#include <omp.h>

#include <stdio.h>

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

{

#pragma omp parallel

printf(“Hello World”);

}

**Compile: gcc –fopenmp hello.c**

#include <omp.h>

#include <stdio.h>

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

{

#pragma omp parallel num\_threads(8)

printf(“Hello World”);

}

**Compile: gcc –fopenmp hello.c**

#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 \*/

}

**Compile: gcc –fopenmp hello.c**

Sum of array:

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

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

{

int i, n;

float a[100], b[100], sum;

/\* Some initializations \*/

n = 100;

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

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

sum = 0.0;

#pragma omp parallel for reduction(+:sum)

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

sum = sum + (a[i] \* b[i]);

printf(" Sum = %f\n",sum);

}

**Compile: gcc –fopenmp sum.c**

#include<omp.h>

#include<stdio.h>

#include<stdlib.h>

int main(){

int r = 2000, c = 2000, i, j, count=0, sum =0, k;

//dynamically allocate arrays

int \*\*arr1 = (int \*\*)malloc(r \* sizeof(int \*));

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

arr1[i] = (int \*)malloc(c \* sizeof(int));

int \*\*arr2 = (int \*\*)malloc(r \* sizeof(int \*));

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

arr2[i] = (int \*)malloc(c \* sizeof(int));

int \*\*arr3 = (int \*\*)malloc(r \* sizeof(int \*));

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

arr3[i] = (int \*)malloc(c \* sizeof(int));

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

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

arr1[i][j] = count++;

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

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

arr2[i][j] = count++;

double x = omp\_get\_wtime();

omp\_set\_num\_threads(2);

//#pragma omp parallel for private(j, k)

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

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

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

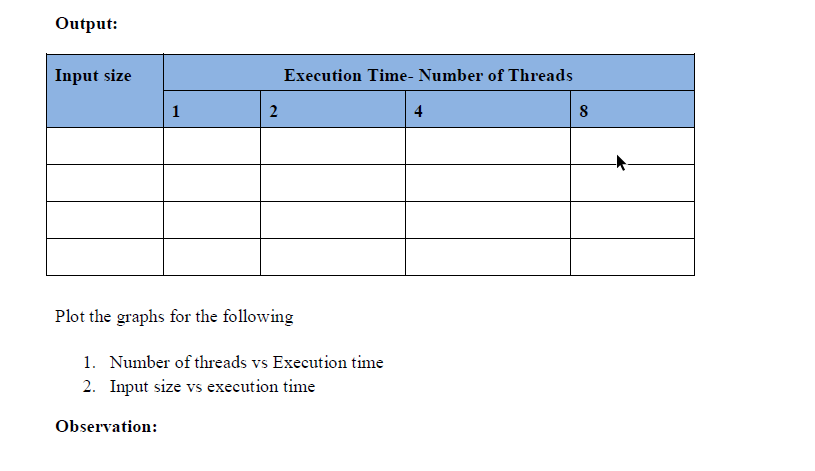
arr3[i][j] += arr1[i][k] \* arr2[k][j];

double y = omp\_get\_wtime();

printf("%lf\n", y-x);

return 0;

}



**Hello world**

Here is the basic Hello world program in C using MPI:

#include <stdio.h>

#include <mpi.h>

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

{

int ierr;

ierr = MPI\_Init(&argc, &argv);

printf("Hello world\n");

ierr = MPI\_Finalize();

}

If you compile hello.c with a command like

mpicc hello.c -o hello

you will create an executable file called hello, which you can execute by using the mpirun command as in the following session segment:

$ **mpirun -np 4 hello**

Hello world

Hello world

Hello world

Hello world

$

When the program starts, it consists of only one process, sometimes called the "parent", "root", or "master" process. When the routine MPI\_Init executes within the root process, it causes the creation of 3 additional processes (to reach the number of processes (np) specified on the mpirun command line), sometimes called "child" processes.

Each of the processes then continues executing separate versions of the hello world program. The next statement in every program is the printf statement, and each process prints "Hello world" as directed. Since terminal output from every program will be directed to the same terminal, we see four lines saying "Hello world".

**dentifying the separate processes**

As written, we cannot tell which "Hello world" line was printed by which process. To identify a process we need some sort of process ID and a routine that lets a process find its own process ID. MPI assigns an integer to each process beginning with 0 for the parent process and incrementing each time a new process is created. A process ID is also called its "rank".

MPI also provides routines that let the process determine its process ID, as well as the number of processes that are have been created.

Here is an enhanced version of the Hello world program that identifies the process that writes each line of output:

#include <stdio.h>

#include <mpi.h>

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

{

int ierr, num\_procs, my\_id;

ierr = MPI\_Init(&argc, &argv);

/\* find out MY process ID, and how many processes were started. \*/

ierr = MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_id);

ierr = MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_procs);

printf("Hello world! I'm process %i out of %i processes\n",

my\_id, num\_procs);

ierr = MPI\_Finalize();

}

When we run this program, each process identifies itself:

$ **mpicc hello2.c -o hello2**

$ **mpirun -np 4 hello2**

Hello world! I'm process 0 out of 4 processes.

Hello world! I'm process 2 out of 4 processes.

Hello world! I'm process 1 out of 4 processes.

Hello world! I'm process 3 out of 4 processes.

$