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

Master 1 Informatique and Master 1 MOSIG

Parallel Algorithms and Programming

Lab 2 – 2020

1 Important information

- This assignment will be graded.
- The assignment is to be done by groups of at most **2** students.
- Deadline: **March, 23, 2020**.
- The assignment is to be turned in on Moodle

1.1 Collaboration and plagiarism

You are encouraged to discuss ideas and problems related to this project with the other students. You can also look for additional resources on the Internet. However, we consider plagiarism very seriously. Hence, if any part of your final submission reflects influences from external sources, you must cite these external sources in your report. Also, any part of your design, your implementation, and your report should come from you and not from other students/sources. We will run tests to detect similarities between source codes. Additionally, we will allow ourselves to question you about your submission if part of it looks suspicious, which means that you should be able to explain each line you submitted. In case of plagiarism, your submission will not be graded and appropriate actions will be taken.

2 Your submission

Your submission is to be turned in on Moodle as an archive named with the last name of the two students involved in the project: `Name1_Name2_lab2.tar.gz`.

The archive will include:

- A report (in `pdf` format[1]) that should include the following sections:
  - The name of the participants.
  - For each exercise:
    - A <u>short</u> description of your work including:
      - The completed tasks
      - The limitations and known bugs, if any.

[1] Other formats will be rejected

* A performance analysis and graphs (when applicable)

- Your source code for each exercise

3 OpenMP Loop Scheduling (4 points)

M is a lower triangular matrix. It means that all the values of the 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. You are provided with the sequential implementation of the matrix-vector product optimized for lower triangular matrices[2] (see file `triangular_matrix.c`).

1. Implement the parallel OpenMP function `mult_mat_vect_tri_inf1` with **static** scheduling.

2. Implement the parallel OpenMP function `mult_mat_vect_tri_inf2` with **dynamic** scheduling.

3. Implement the parallel OpenMP function `mult_mat_vect_tri_inf3` with **guided** scheduling.

4. Draw a figure with the speedups (compared to sequential execution) for 2, 4, 8 and 16 threads.

   - Hint: Choose the chunk size appropriately to have meaningful comparison.

5. Comment the performance you obtain (taking into account the characteristics of the processor you run the experiments on may help).

4 Bubble Sort (6 points)

During this lab, you will implement several algorithms to sort an array of integer elements. You will integrate the *bubble sort* algorithms in the file `bubble.c`.

Bubble sort is one of the most inefficient sorting algorithms. However, it is probably the simplest to understand. At each step, if two adjacent elements of an array are not in order, they will be swapped. Thus, smaller elements will "bubble" to the front, (or bigger elements will be "bubbled" to the back, depending on implementation) and hence the name.

The bubble sort algorithm can be simply described by the following algorithm for an array T of size N to be sorted:

```
do
  sorted = true
  for i in 0..N-1:
    if T[i] > T[i+1]:
      swap T[i] and T[i+1]
      sorted = false
while (sorted == false)
```

7. Implement the sequential bubble sort algorithm.

8. Implement the parallel version of the algorithm.

   - Hint 1:The idea of the parallel version of the bubble sort algorithm is to split the array into several chunks and to have one *bubble* running in each of the chunks. In each iteration, after the parallel step, the elements at the border of two consecutive chunks have to be compared and possibly swapped. A high-level description of the algorithm is as follows:

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

```
while the vector is not sorted:
   for all chunks in parallel:
       execute bubble sort
   for all chunks in parallel:
       swap the values at the border of the chunks
```

Comment 1 Use the scheduling policy that seems to you the more appropriate for this problem.

Comment 2 The provided program (described in file `bubble.c`) takes one parameter N which is the size of the array to be sorted: At the beginning of the program an unsorted array of size 2^N is created.

Comment 3 You are provided with a Makefile to compile your code. By default, the array to be sorted is initialized in such a way that it contains elements sorted in descending order. The variable `RAND_INIT` can be used to test the algorithm on a randomly filled array (call to `make -B RAND_INIT=1`).

5 Quick Sort (5 points)

The *Quick Sort* algorithm is implemented by the `qsort` function of `libc` library.

```
#include <stdlib.h>

  void qsort (void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

The `qsort()` function[3] sorts an array with `nmemb` elements of size `size`. The `base` argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by `compar`, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

9. In file `qsort.c`, use the `qsort` function to implement a sequential sorting function. This requires implementing the `compare` function first.

10. Implement the parallel version of the *Quicksort* algorithm. In this first step, the merging part (see Comment 4) will be done sequentially.

11. Implement the fully parallel version of the *Quicksort* algorithm.

12. Comment on the performance difference you observe between the different versions of the algorithm.

Comment 4 To implement the parallel version of the *Quicksort* algorithm, the principle is to split the array into several chunks, to sort each chunk separately, and finally, to merge consecutive chunks. Some parallelism can also be found during the merging phase: pairs of chunks can be merged in parallel.

Comment 5 The code of a `merge` function is provided to you in the file `qsort.c`.

[3] `man 3 qsort`

6 MergeSort with tasks (5 points)

The sequential `mergesort` algorithm is as follows; its execution is illustrated in Figure 1.

1. If the input sequence has fewer than two elements, return.
2. Partition the input sequence into two halves.
3. Apply the `mergesort` algorithm to the two subsequences.
4. Merge the two sorted subsequences to form the output sequence.

The `Merge` operation employed in step (4) combines two sorted subsequences to produce a single sorted sequence. The same `merge` function as in Exercise 5 can be reused.

In Figure 1, the `MergeSort` is used to sort the sequence 9, 4, 6, 2. The two partitioning phases **P** split the input sequence. The two merging phases **M** combine two sorted subsequences generated in the previous phases.

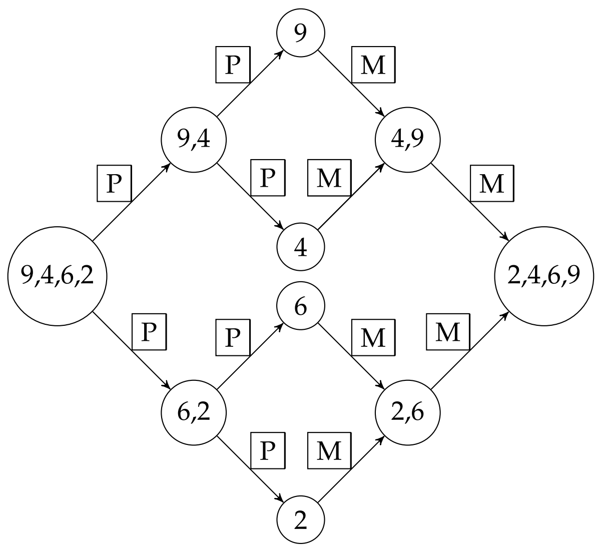

Figure 1: Mergesort execution on a sequence of four elements.

11. Implement the sequential `MergeSort` function in file `mergesort.c`.
12. Implement the parallel algorithm *Mergesort* using OpenMP tasks.
13. Propose a modification to reduce the number of created tasks (and improve performance)

7 Bonus step: Other sorting algorithms

If you would like to further study this topic, we suggest you to implement a parallel version of the following sorting algorithms and to evaluate their performance:

- Odd-Even transposition sort (`http://parallelcomp.uw.hu/ch09lev1sec3.html`)
- Bitonic sort (`https://www.inf.hs-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm`)