

Run C / C++ program

- Type ./[output file]

  e.g. ./hello

```
csl2wk07:yxingag:153> g++ -o hello hello.cpp
csl2wk07:yxingag:154> ./hello
Hello, world!
csl2wk07:yxingag:155> █
```

Using GDB

[Demo] Debugging buggy

```c
#include <stdio.h>

int main() {
    int arr[5];
    int i;
    for (i=0;i<5;i++)
        arr[i] = i;
    for (i=0;i<=5;i++)
        printf("%d\n", arr[i]);
    return 0;
}
```

Using GDB

- GDB stands for GNU Debugger

- Make sure that the file is compiled with "**-g**"

- Usage

  - Enter: gdb [program file]

  - Quit: quit or Ctrl+D

  - List program: list

  - Set breakpoint: break

  - Run until stop: run [args]

  - Print variable: print [variable name]

```
(gdb) l
1 #include <stdio.h>
2
3 int main()
4 {
5       int arr[5];
6       int i;
7       for (i=0;i<5;i++)
8           arr[i] = i;
9       for (i=0;i<=5;i++)

(gdb) break 9
Breakpoint 1 at 0x400544: file buggy.c,
line 9.

(gdb) run
Starting program:
/homes/cspeter/test/buggy


Breakpoint 1, main () at buggy.c:9
9       for (i=0;i<=5;i++)

(gdb) p arr
$1 = {0, 1, 2, 3, 4}
(gdb) p i
$2 = 5
```

Using GDB

- If no breakpoint is set, "run" will execute to the end

- Otherwise, it will stop at the first breakpoint it meets.

  - To continue, you have 3 options:

    - continue: GDB will continue executing until the next break point

    - next: GDB will execute the next line as a single instruction (even if it is a function call)

    - step: GDB will execute the next line, if it is a function call, it will step into it

Part 3:
Programming with OpenMP

Demo / Compile /Run / Debug / Environment variables

OpenMP Components

Directives

- ▸ Parallel regions
- ▸ Work sharing
- ▸ Synchronization
- ▸ Data-sharing attributes
  - ▸ Private
  - ▸ Firstprivate
  - ▸ Lastprivate
  - ▸ Shared
  - ▸ Reduction
- ▸ Orphaning

Runtime environment

- ▸ Number of threads
- ▸ Thread ID
- ▸ Dynamic thread adjustment
- ▸ Nested parallelism
- ▸ Timers
- ▸ API for locking

Environment variables

- ▸ Number of threads
- ▸ Scheduling type
- ▸ Dynamic thread adjustment
- ▸ Nested parallelism

Recall OpenMP Directives

#pragmas **omp parallel [clause[[,] clause] …]**

- Most basic parallel directive.
- The number of threads that run the following structured block of code is determined by the run-time system.
- Supports the following clauses
  - if (scalar expression)
  - private(list)
  - shared(list)
  - default(none/shared)
  - reduction(operator:list)
  - copyin(list)
  - firstprivate(list)
  - num_threads(scalar)

Demo 1 – Hello World

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>          ←──── Runtime header

void Hello(void);   /* Thread function */

int main(int argc, char* argv[]) {
    /* Get number of threads from command line */
    int thread_count = strtol(argv[1], NULL, 10);

#   pragma omp parallel num_threads(thread_count)   ←──── Compiler directive
    Hello();

    return 0;
}   /* main */

void Hello(void) {
    int my_rank = omp_get_thread_num();
    int thread_count = omp_get_num_threads();        ←──── Runtime routines

    printf("Hello from thread %d of %d\n", my_rank, thread_count);

}   /* Hello */
```

Compile and run

- Gcc 4.2 and above supports OpenMP 3.0
  - gcc -o omp_helloc -fopenmp omp_hello.c

- To run:
  - $./omp_hello 4

Debug OpenMP Using GDB

- Additional Usage

  - List all threads: info threads

  - thread-specific breakpoints:

    break *linespec* thread *threadnum*

  - Switch between threads:

    thread *threadnum*

  - apply a command to a list of threads

    thread apply <thread_number> <all> args

```
(gdb) l
1 #include <stdio.h>
2
3 int main()
4 {
5     int arr[5];
6     int i;
7     for (i=0;i<5;i++)
8         arr[i] = i;
9     for (i=0;i<=5;i++)

(gdb) break 9
Breakpoint 1 at 0x400544: file
buggy.c, line 9.

(gdb) run
Starting program:
/homes/cspeter/test/buggy


Breakpoint 1, main () at buggy.c:9
9     for (i=0;i<=5;i++)


(gdb) p arr
$1 = {0, 1, 2, 3, 4}
(gdb) p i
$2 = 5
```

Further notes

- When **any thread in your program stops,** for example, at a breakpoint, **all other threads in the program are also stopped by GDB.**

- GDB **cannot single-step all threads** in lockstep. Since thread scheduling is up to your debugging target's operating system (not controlled by GDB), **other threads may execute more than one statement while the current thread completes a single step** unless you use the command:

  set scheduler-locking on

- GDB is **NOT** able to show the values of private and shared variables in OpenMP parallel regions.

Demo 2 – Estimate π

$$\pi = 4\left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right] = 4\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

Serial solution

```
double factor = 1.0;
double sum = 0.0;
for (k = 0; k < n; k++) {
    sum += factor/(2*k+1);
    factor = -factor;
}
pi_approx = 4.0*sum;
```

```
1   #include <stdio.h>
2
3   |
4   int main(int argc, char* argv[]) {
5       double factor = 1.0;
6       double sum = 0.0;
7       int n = strtol(argv[1], NULL, 10);
8       double pi_approx;
9       int k;
10
11      for (k=0; k < n; k++){
12          sum += factor/(2*k+1);
13          factor = -factor;
14      }
15      pi_approx = 4.0*sum;
16      printf("Approximation value: %f\n", pi_approx);
17
18      return 0;
19  }
20
```

OpenMP Solution

```
         double sum = 0.0;
#        pragma omp parallel for num_threads(thread_count) \
            reduction(+:sum) private(factor)
         for (k = 0; k < n; k++) {
            if (k % 2 == 0)
                factor = 1.0;
            else
                factor = -1.0;
            sum += factor/(2*k+1);
         }
```

Insures factor has private scope.

OpenMP Solution

Sample codes

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]) {
    double factor = 1.0;
    double sum = 0.0;
    int n = 100000;

    double pi_approx;
    int k;

    int thread_count;
    thread_count = strtol(argv[1], NULL, 10);

#   pragma omp parallel for num_threads(thread_count) reduction(+:sum) private(factor)
    for (k=0; k < n; k++){
        if (k % 2 == 0)
            factor = 1.0;
        else
            factor = -1.0;
        sum += factor/(2*k+1);
    }
    pi_approx = 4.0*sum;
    printf("Approximation value: %f\n", pi_approx);

    return 0;
}
```

Debug OpenMP

- Step 1: compile with -g

    $gcc –fopenmp -o omp_estimate_pi -g omp_estimate_pi.c
- Step 2: gdb [program file]

    $gdb omp_estimate_pi

```
csl2wk07:yxingag:127> gcc -fopenmp -o omp_estimate_pi -g omp_estimate_pi.c
csl2wk07:yxingag:128> gdb omp_estimate_pi
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /homes/yxingag/hpc/lab01/omp_estimate_pi...done.
(gdb)
```

Debug OpenMP

- Step 3: display codes

    $list 13

- Step 4: set break points

    (e.g. at serial region)

    $(gdb) b 13

```
(gdb) list 13
8
9              double pi_approx;
10             int k;
11
12             int thread_count;
13             thread_count = strtol(argv[1],
NULL, 10);
14
15      #    pragma omp parallel for
num_threads(thread_count) reduction(+:sum)
private(factor)
16             for (k=0; k < n; k++){
17                  if (k % 2 == 0)

(gdb) break 13
Breakpoint 2 at 0x40075a: file
omp_estimate_pi_correct.c, line 13.
```

Debug OpenMP

- Step 5: run [args]

  $(gdb) run 4

  (run with 4 threads)

- Step 6: info threads

  $(gdb) info threads

- Step 7: next

  $(gdb) next

```
(gdb) run 4
Starting program:
/homes/yxingag/hpc/lab01/omp_estimate_pi_correct 4

Breakpoint 1, main (argc=2, argv=0x7fffffffe128) at
omp_estimate_pi_correct.c:13
13          thread_count = strtol(argv[1], NULL, 10);

(gdb) info threads
 Id    Target Id          Frame
* 1     Thread 0x7ffff7fcb7c0 (LWP 23922)
"omp_estimate_pi" main (argc=2, argv=0x7fffffffe128) at
omp_estimate_pi_correct.c:13

(gdb) next
15      #   pragma omp parallel for
num_threads(thread_count) reduction(+:sum)
private(factor)

(gdb) next
[New Thread 0x7ffff75ca700 (LWP 24561)]
[New Thread 0x7ffff6dc9700 (LWP 24562)]
[New Thread 0x7ffff65c8700 (LWP 24563)]
main._omp_fn.0 () at omp_estimate_pi_correct.c:15
15      #   pragma omp parallel for
num_threads(thread_count) reduction(+:sum)
private(factor)
```

Debug OpenMP

- Step 8: info threads

$(gdb) info threads

```
(gdb) info threads
 Id    Target Id              Frame
  4     Thread 0x7ffff65c8700 (LWP 24563) "omp_estimate_pi" 0x00007ffff7bcdc99 in ?? () from
/lib64/libgomp.so.1
  3     Thread 0x7ffff6dc9700 (LWP 24562) "omp_estimate_pi" 0x00007ffff7bcdc8b in ?? () from
/lib64/libgomp.so.1
  2     Thread 0x7ffff75ca700 (LWP 24561) "omp_estimate_pi" 0x00007ffff7bcdc99 in ?? () from
/lib64/libgomp.so.1
* 1     Thread 0x7ffff7fcb7c0 (LWP 23922) "omp_estimate_pi" main._omp_fn.0 () at
omp_estimate_pi_correct.c:15
```

Current thread: 1

Debug OpenMP

- Step 9: set break point to thread
    $(gdb) thread apply 3 b 21
- Step 10: switch current thread
    $(gdb) thread 3
- Step 11: continue to next break
point (may switching to other thread)
    $(gdb) c
- Step 12: display information
    $(gdb) print factor

```
(gdb) thread apply 3 b 21
Thread 3 (Thread 0x7ffff6dc9700 (LWP
25356)):
Breakpoint 2 at 0x40085c: file
omp_estimate_pi_correct.c, line 21.

(gdb) thread 4
[Switching to thread 3 (Thread
0x7ffff6dc9700 (LWP 25356))]
#0  0x00007ffff7bcdc88 in ?? () from
/lib64/libgomp.so.1

(gdb) c
Continuing.
[Switching to Thread 0x7ffff7fcb7c0 (LWP
25324)]

Breakpoint 2, main._omp_fn.0 () at
omp_estimate_pi_correct.c:21
21              sum += factor/(2*k+1);

(gdb)print factor
$1 = 1
```
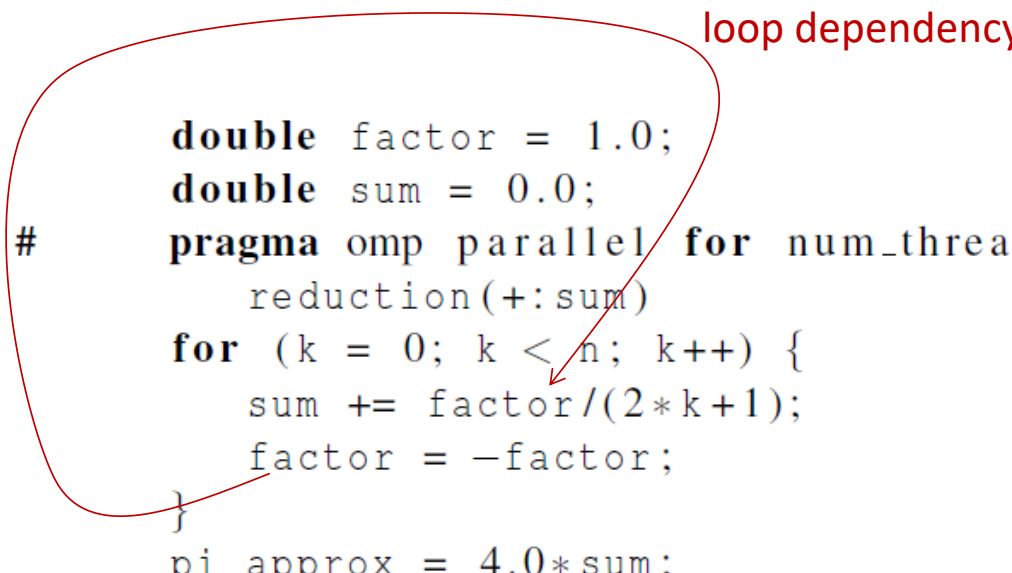
OpenMP Solution (incorrect)

loop dependency

```
        double factor = 1.0;
        double sum = 0.0;
#       pragma omp parallel for num_threads(thread_count) \
            reduction(+:sum)
        for (k = 0; k < n; k++) {
            sum += factor/(2*k+1);
            factor = -factor;
        }
        pi_approx = 4.0*sum;
```

Try to debug by yourself!

Environment Variables

▸ OpenMP provides several environment variables for controlling the execution of parallel code at run-time

  ▸ Example

csh/tcsh	**setenv OMP_NUM_THREADS 8**
sh/bash	**export OMP_NUM_THREADS=8**

▸ Setting the number of threads

▸ Specifying how loop iterations are divided

▸ Binding threads to processors

▸ Enabling/disabling nested parallelism; setting the maximum levels of nested parallelism

▸ Enabling/disabling dynamic threads

▸ Setting thread stack size

▸ Setting thread wait policy

Practice

- Implement Trapezoidal Rule with OpenMP!

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void Trap(double a, double b, int n, double* global_result_p);

int main(int argc, char* argv[]) {
    double  global_result = 0.0;  /* Store result in global_result */
    double  a, b;                 /* Left and right endpoints      */
    int     n;                    /* Total number of trapezoids    */
    int     thread_count;

    thread_count = strtol(argv[1], NULL, 10);
    printf("Enter a, b, and n\n");
    scanf("%lf %lf %d", &a, &b, &n);
#   pragma omp parallel num_threads(thread_count)
    Trap(a, b, n, &global_result);

    printf("With n = %d trapezoids, our estimate\n", n);
    printf("of the integral from %f to %f = %.14e\n",
        a, b, global_result);
    return 0;
}  /* main */
```

```c
void Trap(double a, double b, int n, double* global_result_p) {
    double  h, x, my_result;
    double  local_a, local_b;
    int  i, local_n;
    int my_rank = omp_get_thread_num();
    int thread_count = omp_get_num_threads();

    h = (b-a)/n;
    local_n = n/thread_count;
    local_a = a + my_rank*local_n*h;
    local_b = local_a + local_n*h;
    my_result = (f(local_a) + f(local_b))/2.0;
    for (i = 1; i <= local_n-1; i++) {
      x = local_a + i*h;
      my_result += f(x);
    }
    my_result = my_result*h;

#   pragma omp critical
    *global_result_p += my_result;
}  /* Trap */
```