1 Basic programs

The first example of an OPENMP program is

```c
#include <stdio.h>
#include<omp.h>
int main(void)
{
#pragma omp parallel
{
  printf("Hello, world.\n");
}
printf("Ciao mondo.\n");
return 0;
}
```

```cpp
#include <iostream>
#include<omp.h>
using namespace std;
int main()
{
#pragma omp parallel
{
cout<<"Hello, world!"<<endl;
}
cout<<"Ciao, mondo!"<<endl;
}
```

1. Compile the (left) code with gcc with the option -fopenmp and run it. Compile the (right) code with g++ with the same option and run it.
2. How many Hello World do you have ?
   For Unix users, you can set the number of threads by setting the variable OMP_NUM_THREADS to 2, 8, 20.(In a terminal you type export OMP_NUM_THREADS =4, for instannce.)
   Do you observe some changes ?
3. Compile without the option -fopenmp and run again. What happens ?
   The second code illustrates the loop parallelization

```cpp
#include <iostream>
#include <cmath>
#include<vector>
#include<omp.h>
using namespace std;
double essai(double x)
{ return(5.0+10.0*x+x*x*exp(x)*log(x+0.1)+sqrt(fabs(x)));}

int main(){
const int NITER=200000000;
vector<double> a(NITER);
#pragma omp parallel for default(shared)
for (int j=0;j<NITER;j++){
a[j] = essai(j*0.01);}
exit(0);
}
```

4. Compile (with g++) and run the code.
5. In order to measure the efficiency of the parallelization, type

time progexec
where progexec is the executable.
By setting the variable OMP_NUM_THREADS to 1, 2, 4, 6 8, 16 collect the different results (user time, real time) in a file and plot (by using matplotlib) the two evolutions. Comment your graphics.

2 Internal functions

The following code is able to collect internal information

```
#include<iostream>
#include<ctime>
#include<vector>
#include<omp.h>
using namespace std;
const int NITER=40000;
vector <vector<double> > a(NITER,vector<double> (NITER));
double essai(double x,double y){
return(x*y);
}
int main(){
 clock_t debut=clock();
 double deb,end;
#pragma omp parallelTE3/matrice5.cpp
 if(omp_get_thread_num() == 0) { deb=omp_get_wtime();}
#pragma omp parallel for default(shared)
 for (int i=0;i<NITER;i++){
 for (int j=0;j<NITER;j++){
 a[i][j] = essai((double) i,(double) j);}
}
 if(omp_get_thread_num() == 0) {end=omp_get_wtime();
 cout<<"omp elasped time "<<end-deb<<endl;}

clock_t fin=clock();
cout<<"global elasped time "<<(double) (fin-debut)/CLOCKS_PER_SEC<<endl;
}
```

1. Compile and run the code.
2. Suppress the line #pragma omp parallel. Recompile and rerun. What happens ?

3 Reduction

The following code computes the π number by using a numerical evaluation of an integral by a rectangle method. Each thread computes a part of the loop and a reduction instruction is performed

```
#include <iostream>
#include <cmath>
#include<omp.h>
using namespace std;
double f( double a ) { return (4.0 / (1.0 + a*a)); }
int main()
{
const int n= 1000000000;
double startwtime, endwtime,pi, h, sum=0.0;
double pi_ex=acos(-1);
h = 1.0 / (double) n;
#pragma omp parallel
if (omp_get_thread_num() == 0) {startwtime = omp_get_wtime();}
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i <= n; i ++)
{
double x = h * (i - 0.5);
sum += f(x);
}
pi = h * sum;
if (omp_get_thread_num() == 0)
{ cout.precision(12);
cout<<"pi is approximately " <<pi<<" Error is "<< fabs(pi - pi_ex)<<endl;
endwtime = omp_get_wtime();
cout<<"wall clock time = "<<endwtime-startwtime<<endl;
}
}
```

1. Compile and run the code.
2. Increase the thread number from 1 to 8 Collect data and plot the wall time versus the number of threads
3. Open a second terminal and type top. Rerun the program for a thread number of 1, 2 and 4

4 Synchronization clauses

Because the different threads share the same memory, it is necessary to check that two (or more) threads do not attempt to write in the same location at the same time. This is important to avoid conflict and consequently to avoid unexpected bugs in your code. In the pragma directive of a loop, you can specify the nature of variables used in your code. A simple rule consists in declaring interval variables within the loop.

In order to show this characteristic of parallel coding, change the previous code as follows :

- Remove the declaration for x within the loop
- add the line double x : after the line double pi_ex,...

1. Compile and run for 4 threads. What happens ?
2. Add the clause private (x) at the end of the pragma omp parallel for Compile and run again.
3. Remove the clause private(x) and compile with the option -O3. Compile and run again. What happens ?

A Windows

For Windows users, you need to first install Mingw. Second, you install Codeblocks

In order to use the openmp library, you need to set in compiler option **-fopenmp** as well as in the linker options.

B Linux

Use the package manager of your distribution for installing codeblocks. If you have a recent distribution, gcc is installed with openmp.

C MacOx

Codeblocks is longer supported for MacOx. You can use Xcode and gcc is already installed with your Os system. When you are using the compiler clang, In the terminal, the command is the following

clang(++) -Xpreprocessor -fopenmp filename.c(pp) -lomp -o nameforexec

Note : libraries libomp and llvm must both be installed and up to date. (thanks to Matteo Butano for this information)