Lab Guide 5

Code Parallelisation with OpenMP

Objective:

- manage accesses to data in shared memory environments with OpenMP
- analyse the performance scalability

Introduction

The following exercises aim to introduce the data related concepts of code parallelisation on shared memory environments with OpenMP. Students should run the application multiple times, looking at the output of each run for clues to understand how its behaviour is affected by various OpenMP directives.
The amount of threads to be used by OpenMP can be set through the bash environment, instead of hardcoded in the application, by executing the command:

```
user > export OMP_NUM_THREADS=N
```

The compilation and execution can be performed on a node of the SeARCH cluster (login into the server with `s7edu.di.uminho.pt`). The program should be compiled with the `–fopenmp` flag:

```
gcc -std=c99 -O2 -fopenmp -lm sample.c
```

Execution can be performed on a compute node of the cluster, in the partition "`cpar`".

```
srun --partition=cpar ./a.out
```

Exercise 1 - Data sharing with OpenMP

Copy & paste the following code to a new file, compile and run it 2/3 times. It contains a small loop that displays the current iteration number and the identification of the corresponding thread executing it, as well as a counter `w`, shared among all threads.

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

int main() {
    int w=10;
#pragma omp parallel
#pragma omp for
    for(int i=0;i<100;i++) {
        int id = omp_get_thread_num();
        printf("T%d:i%d w=%d\n", id, i, w++);
    }
    printf("w=%d\n", w);
}
```

Test the code as is (version 1.0) and then add & test the following clauses to the `#pragma omp for` directive:

1.1. `private(w)`

1.2. `firstprivate(w)`

1.3. `lastprivate(w)`

1.4. `reduction(+:w)`

a) How does the initial value of `w` vary with the use of the 1.0-1.4 directives?

b) Does the final value of `w` inside the loop vary with the use of the 1.0-1.4 directives?

c) After the loop execution is the final value of `w` what would be expected?

Exercise 2 - Data races in OpenMP

Consider the following code that computes the dot product of two vectors:

```
#include<omp.h>
#include<stdio.h>
#define size 100000
double a[size], b[size];
int main() {
     // init vectors
     for(int i=0;i<size; i++) {
             a[i] = 1.0/((double) (size-i));
             b[i] = a[i] * a[i];
     }
     // compute dot product
     double dot = 0;
     for(int i=0;i<size; i++) {
         dot += a[i]*b[i];
     }
     printf("Dot is %18.16f\n",dot);
}
```

Parallelise the execution of the dot product (2nd `for-loop`) using the `#pragma omp parallel for` directive.

a) Does the result of the dot product differ from run to run? If so, why?

b) Is the result of the dot product affected by using different amounts of threads?
Suggestion: run the code with 2, 4, and 8 threads.

c) Adapt the parallelisation to produce the correct results using any of the OpenMP directives <u>studied in the previous lab session</u> (lab 4).
Validate the correctness of the implementation by testing different amounts of threads.

d) Is it possible to achieve a better functionally correct parallelisation using different directive(s)?

Exercise 3 – Parallelisation scalability

Consider the following code that computes an approximation of the value of *pi*:

```
double f( double a ) {
    return (4.0 / (1.0 + a*a));
}
double pi = 3.141592653589793238462643;
int main() {
    double mypi = 0;
    int n = 1000000000; // number of points to compute
    float h = 1.0 / n;
    for(int i=0; i<n; i++) {
        mypi = mypi + f(i*h);
    }
    mypi = mypi * h;
    printf(" pi = %.10f \n", mypi);
}
```

Parallelise the code with OpenMP, using adequate directives to tackle potential data races. Measure the performance of the code for 2, 4, 8, and 16 threads, and calculate the speedup relative to its sequential execution. Does the performance improve as expected?

Note: compile the code without `-fopenmp` *to generate a sequential application.*

Suggestion: Consider the real clock time of the `time` *command as the application execution time*

`time srun -partition=cpar./a.out`