

Parallel Architectures and Programming Models, 2023W

Assignment 1: OpenMP Tasking, Roofline Model

PROGRAMMING PART :: SHORT DESCRIPTION

Implement efficient task-parallel versions of the Global Pair-wise Sequence Alignment algorithm using:

1. OpenMP **taskloop** and related constructs only
2. OpenMP explicit **tasks** and related constructs only

Remarks for both variants:

- Use **explicit data scoping** for variables that you use in OpenMP directives
- Use **only tasking** and related constructs and clauses (parallel is naturally ok)
 - **No worksharing constructs such as “for” and “section/sections”** but **single** or **master** are fine
- Try at least **3 different task granularity settings** (amount of work per task)
 - E.g, a fine-grained task would process only 1 element, while the coarser grained task would process more elements sequentially
- Achieve the required speedup on Alma (see “assignment description” slides)
- Write a report describing the details of your work

Hints: You can use constructs like task, and **taskloop**, and all clauses, such as those for data scoping, **reduction**, **in_reduction**, **depend**, if, **grainsize**, **num_tasks**, **nowait**, and similar

GLOBAL PAIR-WISE SEQUENCE ALIGNMENT (GPSA)

Problem statement

- Given a pair of sequences, calculate the best possible alignment (the one that gives us the highest score)
- The optimal path must stretch from beginning to end in both sequences (global alignment)
- i.e., maximize the number matches and minimize the number of mismatches and gaps

Example with (two short sequences)

- Sequence X: A T A
- Sequence Y: A G T T A

A-T-A A--TA
AGTTA AGTTA

Best similarity
scores

These two sequences can have different alignments

- Substitution Matrix
 - Score/penalty for matches/mismatches
 - Penalty for gaps (we use the linear gap penalty)
- We will use a substitution matrix in this example (see `blosum62.txt`)

GPSA :: DYNAMIC PROGRAMMING

Step 1: Preparation (already implemented)

- **Allocate** a matrix with (let's call this matrix **S**)
X.length()+1 rows and
Y.length()+1 columns
- **Allocate** a “substitution” matrix (let's call it **SUB**)
 - A way of scoring two characters in two sequences
 - Block Substitution Matrix (BLOSUM)*
 - [Already provided](#), loaded from a file (blosom62.txt file (24x24))
 - Can also be generated
- **Map** characters (letters) to indices (let's call it **cmap**)
 - We will use the map in the code to access the match/mismatch values

A	G	T	*
0	1	2	3

$$SUB_{a,b} = \begin{cases} 1, & \text{if } a==b \\ -1, & \text{if } a!=b \end{cases}$$

	A	G	T	*
A	1	-1	-1	-2
G	-1	1	-1	-2
T	-1	-1	1	-2
*	-2	-2	-2	-2

**Example substitution matrix
with gap_penalties**

For simplicity, with 3 letters A, G and T

*https://www.ncbi.nlm.nih.gov/IEB/ToolBox/C_DOC/lxr/source/data/BLOSUM62

This substitution matrix also corresponds to a simple scheme:
match=1, mismatch=-1, gap = -2

GPSA :: DYNAMIC PROGRAMMING

Step 2: Calculating Similarity Matrix (S) (to parallelize)

- **Initialize** the first row and column with initial values i.e., gap penalties (boundary conditions)
- We will use the lowest score from the Substitution Matrix “-2” in this example

```
gap_penalty = SUB[0][ cmap['*'] ];

for (int i = 1; i < rows; i++)
    S[i][0] = i * gap_penalty;

for (int j = 0; j < cols; j++)
    S[0][j] = j * gap_penalty;
```

Initialize the 1st row and 1st column

*Note that we use the **cmap** to get the actual index in the SUM matrix

This substitution matrix also corresponds to a simple scheme:
match=1, mismatch=-1, gap = -2

lowest

	A	G	T	*
A	1	-1	-1	-2
G	-1	1	-1	-2
T	-1	-1	1	-2
*	-2	-2	-2	-2

Simple substitution matrix
for this example

		A	G	T	T	A
	0	-2	-4	-6	-8	-10
A	-2					
T	-4					
A	-6					

This is the result

Similarity matrix 6x6
5 letter long sequences

GPSA :: DYNAMIC PROGRAMMING

Step 2: Calculating Similarity Matrix (S) (to parallelize)

- Scan the matrix in row-major order, and apply the following:

$$S_{(i,j)} = \max \begin{cases} S_{(i-1,j-1)} + SUB_{x[i],y[j]} & \rightarrow \text{match/mismatch} \\ S_{(i-1,j)} + gap_penalty & \rightarrow \text{gap} \\ S_{(i,j-1)} + gap_penalty & \rightarrow \text{gap} \end{cases}$$

- We will use the following kernel to compute it:

```
float match = S[i-1][j-1] +
              SUB[ cmap[X[i-1]] ][ cmap[Y[j-1]] ];
float del = S[i-1][j] + gap_penalty;
float insert = S[i][j-1] + gap_penalty;
S[i][j] = std::max( {match, del, insert} );
```

	A	G	T	*
A	1	-1	-1	-2
G	-1	1	-1	-2
T	-1	-1	1	-2
*	-2	-2	-2	-2

Substitution matrix

		A	G	T	T	A
	0	-2	-4	-6	-8	-10
A	-2	1				
T	-4					
A	-6					

Similarity matrix 6x6

5 letter long sequences

GPSA :: DYNAMIC PROGRAMMING

Step 2: Calculating Similarity Matrix (S) (to parallelize)

- **Scan** the matrix in row-major order, and apply the following:

```
float match = S[i-1][j-1] +
              SUB[ cmap[X[i-1]] ][ cmap[Y[j-1]] ];
float del = S[i-1][j] + gap_penalty;
float insert = S[i][j-1] + gap_penalty;
S[i][j] = std::max( {match, del, insert} );
```

- [i==1 and j == 1]:
 match = S[0][0] + match; // 1, S[0][0] is 0 and SUB[A][A] is 1
 delete = -2 + gap_penalty; // -4, S[0][1] is -2 and gap_penalty is -2
 insert = 0 + match; // -4, since gap_penalty is -2
 S[i][j] = max(1, -4, -4); // 1

	A	G	T	*
A	1	-1	-1	-2
G	-1	1	-1	-2
T	-1	-1	1	-2
*	-2	-2	-2	-2

Substitution matrix

		A	G	T	T	A
	0	-2	-4	-6	-8	-10
A	-2	1				
T	-4					
A	-6					

Similarity matrix 6x6

5 letter long sequences

GPSA :: DYNAMIC PROGRAMMING

Step 2: Calculating Similarity Matrix (S) (to parallelize)

- Scan the matrix in row-major order, and apply the following:

```
float match = S[i-1][j-1] +
              SUB[ cmap[X[i-1]] ][ cmap[Y[j-1]] ];
float del = S[i-1][j] + gap_penalty;
float insert = S[i][j-1] + gap_penalty;
S[i][j] = std::max( {match, del, insert} );
```

- [i==1 and j == 1]:


```
match = S[0][0] + match; // 1, S[0][0] is 0 and SUB[A][A] is 1
delete = -2 + gap_penalty; // -4, S[0][1] is -2 and gap_penalty is -2
insert = 0 + match; // -4, since gap_penalty is -2
S[i][j] = max(1, -4, -4); // 1
```
- [i==1 and j == 2]:


```
match = -2 + mismatch_score // -3, S[0][1] is -2 and SUB[A][G] is -1
delete = 1 + gap_penalty // -1, since the left neighbour is 1
insert = -2 + gap_penalty // -6, since the neighbour above is -4
S[i][j] = max(-3, -1, -6) // -1
```

	A	G	T	*
A	1	-1	-1	-2
G	-1	1	-1	-2
T	-1	-1	1	-2
*	-2	-2	-2	-2

Substitution matrix

		A	G	T	T	A
	0	-2	-4	-6	-8	-10
A	-2	1	-1			
T	-4					
A	-6					

Similarity matrix 6x6
5 letter long sequences

You can see how algorithm works here:

https://bioboot.github.io/bimm143_W20/class-material/nw/

GPSA :: DYNAMIC PROGRAMMING

Step 2: Calculating Similarity Matrix (S) (to parallelize)

- Scan the matrix in row-major order, and apply the following:

```
float match = S[i-1][j-1] +
              SUB[ cmap[X[i-1]] ][ cmap[Y[j-1]] ];
float del = S[i-1][j] + gap_penalty;
float insert = S[i][j-1] + gap_penalty;
S[i][j] = std::max( {match, del, insert} );
```

- Iterate** until everything has been calculated
- We use float values only for the example-specific purposes

→ Next, we also need to traceback to align the sequences, and get some statistics ([already implemented](#))

		A	G	T	T	A
	0	-2	-4	-6	-8	-10
A	-2	1	-1	-3	-5	-7
T	-4	-1	0	0	-2	-4
A	-6	-3	-2	-1	-1	-1

Similarity matrix 6x6
5 letter long sequences

GPSA :: DYNAMIC PROGRAMMING

Step 3: Traceback (already implemented)

- Process of deduction of the best alignment
- Begins with the last cell
- We iterate in reverse and look for maximal scores

		A	G	T	T	A
	0	-2	-4	-6	-8	-10
A	-2	1	-1	-3	-5	-7
T	-4	-1	0	0	-2	-4
A	-6	-3	-2	-1	-1	-1

GPSA :: DYNAMIC PROGRAMMING :: SUMMARY

Step 1: Preparation

- Allocation is already implemented

Step 2: Calculating Similarity Matrix (S)

- Only sequential version implemented
- You need to develop/parallelize 2 different versions of this step with OpenMP tasking
- You can start from the sequential version

Step 3: Traceback to aligning the sequences

- Already Implemented

		A	G	T	T	A
	0	-2	-4	-6	-8	-10
A	-2	1	-1	-3	-5	-7
T	-4	-1	0	0	-2	-4
A	-6	-3	-2	-1	-1	-1

GPSA :: SERIAL IMPLEMENTATION

`SequenceInfo::gpsa_sequential(...);`

- Class member function
- Called automatically for you from the **main**
- Nested for-loops scans matrix *S* in row-major order and assigns a value to matrix entry *S*[*i*][*j*] per iteration
- Once the algorithm halts, it returns the number of “**visited**” cells for verification purposes
- Uses float values only for this purposes of this exercise
- This is you starting point, **gpsa_tasks()** and **gpsa_taskloop()** member function will have the same code at start
- Goal is to add OpenMP directives, and transform some loops so that the and the way program iterates over the *S* matrix so that the overall execution time is reduced

```
unsigned long SequenceInfo::gpsa_sequential(float** S, float** SUB,
                                           std::unordered_map<char, int>& cmap)
{
    unsigned long visited = 0;
    gap_penalty = SUB[0][cmap['*']]; // min score

    // Boundary
    for (unsigned int i = 1; i < rows; i++) {
        S[i][0] = i * gap_penalty;
        visited++;
    }

    for (unsigned int j = 0; j < cols; j++) {
        S[0][j] = j * gap_penalty;
        visited++;
    }

    for (unsigned int i = 1; i < rows; i++) {
        for (unsigned int j = 1; j < cols; j++) {
            float match = S[i - 1][j - 1]
                          + SUB[ cmap[X[i - 1]] ][ cmap[Y[j-1]] ];
            float del = S[i - 1][j] + gap_penalty;
            float insert = S[i][j - 1] + gap_penalty;
            S[i][j] = std::max({match, del, insert});

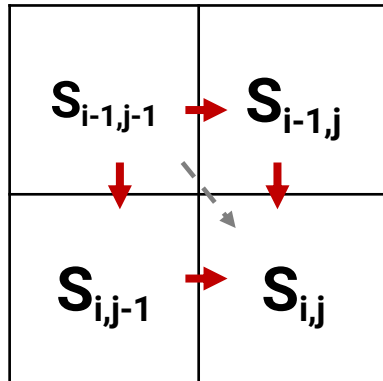
            visited++;
        }
    }

    return visited;
}
```

GPSA :: TASK-PARALLEL IMPLEMENTATION

Data dependences

- Maybe we can iterate the matrix differently?
- Example: **Wavefront (Antidiagonal)**



Tasks

- Still some tasks must wait for previous tasks
 - You must **avoid data races!**
- **Good task granularity?**
 - How much work should each task do?
 - 1 Element per task or more elements per task (blocks/group of elements)?
 - Should all tasks be of the same granularity?

		A	B	B	D	C
	0	-2	-4	-6	-8	-10
0	0					
-2	0					
-4	0					
-6	0					
-8	0					

Iterating over antidiagonals

ASSIGNMENT DESCRIPTION (1/3)

Implement efficient task-parallel implementations of the given algorithm using (1) OpenMP explicit tasks and (2) taskloops.

- Your functions need to be added to “implementation.hpp” and have the following signatures:
 - `unsigned long SequenceInfo::gpsa_taskloop(float** S, float** SUB, std::unordered_map<char, int>& cmap, int grain_size=1);`
 - `unsigned long SequenceInfo::gpsa_tasks(float** S, float** SUB, std::unordered_map<char, int>& cmap, int grain_size=1);`
- There are already templates for the two function above, so you can focus on writing the code inside these two functions
- Granularity modifier is an argument that you can specify on terminal, and you can use to parametrize granularity if you need it
- Remarks:
 - Functions for additional versions may also be provided, e.g. `gpsa_tasks_v2`, `gpsa_taskloop_v2`, etc. For that, you may need to adjust testing as well. Alternatively, you can just make another source file. However, they are not necessary.
 - Typically, there is **no need to change other files** and the rest of the code is purposefully implemented as it is for this specific use case and for this particular input data, and it should not be changed.
 - Data structures are passed as pointers to allow full compatibility with OpenMP Features
- The value of the variable **”visited”** must be incremented at each iteration of the nested for-loop. This value has to be equal to the value produced by the sequential version.
- **Use explicit data-scoping** for variables declared outside of the task generating constructs

ASSIGNMENT DESCRIPTION (2/3)

Are these safe?

Implementation.hpp

- This is the only file that you need to edit
- Analyze loops and see if they can be parallelized
 - There are three, and the last one can be tricky
- Analyze all variable before and after OpenMP **parallel regions**, and check if anything needs synchronized accesses
- Check if loop 3 can be somehow transformed with respect to slide 13
- Adjust granularity with OpenMP clauses and with loop transformations
- Which variables can be made **private** or **firstprivate**, and which need to be **shared**?
- Make sure that you generate task from a single thread
- Taskloop and explicit task will differ both in complexity and possibly in performance
You can ignore most of the other code
- Check [last 10 slides of OpenMP Tasking](#) lecture for additional ideas

```

unsigned long SequenceInfo::gpsa_taskloop(float** S, float** SUB,
std::unordered_map<char, int>& cmap) {
    unsigned long visited = 0;
    gap_penalty = SUB[0][cmap['*']], // min score

1   for (unsigned int i = 1; i < rows; i++) {
        S[i][0] = i * gap_penalty;
        visited++;
    }

2   for (unsigned int j = 0; j < cols; j++) {
        S[0][j] = j * gap_penalty;
        visited++;
    }

3   for (unsigned int i = 1; i < rows; i++) {
        for (unsigned int j = 1; j < cols; j++) {
            float match = S[i-1][j-1]
                + SUB[cmap.at(X[i-1])][cmap.at(Y[j-1])];
            float del = S[i-1][j] + gap_penalty;
            float insert = S[i][j-1] + gap_penalty;
            S[i][j] = std::max({match, del, insert});

            visited++;
        }
    }

    return visited;
}

```

ASSIGNMENT DESCRIPTION (3/3)

Measure performance with **at least 3** different task granularities per implementation version

- E.g, a *fine-grained* task would process only 1 element, while the coarser grained task would process more elements sequentially – are there values that work better?
- The tree values should show at which grain size does the performance starts to drop

Test and measure on Alma

- Test early to avoid peak usage times
- Performance numbers (speedup) for your report is only relevant for the assignment if measured on Alma
- You need to be connect to university network to access it (or VPN), and then
ssh <your_username>@alma.par.univie.ac.at (<https://moodle.univie.ac.at/mod/forum/discuss.php?d=3695157>)

You best effort solution should aim to achieve a speedup of **12+** on Alma for both versions

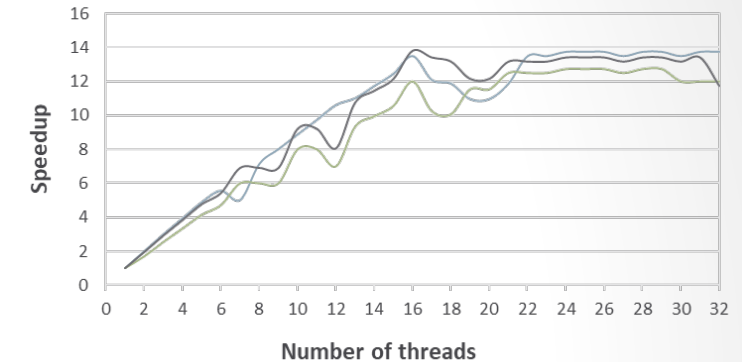
- Better versions go above this speedup with the given dataset(s)

The total number of threads in use should work via OMP_NUM_THREADS environment variables

REPORT

Write a report addressing the following:

- A brief description of your approach (tasks vs taskloop)
- Granularity and the total number of tasks
 - How many tasks are generated in each loop
 - No need for a concrete number, just a description
 - Does this number change or is it statically determined?
 - How many items per task is processed?
 - If this value changes dynamically, describe how
- Dependences
 - How are task dependences handled so that the produced result is correct?
- How data scoping was used and what did you need to declare and why?
- **Speedup graph(s)** and numbers with respect to the **sequential version (~10s on Alma with the default data)**
 - Different variants (and different task granularities) using **1 – 32** threads
- Bottlenecks, observations, and how can your code be improved
 - Do V1 and V2 exhibit different performance? If that is the case, try to explain why!
 - What helped you code run faster and why?



An example how a speedup graph should look like

SUBMISSION

Put your files in a zip file and upload to Moodle

~aMatrNr/(your files)

- Source files + report-a1.pdf

Upload all to Assignment 1 on Moodle before the deadline.

Notes:

- Both code and documentation are required!
i.e., your code without the report **will not be graded positive!**

GETTING STARTED

Serial version in Moodle

- `main.cpp` (main source file)
- `helpers.cpp` (helper classes and functions, e.g., loading a sequence from file, traceback, etc..)
- `implementation.hpp` (your work goes in the two tasking related functions in this file)
- `X.txt`, `Y.txt` (input data - sequences)
- `X2.txt`, `Y2.txt` (input data – sequences with equal lengths)

To compile the code on Alma you need to:

- `/opt/global/gcc-11.2.0/bin/g++ -O2 -std=c++20 -fopenmp -o gpsa main.cpp`

This version is **necessary** on Alma, if the compiler is older, it may compile, but some of the OpenMP constructs and clauses may be ignored. For local development, you need to check your GCC version and modify/remove this switch.

To run with all available threads:

- `srun --nodes=1 ./gpsa`
- To run with 16 threads instead of default 32: `OMP_NUM_THREADS=16 srun --nodes=1 ./gpsa`

Alma system: <http://www.par.univie.ac.at/teach/doc/alma.html>

GETTING STARTED

Usage ./gpsa

- x provide an input file for the first sequence
- y provide an input file for the second sequence
- exec-mode run different versions of the program:
 - 0 – all versions, 1 – sequential version only
 - 2- taskloop version only, 3 – explicit tasks version only
- grain-size an optional parameter passed to functions, which you optionally use to easier testing of task granularity
- save-to an optional parameter to specify output filename for the sequential alignment

There is also a “runall.sh” script that you can run to executes configurations on 1 to 32 threads. You can use it after you compile your code, as you need to pass the executable as the first argument, e.g., “sh runall.sh ./gpsa”

GETTING STARTED

Sequences

X.txt, Y.txt, size: [18481x18961]

- Random, big sequences.

X2.txt, Y2.txt, size: [16383x16383]

- [It may be easier to start with](#), since everything divides well with 16,32, 64, 128, 256, and both sequences have the same size, which means that it transform the for-loops if needed.

simple1.txt, simple2.txt, size: [5x6]

- Small sequences that you can use for debugging.
- You can change this file as you please.

To switch to a different sequence, you can pass the command line names:

```
./gpsa --x X2 --y Y2
```