Approximate Pattern Matching

Riade Benbaki, Ali Mammadov

INF560 Project defense

Plan

1. Approximate pattern matching problem
2. Parallelism approaches
   - MPI
   - OpenMP
   - CUDA
   - Mixing them all together
3. Results and experiments

1 - Approximate pattern matching problem

- We consider each pattern independently
- For each pattern, we check if there is a match between the pattern and the string and the current starting position
- We count the total number of matches obtained

What can we parallelize ?

- Over the different patterns
- Over each starting position in the file
- Not over the distance computation
    - Based on Levenshtein distance -> Hard to parallelize

MPI

- Split the file evenly over all MPI ranks (data parallelism)
- Every rank received it’s own part + some shadow cells necessary for the last computations
- An MPI reduction at the end to sum up the results of all ranks

OpenMP

- For every MPI rank, we use OpenMP threads to parallelize the loop over all starting positions
- Every threads needs to read the string, but not modify it
- Every thread needs a private array for computation

CUDA

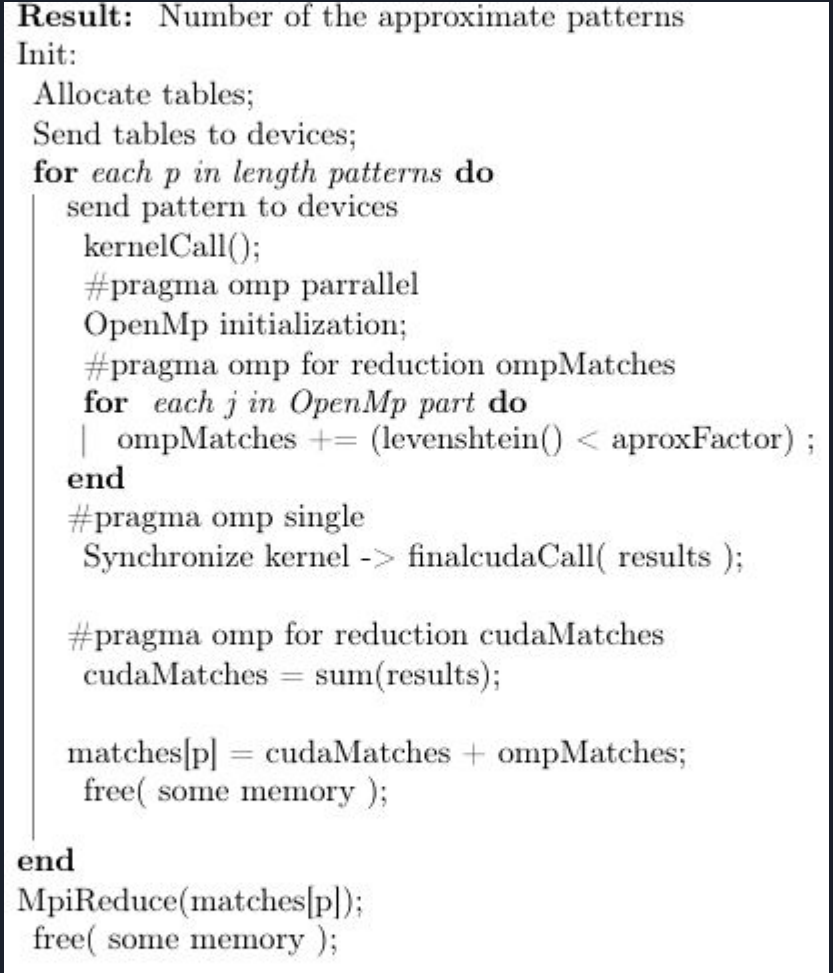
- The deepest level that can be parallelized is the previous
- We therefore use CUDA with OpenMP, each of them processing a part of the chunk of the current MPI rank
- One cuda thread needs access to
  - String buffer -> size of buf_size = n_bytes/size
  - Pattern -> size of at most max_pat
  - An array column (private to each thread) -> n_threads array of ints
  - A results array -> n_threads array of ints
- All this for each MPI process -> multiple ranks may share same GPU
- Worst case scenario : all ranks share the same GPU

$$[nth_{max}(max\_pat + 2) \times sizeof(int) + buf\_size \times sizeof(char)]\ size \leq freeGpUMemory()$$

- Use a ratio value to divide the work between CUDA and OpenMP

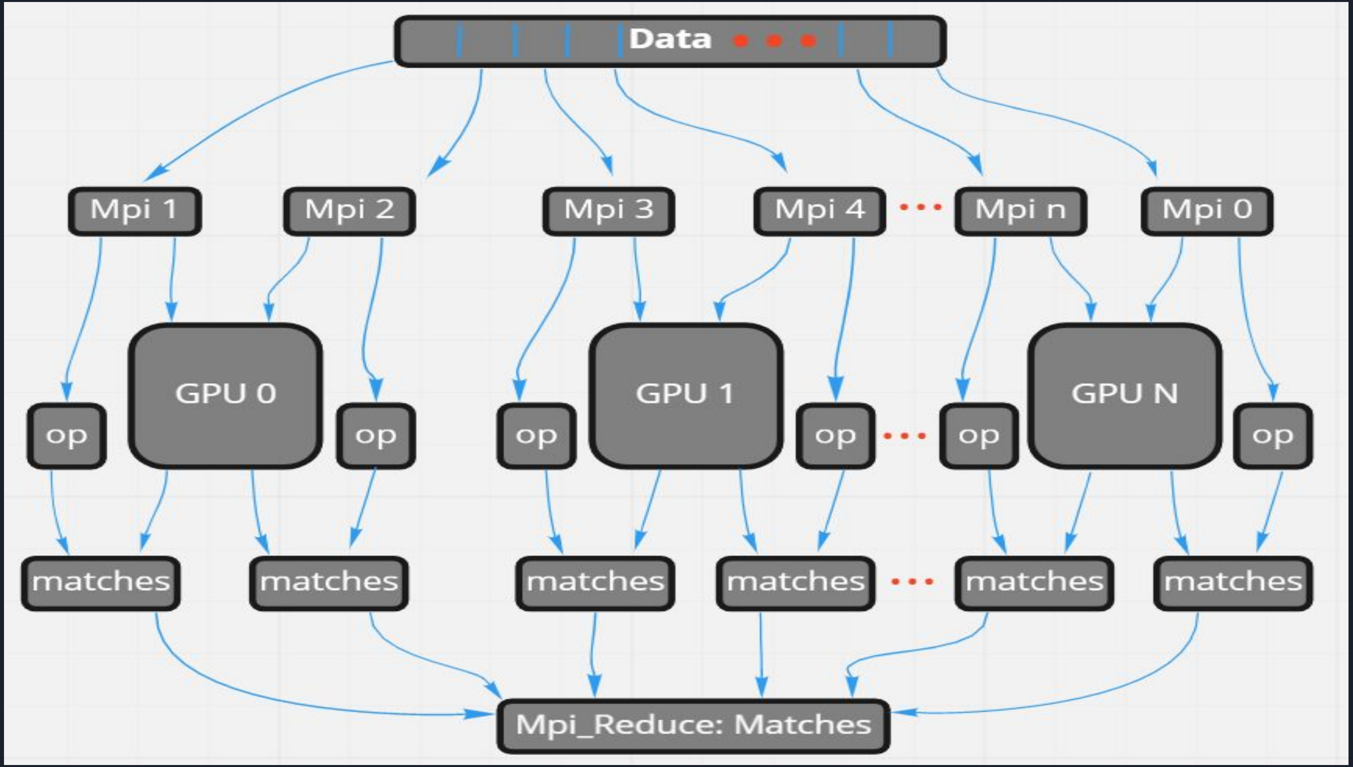
Parallel Algorithm

- kernelCall() - calls kernel function
    - Asynchronous
- finalcudaCall() - Synchronizes kernel
    - Returns results
- Mpi_Reduce() - For getting the
    - results from each process

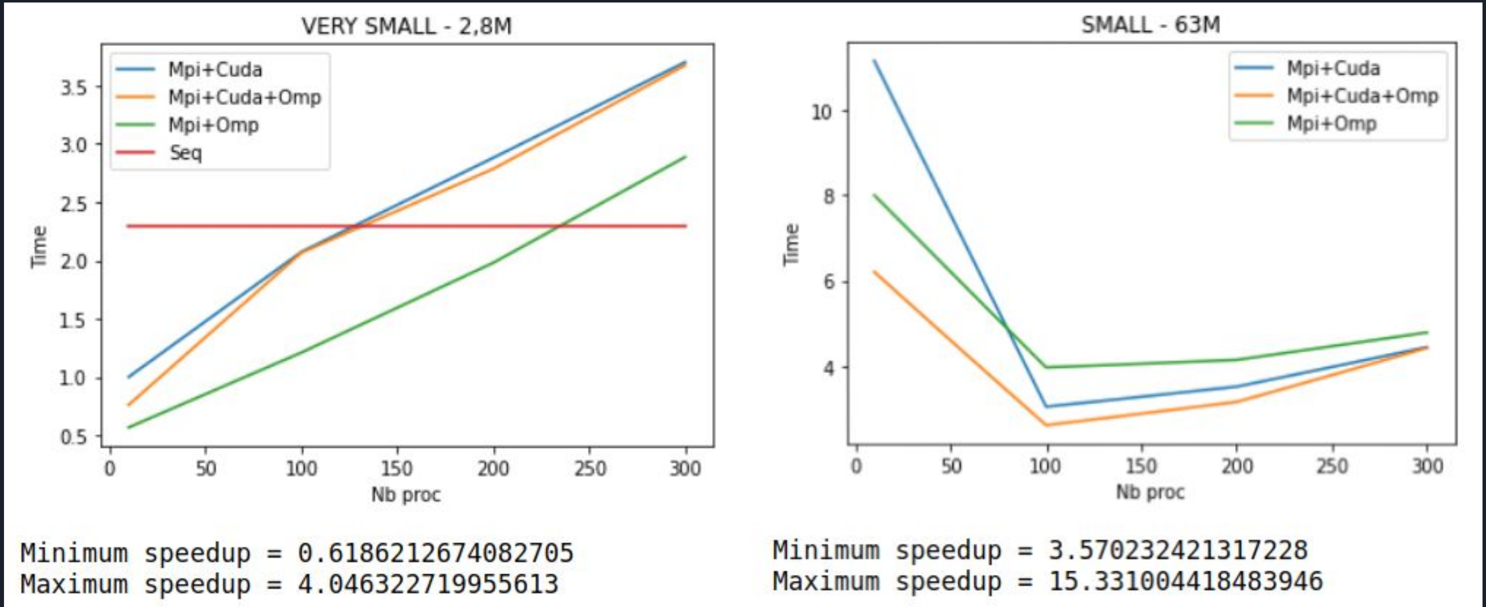


```
Result:  Number of the approximate patterns
Init:
 Allocate tables;
 Send tables to devices;
 for each p in length patterns do
    send pattern to devices
     kernelCall();
     #pragma omp parrallel
     OpenMp initialization;
     #pragma omp for reduction ompMatches
     for  each j in OpenMp part do
        ompMatches += (levenshtein() < aproxFactor) ;
    end
    #pragma omp single
     Synchronize kernel -> finalcudaCall( results );

    #pragma omp for reduction cudaMatches
     cudaMatches = sum(results);

    matches[p] = cudaMatches + ompMatches;
     free( some memory );

end
MpiReduce(matches[p]);
 free( some memory );
```

Parallel Algorithm



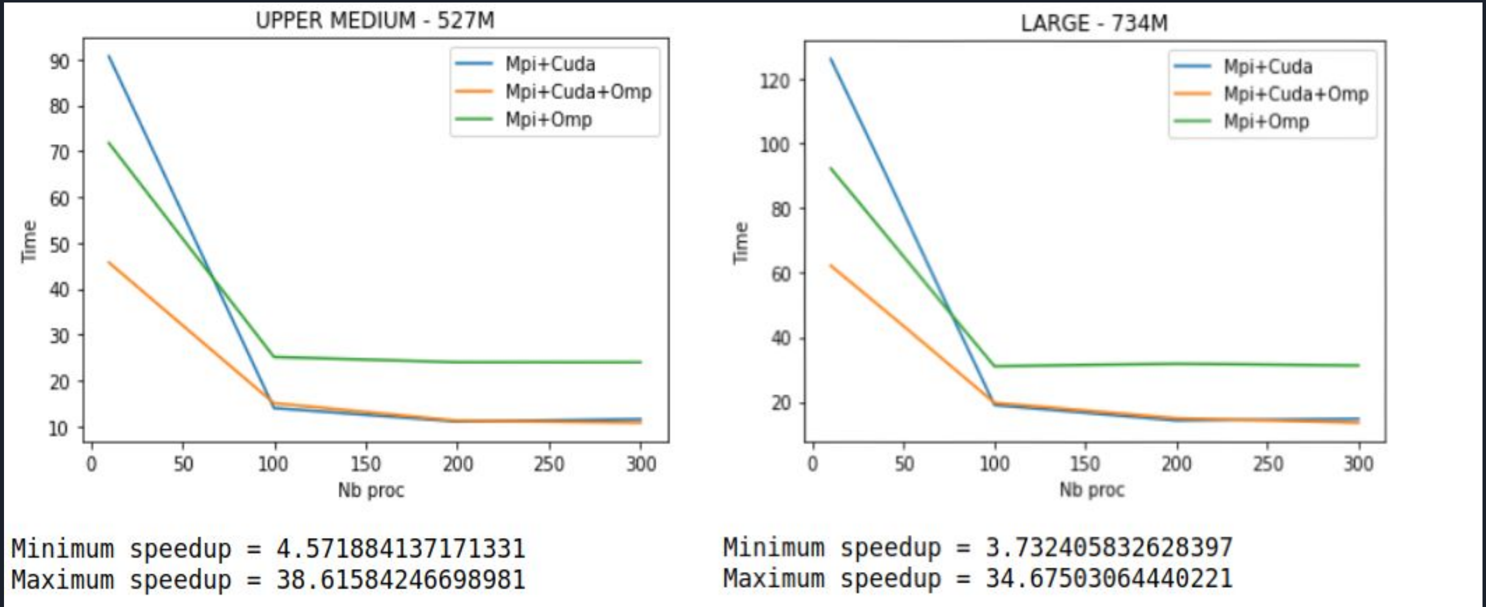
Experimental evaluations - small files



```
Minimum speedup = 0.6186212674082705
Maximum speedup = 4.046322719955613
```

```
Minimum speedup = 3.570232421317228
Maximum speedup = 15.331004418483946
```

Experimental evaluations - large files



```
Minimum speedup = 4.571884137171331
Maximum speedup = 38.61584246698981
```

```
Minimum speedup = 3.732405832628397
Maximum speedup = 34.67503064440221
```

Thank you for your attention