Parallel Computing, 2022S

Assignment 3: OpenMP

Assignment #3: OpenMP

Your assignment consists of two parts:

1. **Parallel Length of the Longest-Common-Subsequence (LLCS)**
   Implement a parallel OpenMP version of LCS problem

2. **Optimized Parallel Length of the Longest-Common-Subsequence (LLCS)**
   Identify and remove bottlenecks to improve the performance of your first implementation.

To get a positive grade you need to at least submit the first part of the assignment accompanied by the report.

Length of the Longest Common Subsequence (LLCS)

Problem statement:

Given a set of sequences determine the length of the longest subsequence common to all input sequences.

(do not confuse this problem with the Longest Common Substring, where all elements of the substring need to be adjacent)

Example:

- Assume we have two sequences:

  Sequence X: A B B D C

  Sequence Y: B A B D C

- These two sequences share the following subsequences:

{A}, {B}, {C}, {D}, {A B}, {A D}, {A C}, {B B}, {B C}, {B D}, {D C}, {A B C}, {A B D}, {A B D C} and {B B D C}

The Longest Common Subsequences between sequences X and Y are **{A B D C}** and **{B B D C}**. Meaning that the **length of the longest common subsequence (LLCS)** is equal to **4**.

Dynamic Programming LLCS

Step 1:
Allocate memory for a len(X)+1 x len(Y)+1 matrix M

		A	B	B	D	C
B						
A						
B						
D						
C						

Dynamic Programming LCS

Step 1:
Allocate memory for a len(X)+1 x len(Y)+1 matrix M

Step 2:
Initialize a length(X) x length(Y) matrix where the first row and column are filled with 0's .

		A	B	B	D	C
	0	0	0	0	0	0
B	0					
A	0					
B	0					
D	0					
C	0					

Dynamic Programming LCS

Step 3:
Scan the matrix in row-major order, while applying the following transformations:

```
If X[j] = Y[i]:
   then M[i+1][j+1] := M[i][j] + 1

If X[j] != Y[i]:
   Then M[i+1][j+1] := max(X[j],Y[i])
```

```
E.g.:
   X[0]!=Y[0], M[1][1] := max(X[0],Y[0])

   X[1]=Y[0], M[1][2] := M[0][1] + 1
```

		A	B	B	D	C
	0	0	0	0	0	0
B	0	0	1			
A	0					
B	0					
D	0					
C	0					

Dynamic Programming LLCS

Step 1:
Allocate memory for a len(X) x len(Y) matrix M

Step 2:
Initialize a length(X) x length(Y) matrix where the first row and column are filled with 0's .

Step 3:
Scan the matrix in row-major order, while applying the following transformations…

Step 4:
Repeat **Step 3** until all entries of M are filled.

		A	B	B	D	C
	0	0	0	0	0	0
B	0	0	1	1	1	1
A	0	1	1	1	1	1
B	0	1	2	2	2	2
D	0	1	2	2	3	3
C	0	1	2	2	3	4

Dynamic Programming LLCS

At this point the algorithm to find the <u>Length of the Longest Common Subsequence</u> is complete. (the value is in `M[len(X)+1][len(X)+1]`)

		A	B	B	D	C
	0	0	0	0	0	0
B	0	0	1	1	1	1
A	0	1	1	1	1	1
B	0	1	2	2	2	2
D	0	1	2	2	3	3
C	0	1	2	2	3	$\boxed{4}$

LLCS: Serial Implementation

On the right side you can find a **serial version of the LLCS problem**. This is the serial version that is provided in Moodle.

The function **llcs_serial** receives **X**, **Y** and **M** as arguments and it is assumed that these variables were already **initialized**. (I.e. **step 1 and 2** were already computed.)

As you can see nested for-loop scans matrix **M** in **row-major order** and at it assigns a value to matrix entry **M[i+1][j+1]** per iteration. (i.e. **step 3**)

Once the algorithm halts, it **returns** the value of the **llcs** by resorting to the value held by variable **`entries_visited`**.

```
int llcs_serial(const char* X, const char* Y, unsigned int* M)
{
   unsigned long long entries_visited = 0;

   int i = 0;
   while(i < LEN)
   {
      int j = 0;
      while(j < LEN)
      {
            if(X[i] == Y[j])
                M[i+1][j+1] = M[i][j] + 1;
            else if(M[i+1][j] < M[i][j+1])
                M[i+1][j+1] = M[i][j+1];
            else
                M[i+1][j+1] = M[i+1][j];
            entries_visited++;
            j++;
        }
        i++;
    }
    return entries_visited;
}
```

Task #1 (Implementation)

Implement a parallel version of the previous code with following requirements:

- The function must be called llcs_parallel and have the following signature:
  - `int llcs_parallel(const char* X, const char* Y, char* M)`
- The value of **`entries_visited`** must be incremented at each iteration of the nested for-loop. After leaving the parallel region the value of this variable must be equal to **`LEN * LEN`**, where **`LEN`** represents the size of strings **X** and **Y**. Do <u>**NOT hardcode**</u> this value after leaving the parallel region.
- Use <u>**explicit data-scoping**</u> for variables declared outside of the parallel region, e.g. use the **`default(none)`** clause at the start of your parallel region.
- You are free to use any OpenMP construct/clauses, with the <u>**exception**</u> of **tasks**, **taskgroup**, etc…
- In order to obtain 100% score on this task your solution must achieve a speedup of 2.5X for 16 cores.

Task #1 (Report)

Write a succinct report where you include:

- The reasoning behind your implementation choices and data scoping used.
- A speedup plot between the performance of the serial version of the code against the parallel version implemented by you (**for 2, 4, 8, 16 and 32 threads**).
  - **Note:** the Alma cluster contains 16 physical cores and 32 hyperthreads, keep this in mind during your analysis.
- Finally and most importantly, **identify and justify** the main source of overhead or contention present in your parallel implementation. This step will be fundamental for **Task #2**, where you will try to improve the performance of this approach.

60% of this assignment score is attributed to the report

Task #2 (Implementation)

Implement an optimized version of your previous OpenMP code with the following requirements:

- The function must be called llcs_parallel_opt, and have the following signature:
  - `int llcs_parallel_opt(const char* X, const char* Y, char* M)`
- The value of variable `entries_visited` is computed like in the previous version.
- Use **explicit data-scoping** for variables declared outside of the parallel region, e.g. use the **`default(none)`** clause at the start of your parallel region.
- Again, do **NOT** use OpenMP constructs related to tasking.
- In order to obtain 100% score on this task your solution must achieve a speedup of 8X for 16 cores.

Task #2 (Report)

Again, write a succinct report you include:

- How did you overcome the overhead/contention identified in **Task #1**.
- The reasoning behind your implementation choices and data scoping used.
- A speedup plot where you compare the performance of all versions (**for 2, 4, 8, 16 and 32 threads)**.
- Finally **identify and justify** a possible source of overhead (related to parallelism) present in your optimized version. (No need to implement anything, just present your reasoning).

60% of this assignment score is attributed to the report

Source Code/Compilation/Execution

The source files for this assignment can be found in the course moodle page, under **omp-assignment.**

In this folder you will find the following items:

- **omp-assignment.c*:** this is the driver program to compute the llcs problem.
- **X.in and Y.in*:** these files contain strings X and Y and are read by the driver program.
- **implementation.h:** this is the only file you should modify to complete this assignment.
- **makefile*:** this file contains the compilation commands for all versions

Run `Make` to obtain the executables: `llcs_serial`, `llcs_parallel` and `llcs_parallel_opt`.

Use slurm to run your code in one of ALMA's node:

- "srun --node=`1./executable_name`", where `executable_name` is one of the previous executables.

*** Do NOT modify these files**

Implementation Guidelines

Do **<u>NOT</u>** modify the LLCS kernel (code inside the nested loop) in any way:

- There is no need to modify the kernel to achieve the speedups desired,
- Modifying the kernel will difficult the speedup comparison between versions.

In "**<u>implementations.h</u>**" you will find functions **<u>llcs_serial</u>**, **<u>llcs_parallel</u>** and **<u>llcs_parallel_opt</u>**:

- Complete the implementation of llcs_parallel and llcs_parallel_opt.

If it helps, you can assume the length of string X and Y is the same, and you can also assume this length is constant. (**Note:** the length of string X and Y is defined in macro `#define LEN 51200`.)

Submission Guidelines

1. **Put your files in:**
   - ~aMatrNr/omp-assignment
2. **Make a zip-file with that contains:**
   - source files provided in moodle
   - Input files (“X.in” and “Y.in”)
   - Your version of “implementation.h”
   - Makefile
   - Your report (.pdf)

Upload it to Assignment 3 on Moodle before the deadline: 22.06.2022