OpenMP Examples (1A)

-
-

Young Won Lim
7/4/20

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Installation

STEP 1: Check the GCC version of the compiler
gcc –version

STEP 2: Configuring OpenMP
echo | cpp -fopenmp -dM |grep -i open
sudo apt install libomp-dev

STEP 3: Setting the number of threads
export OMP_NUM_THREADS=8

https://www.geeksforgeeks.org/openmp-introduction-with-installation-guide/

Parallel regions

```c
// OpenMP header
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    int nthreads, tid;

    // Begin of parallel region
    #pragma omp parallel private(nthreads, tid)
    {
        // Getting thread number
        tid = omp_get_thread_num();
        printf("Welcome to GFG from thread = %d\n", tid);
        if (tid == 0) {
            // Only master thread does this
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
    }
}
```

https://www.geeksforgeeks.org/openmp-introduction-with-installation-guide/

Private variables

```c
#include <omp.h>

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

    int nthreads, tid;

    /* Fork a team of threads with each thread having a private tid variable */
    #pragma omp parallel private(tid)
    {

        /* Obtain and print thread id */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);

        /* Only master thread does this */
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }

    }  /* All threads join master thread and terminate */

}
```

https://computing.llnl.gov/tutorials/openMP/#Compiling

OpenMP Code Structure

```
#include <omp.h>

main () {
    int var1, var2, var3;
    Serial code
    …

    Beginning of parallel region. Fork a team of threads.
    Specify variable scoping

    #pragma omp parallel private(var1, var2) shared(var3)
    {
        Parallel region executed by all threads
        Other OpenMP directives
        Run-time Library calls
        All threads join master thread and disband
    }

    Resume serial code
    ...
}
```

https://computing.llnl.gov/tutorials/openMP/

OpenMP Directives

#pragma omp parallel [**clause** ...] newline

       if (scalar_expression)

       private (list)

       shared (list)

       default (shared | none)

       firstprivate (list)

       reduction (operator: list)

       copyin (list)

       num_threads (integer-expression)

structured_block

https://computing.llnl.gov/tutorials/openMP/

OpenMP Directives

Directive name

A valid OpenMP directive.

Must appear after the pragma and before any clauses.

[clause, …]

Optional.

Clauses can be in any order, and repeated as necessary

unless otherwise restricted.

Newline

Required.

Precedes the structured block

which is enclosed by this directive.

https://computing.llnl.gov/tutorials/openMP/

Installation

Compile:

gcc -fopenmp test.c

Execute:

./a.out

https://www.geeksforgeeks.org/openmp-introduction-with-installation-guide/

Number of cores

grep processor /proc/cpuinfo | wc -l

sysconf(_SC_NPROCESSORS_CONF)
sysconf(_SC_NPROCESSORS_ONLN)

grep -c ^processor /proc/cpuinfo

grep -c ^cpu /proc/stat # subtract 1 from the result

https://stackoverflow.com/questions/150355/programmatically-find-the-number-of-cores-on-a-machine

OpenMP API Overview

The OpenMP 3.1 API is comprised of three distinct components:

- **Compiler Directives**
- **Runtime Library Routines**
- **Environment Variables**

https://computing.llnl.gov/tutorials/openMP/#API

Compiler Directives

- Spawning a <u>parallel</u> <u>region</u>
- Dividing <u>blocks</u> of code among threads
- Distributing <u>loop</u> <u>iterations</u> between threads
- <u>Serializing</u> sections of code
- <u>Synchronization</u> of work among threads

https://computing.llnl.gov/tutorials/openMP/#API

Runtime Library Routines

- Setting and querying the <u>number</u> of <u>threads</u>
- Querying a thread's unique identifier (<u>thread ID</u>),
  a thread's <u>ancestor's</u> <u>identifier</u>, the thread <u>team</u> <u>size</u>
- Setting and querying the <u>dynamic</u> <u>threads</u> feature
- Querying if in a <u>parallel</u> <u>region</u>, and at what <u>level</u>
- Setting and querying <u>nested</u> <u>parallelism</u>
- Setting, initializing and terminating <u>locks</u> and <u>nested</u> <u>locks</u>
- Querying <u>wall</u> <u>clock</u> <u>time</u> and <u>resolution</u>

https://computing.llnl.gov/tutorials/openMP/#API

Environment Variables

- Setting the <u>number</u> of <u>threads</u>
- Specifying how <u>loop</u> <u>iterations</u> are divided
- Binding <u>threads</u> to <u>processors</u>
- Enabling/disabling <u>nested</u> <u>parallelism</u>;
  setting the <u>maximum</u> <u>levels</u> of nested parallelism
- Enabling/disabling <u>dynamic</u> <u>threads</u>
- Setting <u>thread</u> <u>stack</u> <u>size</u>
- Setting <u>thread</u> <u>wait</u> <u>policy</u>

https://computing.llnl.gov/tutorials/openMP/#API

Examples

Compiler Directive Examples

#pragma omp **parallel**
#pragma omp **parallel private**(partial_Sum) **shared**(total_Sum)
#pragma omp **parallel private**(thread_id)
#pragma omp **barrier**
#pragma omp **for**
#pragma omp **critical**

Runtime Library Routine Examples

omp_get_thread_num();
omp_get_max_threads();

https://stackoverflow.com/questions/150355/programmatically-find-the-number-of-cores-on-a-machine

Hello

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

int main(int argc, char** argv) {

    printf("Hello from process: %d\n", omp_get_thread_num());

    return 0;
}
```

```
// only one thread giving us a Hello statement
// must use the #pragma omp parallel { … } directive
// for multiple threads
```

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

Hello

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

int main(int argc, char** argv){
    int thread_id;

    #pragma omp parallel
    {
        printf("Hello from process: %d\n", omp_get_thread_num());
    }
    return 0;
}
```

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

Private clauses

The PRIVATE clause declares variables in its list
to be private to each thread.

- A new object of the same type is declared once
  for each thread in the team
- All references to the original object are replaced with
  references to the new object
- Should be assumed to be uninitialized for each thread

Shared clauses

The SHARED clause declares variables in its list t
o be shared among all threads in the team.

A shared variable exists in only one memory location and
all threads can read or write to that address

It is the programmer's responsibility to ensure that
multiple threads properly access SHARED variables
(such as via CRITICAL sections)

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

Shared clauses

Variables that are created and assigned
inside of a parallel section of code will be
inherently be **private**

variables created outside of parallel sections
will be inherently **public**.

Data Sharing Rules – Implicit Rules

```
int n = 10;              // shared
int a = 7;               // shared

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

http://jakascorner.com/blog/2016/06/omp-data-sharing-attributes.html

Data Sharing Rules – Explicit Rules

```
#pragma omp parallel for shared(n, a)
for (int i = 0; i < n; i++)
{
    int b = a + i;

    ...

}



#pragma omp parallel for shared(n, a) private(b)
for (int i = 0; i < n; i++)
{
    b = a + i;

    ...

}
```

http://jakascorner.com/blog/2016/06/omp-data-sharing-attributes.html

Data Sharing Rules – Explicit Rules

```
int p = 0;
// the value of p is 0

#pragma omp parallel private(p)
{
    // the value of p is undefined
    p = omp_get_thread_num();
    // the value of p is defined

    ...
}
// the value of p is undefined



#pragma omp parallel
{
    int p = omp_get_thread_num();

    ...
}
```

http://jakascorner.com/blog/2016/06/omp-data-sharing-attributes.html

Data Sharing Rules – Default(Shared)

```
int a, b, c, n;
...

#pragma omp parallel for default(shared)
for (int i = 0; i < n; i++)
{
    // using a, b, c
}
```

http://jakascorner.com/blog/2016/06/omp-data-sharing-attributes.html

Data Sharing Rules – Default(Shared)

```
int n = 10;
std::vector<int> vector(n);
int a = 10;

#pragma omp parallel for default(none) shared(n, vector)
for (int i = 0; i < n; i++)
{
    vector[i] = i * a;
}


error: 'a' not specified in enclosing parallel
        vector[i] = i * a;
                      ^
error: enclosing parallel
    #pragma omp parallel for default(none) shared(n, vector)
        ^
```

http://jakascorner.com/blog/2016/06/omp-data-sharing-attributes.html

Hello

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

int main(int argc, char** argv){
    int thread_id;

    #pragma omp parallel private(thread_id)
    {
        thread_id = omp_get_thread_num();
        printf("Hello from process: %d\n", thread_id );
    }

    return 0;
}
```

// create a separate instance of thread_id for each task.

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

Barrier and critical directives

#pragma omp **barrier**

The barrier directive <u>stops</u> all processes
for proceeding to the next line of code
<u>until</u> <u>all processes</u> have reached the barrier.
This allows a programmer
to **synchronize** sequences in the parallel process.

#pragma omp **critical** { … }

A critical directive ensures that
<u>a line of code</u> is only run <u>by one process</u> at a time,
ensuring **thread safety** in the body of code.

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

Barrier (1)

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

int main(int argc, char** argv){
    //define loop iterator variable outside parallel region
    int i;
    int thread_id;

    #pragma omp parallel
    {
        thread_id = omp_get_thread_num();

        //create the loop to have each thread print hello.
        for(i = 0; i < omp_get_max_threads(); i++){
            printf("Hello from process: %d\n", thread_id);
        }
    }
    return 0;
}
```

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

Barrier (2)

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

int main(int argc, char** argv){
    int i;
    int thread_id;

    #pragma omp parallel
    {
        thread_id = omp_get_thread_num();

        for(i = 0; i < omp_get_max_threads(); i++){
            if(i == thread_ID){
                printf("Hello from process: %d\n", thread_id);
            }
        }
    }
    return 0;
}
```

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

Barrier (3)

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

int main(int argc, char** argv){
    int i;
    int thread_id;

    #pragma omp parallel
    {
        thread_id = omp_get_thread_num();

        for( int i = 0; i < omp_get_max_threads(); i++){
            if(i == omp_get_thread_num()){
                printf("Hello from process: %d\n", thread_id);
            }
            #pragma omp barrier
        }
    }
    return 0;
}
```

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

OMP for

OpenMP's power comes from
easily splitting a larger task into multiple smaller tasks.
Work-sharing directives allow for simple and effective **splitting**
of normally serial tasks into fast parallel sections of code.

The directive omp for divides a normally serial for loop into a parallel task.

#pragma omp for { … }

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

OMP for

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

int main(int argc, char** argv){
    int partial_Sum, total_Sum;


    #pragma omp parallel private(partial_Sum) shared(total_Sum)
    {
        partial_Sum = 0;
        total_Sum = 0;


        #pragma omp for
        {
            for(int i = 1; i <= 1000; i++){
                partial_Sum += i;
            }
        }

        //Create thread safe region.
        #pragma omp critical
        {
            //add each threads partial sum to the total sum
            total_Sum += partial_Sum;
        }
    }

    printf("Total Sum: %d\n", total_Sum);
    return 0;
}
```

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#parallel-hello-world-program

References

[1]   en.wikipedia.org

[2]   M Harris, http://beowulf.lcs.mit.edu/18.337-2008/lectslides/scan.pdf