OpenMP Tutorial  
  
<https://computing.llnl.gov/tutorials/openMP/>

**C / C++ - General Code Structure**

#include <omp.h>

main () {

int var1, var2, var3;

*Serial code*

.

.

.

*Beginning of parallel section. Fork a team of threads.*

*Specify variable scoping*

#pragma omp parallel private(var1, var2) shared(var3)

{

*Parallel section executed by all threads*

.

*Other OpenMP directives*

.

*Run-time Library calls*

.

*All threads join master thread and disband*

}

*Resume serial code*

.

.

.

}

|  |  |  |
| --- | --- | --- |
| GNU C/C++ | 4.4.7 | OpenMP 3.0 |
| OpenMP 4.0 Support: according to vendor documentation, beginning with the following compiler versions, OpenMP 4.0 is supported:   * GNU: 4.9 for C/C++ |  |  |

|  |  |  |
| --- | --- | --- |
| **Compiler / Platform** | **Compiler** | **Flag** |

GNU gcc -fopenmp

**C / C++ Directives Format**

**Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| **#pragma omp** | **directive-name** | **[clause, ...]** | **newline** |
| Required for all OpenMP C/C++ directives. | A valid OpenMP directive. Must appear after the pragma and before any clauses. | Optional. Clauses can be in any order, and repeated as necessary unless otherwise restricted. | Required. Precedes the structured block which is enclosed by this directive. |

**Example:**

|  |
| --- |
| **#pragma omp parallel default(shared) private(beta,pi)** |

|  |  |
| --- | --- |
| **C/C++** | #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* |

**Notes:**

* When a thread reaches a PARALLEL directive, it creates a team of threads and becomes the master of the team. The master is a member of that team and has thread number 0 within that team.
* Starting from the beginning of this parallel region, the code is duplicated and all threads will execute that code.
* There is an implied barrier at the end of a parallel section. Only the master thread continues execution past this point.
* If any thread terminates within a parallel region, all threads in the team will terminate, and the work done up until that point is undefined.

**How Many Threads?**

* The number of threads in a parallel region is determined by the following factors, in order of precedence:
  1. Evaluation of the **IF** clause
  2. Setting of the **NUM\_THREADS** clause
  3. Use of the **omp\_set\_num\_threads()** library function
  4. Setting of the **OMP\_NUM\_THREADS** environment variable
  5. Implementation default - usually the number of CPUs on a node, though it could be dynamic (see next bullet).
* Threads are numbered from 0 (master thread) to N-1

**C / C++ - Parallel Region Example**

#include <omp.h>

main () {

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 \*/

}

|  |
| --- |
| **OpenMP Exercise 1** |

**Getting Started**

|  |
| --- |
| **Overview:**   * **Login to the workshop cluster using your workshop username and OTP token** * **Copy the exercise files to your home directory** * **Familiarize yourself with LC's OpenMP environment** * **Write a simple "Hello World" OpenMP program** * **Successfully compile your program** * **Successfully run your program** * **Modify the number of threads used to run your program**   **https://computing.llnl.gov/tutorials/images/point02.jpg**[**GO TO THE EXERCISE HERE**](https://computing.llnl.gov/tutorials/openMP/exercise.html)  https://computing.llnl.gov/tutorials/openMP/exercise.html |

|  |  |
| --- | --- |
| |  | | --- | | OpenMP Exercise | |

## <https://computing.llnl.gov/tutorials/openMP/exercise.html> Exercise 1

1. **Login to the workshop machine**

Workshops differ in how this is done. The instructor will go over this beforehand.

1. **Copy the example files**
   1. In your home directory, create a subdirectory for the example codes and then cd to it.
   2. **mkdir openMP**

**cd openMP**

* 1. Then, copy the C version of the parallel OpenMP exercise files to your openMP subdirectory:

|  |  |
| --- | --- |
| **C:** | **cp /usr/global/docs/training/blaise/openMP/C/\* ~/openMP** |

<https://computing.llnl.gov/tutorials/openMP/exercise.html>

EXAMPLE 1 - hello world

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **FILE: omp\_hello.c** Hello world

\* DESCRIPTION:

\* OpenMP Example - Hello World - C/C++ Version

\* In this simple example, the master thread forks a parallel region.

\* All threads in the team obtain their unique thread number and print it.

\* The master thread only prints the total number of threads. Two OpenMP

\* library routines are used to obtain the number of threads and each

\* thread's number.

\* AUTHOR: Blaise Barney 5/99

\* LAST REVISED: 04/06/05

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

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

{

int nthreads, tid;

/\* Fork a team of threads giving them their own copies of variables \*/

#pragma omp parallel private(nthreads, tid)

{

/\* Obtain thread number \*/

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 disband \*/

}

Using your choice of compiler (see above section 4), compile your hello world OpenMP program. This may take several attempts if there are any code errors. For example:

**gcc -fopenmp omp\_hello.c -o hello**

1. When you get a clean compile, proceed.
2. Run your **hello** executable and notice its output.
   * Is it what you expected? As a comparison, you can compile and run the provided **omp\_hello.c**  example program.
3. How many threads were created? By default, the GNU compilers will create 1 thread for each core.
4. Notes:
   * For the remainder of this exercise, you can use the compiler command of your choice unless indicated otherwise.
   * Compilers will differ in which warnings they issue, but all can be ignored for this exercise. Errors are different, of course.

EXAMPLE 2 – workShare1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE: omp\_workshare1.c Loop work-sharing

\* DESCRIPTION:

\* OpenMP Example - Loop Work-sharing - C/C++ Version

\* In this example, the iterations of a loop are scheduled dynamically

\* across the team of threads. A thread will perform CHUNK iterations

\* at a time before being scheduled for the next CHUNK of work.

\* AUTHOR: Blaise Barney 5/99

\* LAST REVISED: 04/06/05

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define CHUNKSIZE 10

#define N 100

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

{

int nthreads, tid, i, chunk;

float a[N], b[N], c[N];

/\* Some initializations \*/

for (i=0; i < N; i++)

a[i] = b[i] = i \* 1.0;

chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid)

{

tid = omp\_get\_thread\_num();

if (tid == 0)

{

nthreads = omp\_get\_num\_threads();

printf("Number of threads = %d\n", nthreads);

}

printf("Thread %d starting...\n",tid);

#pragma omp for schedule(dynamic,chunk)

for (i=0; i<N; i++)

{

c[i] = a[i] + b[i];

printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);

}

} /\* end of parallel section \*/

}

EXAMPLE 3 - workShare2

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE: omp\_workshare2.c

\* DESCRIPTION:

\* OpenMP Example - Sections Work-sharing - C Version

\* In this example, the OpenMP SECTION directive is used to assign

\* different array operations to each thread that executes a SECTION.

\* AUTHOR: Blaise Barney 5/99

\* LAST REVISED: 07/16/07

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define N 50

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

{

int i, nthreads, tid;

float a[N], b[N], c[N], d[N];

/\* Some initializations \*/

for (i=0; i<N; i++) {

a[i] = i \* 1.5;

b[i] = i + 22.35;

c[i] = d[i] = 0.0;

}

#pragma omp parallel shared(a,b,c,d,nthreads) private(i,tid)

{

tid = omp\_get\_thread\_num();

if (tid == 0)

{

nthreads = omp\_get\_num\_threads();

printf("Number of threads = %d\n", nthreads);

}

printf("Thread %d starting...\n",tid);

#pragma omp sections nowait

{

#pragma omp section

{

printf("Thread %d doing section 1\n",tid);

for (i=0; i<N; i++)

{

c[i] = a[i] + b[i];

printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);

}

}

#pragma omp section

{

printf("Thread %d doing section 2\n",tid);

for (i=0; i<N; i++)

{

d[i] = a[i] \* b[i];

printf("Thread %d: d[%d]= %f\n",tid,i,d[i]);

}

}

} /\* end of sections \*/

printf("Thread %d done.\n",tid);

} /\* end of parallel section \*/

}

EXAMPLE

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE: omp\_mm.c Matrix multiply

\* DESCRIPTION:

\* OpenMp Example - Matrix Multiply - C Version

\* Demonstrates a matrix multiply using OpenMP. Threads share row iterations

\* according to a predefined chunk size.

\* AUTHOR: Blaise Barney

\* LAST REVISED: 06/28/05

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define NRA 62 /\* number of rows in matrix A \*/

#define NCA 15 /\* number of columns in matrix A \*/

#define NCB 7 /\* number of columns in matrix B \*/

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

{

int tid, nthreads, i, j, k, chunk;

double a[NRA][NCA], /\* matrix A to be multiplied \*/

b[NCA][NCB], /\* matrix B to be multiplied \*/

c[NRA][NCB]; /\* result matrix C \*/

chunk = 10; /\* set loop iteration chunk size \*/

/\*\*\* Spawn a parallel region explicitly scoping all variables \*\*\*/

#pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)

{

tid = omp\_get\_thread\_num();

if (tid == 0)

{

nthreads = omp\_get\_num\_threads();

printf("Starting matrix multiple example with %d threads\n",nthreads);

printf("Initializing matrices...\n");

}

/\*\*\* Initialize matrices \*\*\*/

#pragma omp for schedule (static, chunk)

for (i=0; i<NRA; i++)

for (j=0; j<NCA; j++)

a[i][j]= i+j;

#pragma omp for schedule (static, chunk)

for (i=0; i<NCA; i++)

for (j=0; j<NCB; j++)

b[i][j]= i\*j;

#pragma omp for schedule (static, chunk)

for (i=0; i<NRA; i++)

for (j=0; j<NCB; j++)

c[i][j]= 0;

/\*\*\* Do matrix multiply sharing iterations on outer loop \*\*\*/

/\*\*\* Display who does which iterations for demonstration purposes \*\*\*/

printf("Thread %d starting matrix multiply...\n",tid);

#pragma omp for schedule (static, chunk)

for (i=0; i<NRA; i++)

{

printf("Thread=%d did row=%d\n",tid,i);

for(j=0; j<NCB; j++)

for (k=0; k<NCA; k++)

c[i][j] += a[i][k] \* b[k][j];

}

} /\*\*\* End of parallel region \*\*\*/

/\*\*\* Print results \*\*\*/

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Result Matrix:\n");

for (i=0; i<NRA; i++)

{

for (j=0; j<NCB; j++)

printf("%6.2f ", c[i][j]);

printf("\n");

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

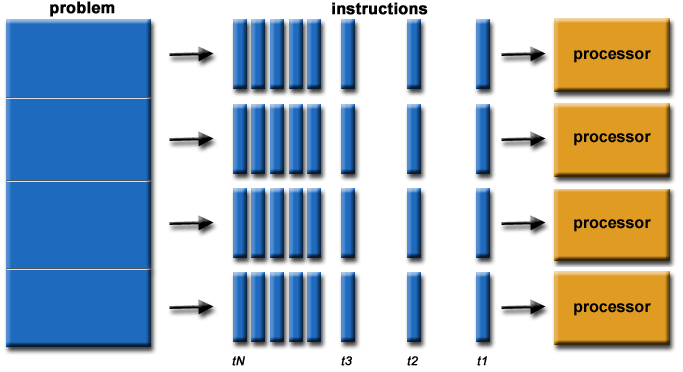
printf ("Done.\n");

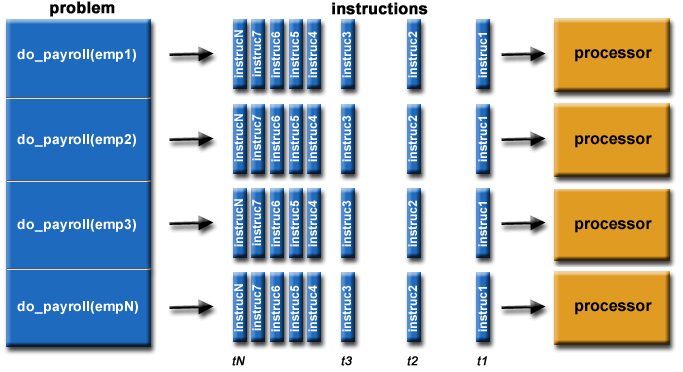
}

<https://computing.llnl.gov/tutorials/parallel_comp/>

**Parallel Computing:**

* In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem:
  + A problem is broken into discrete parts that can be solved concurrently
  + Each part is further broken down to a series of instructions
  + Instructions from each part execute simultaneously on different processors
  + An overall control/coordination mechanism is employed





* The computational problem should be able to:
  + Be broken apart into discrete pieces of work that can be solved simultaneously;
  + Execute multiple program instructions at any moment in time;
  + Be solved in less time with multiple compute resources than with a single compute resource.
* The compute resources are typically:
  + A single computer with multiple processors/cores
  + An arbitrary number of such computers connected by a network.

# OPENMP  C Examples of Parallel Programming with OpenMP

<https://people.sc.fsu.edu/~jburkardt/c_src/openmp/openmp.html>

# OpenMP Exercise

<https://computing.llnl.gov/tutorials/openMP/exercise.html>

# Aprendendo a usar a estrutura OpenMP com GCC

<http://www.ibm.com/developerworks/br/aix/library/au-aix-openmp-framework/#list2>