Université Grenoble Alpes, Grenoble INP, UFR IM^{2}AG

Master 1 Informatique and Master 1 MOSIG

UE Parallel Algorithms and Programming

Lab 1 – 2021

This lab is an introduction to programming with OpenMP.

Main information about this lab:

- This lab is to be done by groups of at most 2.
- This lab is not graded. You don't have to submit anything at the end of this lab.

1 Compiling and executing OpenMP programs

Before jumping into the exercises, a few comments about the requirements to be able to compile and execute OpenMP programs.

On your laptop

OpenMP is supported by default by the the main C compilers (including `gcc` and `clang`). Hence, if you run on a machine with Linux, you should be able to compile and execute OpenMP programs without any problems. It might also work on MacOS.

Connecting to the UFR server using ssh

An alternative is to run this lab on the server of the university called `Mandelbrot`.

To connect to this server, simply run:

```
ssh [LOGIN]@im2ag-mandelbrot.univ-grenoble-alpes.fr
```

This server also has the advantage to give you access to a machine with more cores. However, this is a shared platform. The activity of others can impact your performance evaluations.

Running experiments on Google Cloud

You will receive in the next few days instructions on how you can obtain Google Cloud credits, and how you can use this platform to run more experiments.

2 Compiling and executing OpenMP programs

The source files for this lab are in the `lab1.tar.gz` archive stored on the moodle (PAPC course).

```
$ cd
$ tar xvfz lab1.tar.gz
$ cd lab1
```

You can compile OpenMP programs with the GNU (`gcc`) compiler (`clang`). Start by compiling the file `ex1.c`.

```
$ gcc -O0 -fopenmp -o ex1 ex1.c
```

The `ex1` program takes one argument.

```
$ ./ex1 8
I am the master thread 0 and I start
Starting Region 1
Region 1  thread 0 of team 8 (max_num_threads is 8)
Region 1  thread 7 of team 8 (max_num_threads is 8)
Region 1  thread 4 of team 8 (max_num_threads is 8)
Region 1  thread 2 of team 8 (max_num_threads is 8)
Region 1  thread 1 of team 8 (max_num_threads is 8)
Region 1  thread 6 of team 8 (max_num_threads is 8)
Region 1  thread 5 of team 8 (max_num_threads is 8)
Region 1  thread 3 of team 8 (max_num_threads is 8)
End of Region 1
Starting Region 2
Region 2  thread 3 of team 4 (max_num_threads is 8)
Region 2  thread 0 of team 4 (max_num_threads is 8)
Region 2  thread 2 of team 4 (max_num_threads is 8)
Region 2  thread 1 of team 4 (max_num_threads is 8)
End of Region 2
Region 3  thread 0 of team 2 (max_num_threads is 8)
Region 3  thread 1 of team 2 (max_num_threads is 8)
End of Region 3
I am the master thread 0 and I complete
$
```

1. Explain the output of this OpenMP program.
2. Compare and analyze the output of different runs of this program.

3 Performance Analysis of Vector and Matrix Operations

In this exercise, we will study how basic numerical functions can be simply parallelized with OpenMP. The functions to be studied are defined in file `ex2.c`.

A `vector` and `matrix` types are defined in this file. The size of the manipulated vectors and matrices are defined on by the single argument taken by the program.

The `init_vector()` and `init_matrix()` functions allocate and initialize vectors and matrices. The computing functions we are going to study are:

- addition of two vectors (`add_vectors`), scalar product (`dot`) of two vectors.

- multiplication between a matrix and a vector (`mult_mat_vector`). The implementations are not provided.
- multiplication between two matrices (`mult_mat_mat`). The implementations are not provided.

The main function of this `ex2` program calls the different functions. For each call, the number of processor cycles is measured using the intrinsic _rdtsc (). Several runs are performed to compute an average.

You can get the nominal frequency of your processor in the `/proc/cpuinfo` file.

Operations on vectors Several implementations of the operations on vectors (addition and dot product) are provided to you.

1. Measure the execution time of each function in cycles (time measurement is already implemented)
2. Modify the main function to compute the number of floating point operations per second (MFLOPS or GFLOPS) achieved by each function
3. Analyze the speedups with 2, 4, 8 and 16 threads[1].
4. Compile with the `-O2` option (Optimization level 2). Analyze the performance results with this option.

Operations on matrices The operations on matrices are not yet implemented. It is your work to implement them.

1. Implement the multiplication functions between a matrix and a vector (`mult_mat_vector`), and run the same evaluation as before.
2. Implement the multiplication between functions two matrices (`mult_mat_mat`, and run the same evaluation as before.

4 OpenMP Loop Scheduling

M is a lower triangular matrix. It means that all the values of M matrix above the diagonal are zero. All the values below (and also) the diagonal are useful for the computation. We are going to study the matrix-vector product thanks to the code provided in `ex3.c`.

1. Implement the sequential function of `mult_mat_vect_tri_inf` with M which is lower triangular [2].
2. Implement the parallel OpenMP function `mult_mat_vect_tri_inf1` with **static** scheduling.
3. Implement the parallel OpenMP function `mult_mat_vect_tri_inf2` with **dynamic** scheduling.
4. Implement the parallel OpenMP function `mult_mat_vect_tri_inf3` with **guided** scheduling.
5. Draw a figure with the speedups for 2, 4, 8 and 16 threads.
6. Study the impact of the chunk size on the performance results. For this, you can use the **runtime** schedule to set the chunk size at runtime.

[1] You can select the number of threads to be used by using the environment variable `OMP_NUM_THREADS`.

[2] To optimize performance, the idea is to exclude from the computation the entries that are zero by construction