Parallel Programming – Project Report

Sachith Sri Ram Kothur and Sankalp Kolhe

1. Project Thesis

We analyze the task of BLEU score computation (an important step in the evaluation of Machine Translation systems) from a parallelization standpoint. We analyze the aspect of parallelizing the task with a shared memory model vs a distributed memory model. Analyzing the bottlenecks in the parallelization of this task we justify which method of parallelism is best suited for the current scenario.

2. Introduction

BLEU score[1] is a metric designed to measure the similarity between two sentences. It is widely used in the field of machine translation where we are given a target translation – one that is generally outputted by our model and the reference translation – the gold standard output which is typically done by a human translator. We then compute the BLEU score between the target and reference translation with higher BLEU score indicating better performance of our model.

The vanilla implementation of BLEU[2} score is as follows :

```
Let w be the total number of words [unigram/bigram/..n-gram] in the candidate sentence;

For every word in the candidate sentence that appears in the reference :
    if $m_c$ is the number of times it occurs in the candidate sentence and
    if $m_w$ is the number of times it occurs in the reference sentence
    then $sum = sum + min(m_c , m_w)$

$BLEU = sum/w$
```

3. Motivation

- Though BLEU score computation is not the most time-consuming part of the Machine Translation (MT) pipeline, for large text corpora (which is a commonality these days due to Neural Machine Translation approaches), computing the BLEU score does take considerable amount of time
- Typically, BLEU computation is also done on the same compute intensive hardware that run the training phase of MT pipeline. This means that a default serial implementation is not accessing the full benefits of the powerful hardware it runs upon.

- Not much work has been done on making the BLEU score calculations more efficient to take use of the parallelism of underlying hardware.

4. Work done

We have parallelized the computation of BLEU score using both the *shared memory paradigm* and the *distributed memory paradigm*. Details are below.

4.1 Shared Memory Implementation

We have written our code for BLEU computation in C++ and have used OpenMP to distribute the load of the computation among multiple threads sharing the same memory. The code is in *Parallel-BLEU.cpp*

The basic overview of the code is as follows:

- Thread(0) or master thread reads the target file and reference file and stores in memory. I/O and computation are done in chunks to avoid loading the whole file into memory
- We then spawn multiple threads , each taking its share of target and reference sentences and computes the BLEU score
- The final BLEU score is calculated by reducing the BLEU computations of individual threads

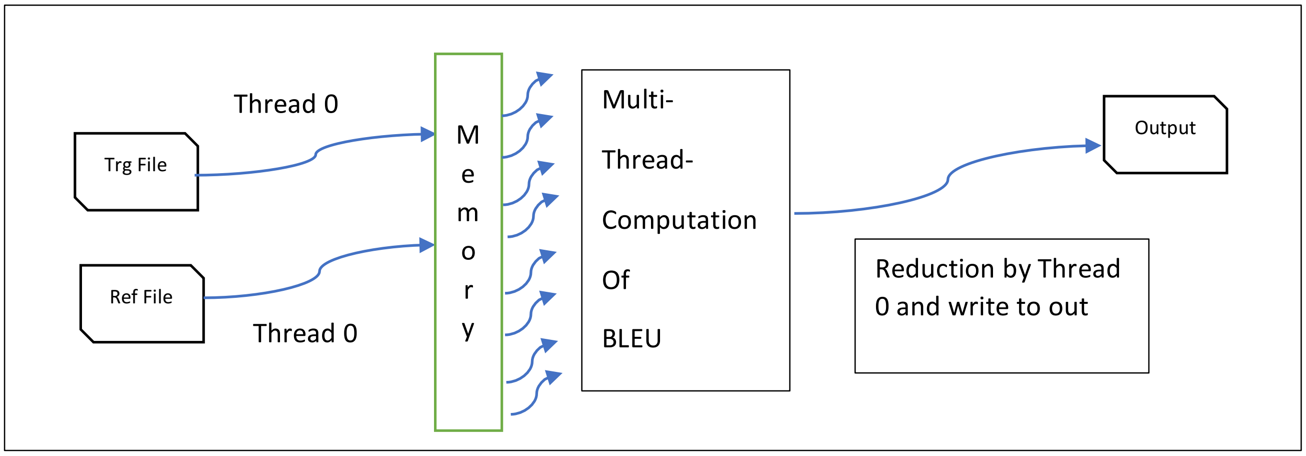
Thread 0
Trg File
Ref File
Thread 0
M e m o r y
Multi- Thread- Computation Of BLEU
Output
Reduction by Thread 0 and write to out

Fig 1. Shared Memory Architecture

4.2 Distribured Memory Implentation

We used MPI for distributed parallelism. The code is in Distr-BLEU.cpp. The basic overview of the code is below:

- Process 0 does all the I/O. It reads the input files and divides the sentences among all the processes. It then sends the share of each process over the network.
- All other processes receive their of data from Process 0
- BLEU score computation is now done in parallel among all the processes

- All processes send their individual BLEU scores to process 0 which reduces them and outputs the final value

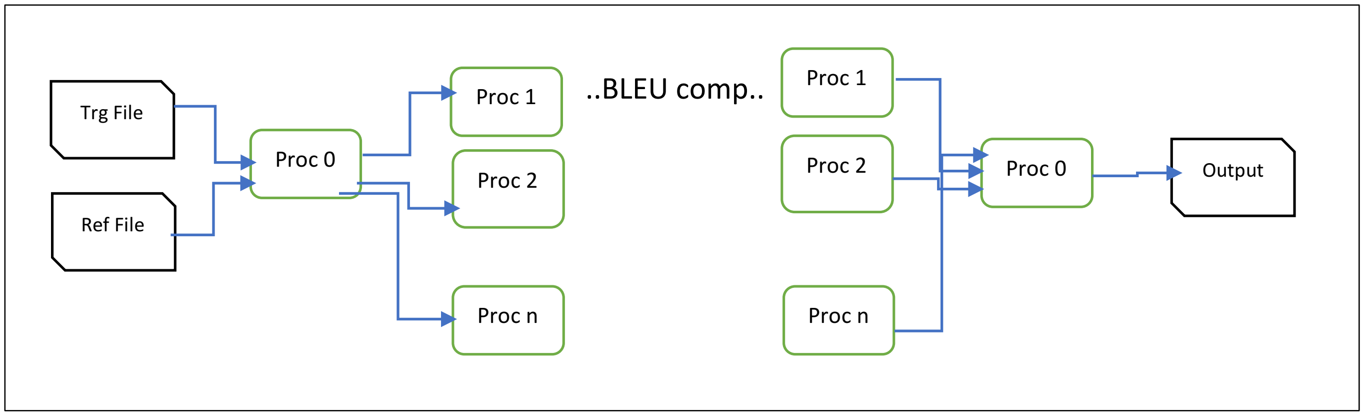

Fig2. Distributed Memory architecture

5. Data-Set Size and Hardware Architecture

- We are using the OpenSubtitles English-to-German dataset from OPUS[3]. It consists of 26 million source-target sentence pairs taking around 2GB of memory (1GB for each of target and reference files)
- We use Amazon **c3.2x large** machines for running the tests. Each node has 8 vcpu's with 15G memory
- For distributed architecture we deployed a cluster of **four c3.2x large nodes** using star-cluster
- For shared memory architecture , we run it on a **single 3.2x node with 4 threads**. We could have gone upto 8 threads with almost linear increase of speedup but we stopped till 4 so that we can compare it with our MPI architecture (having same number of parallel execution units).

6. Results

We have run our experiments on multiple sizes of training data from 3% to 200%. We wanted to see how the two models fare with increasing data-size to offset initial startup costs.

We also plotted the IO time vs training data. Here IO includes the time taken to read the files by process 0 and allot appropriate chunks to other processes.

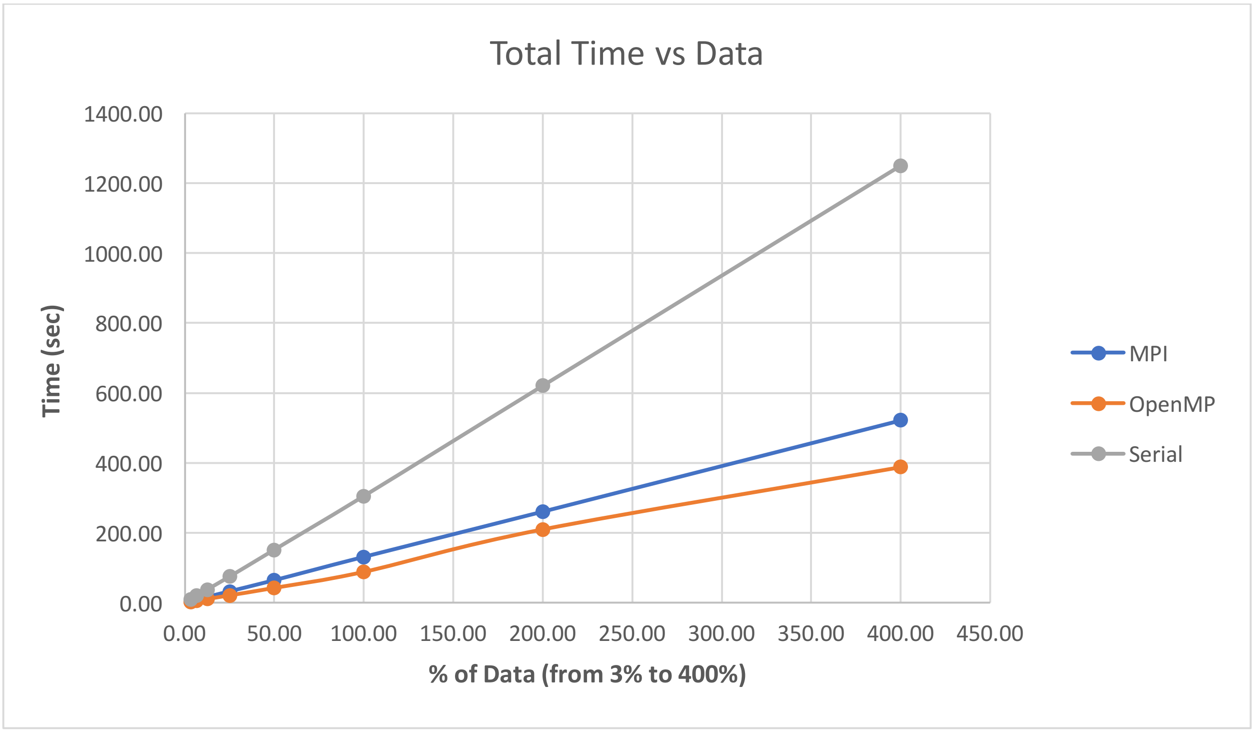

Fig3. Total time for different implementations

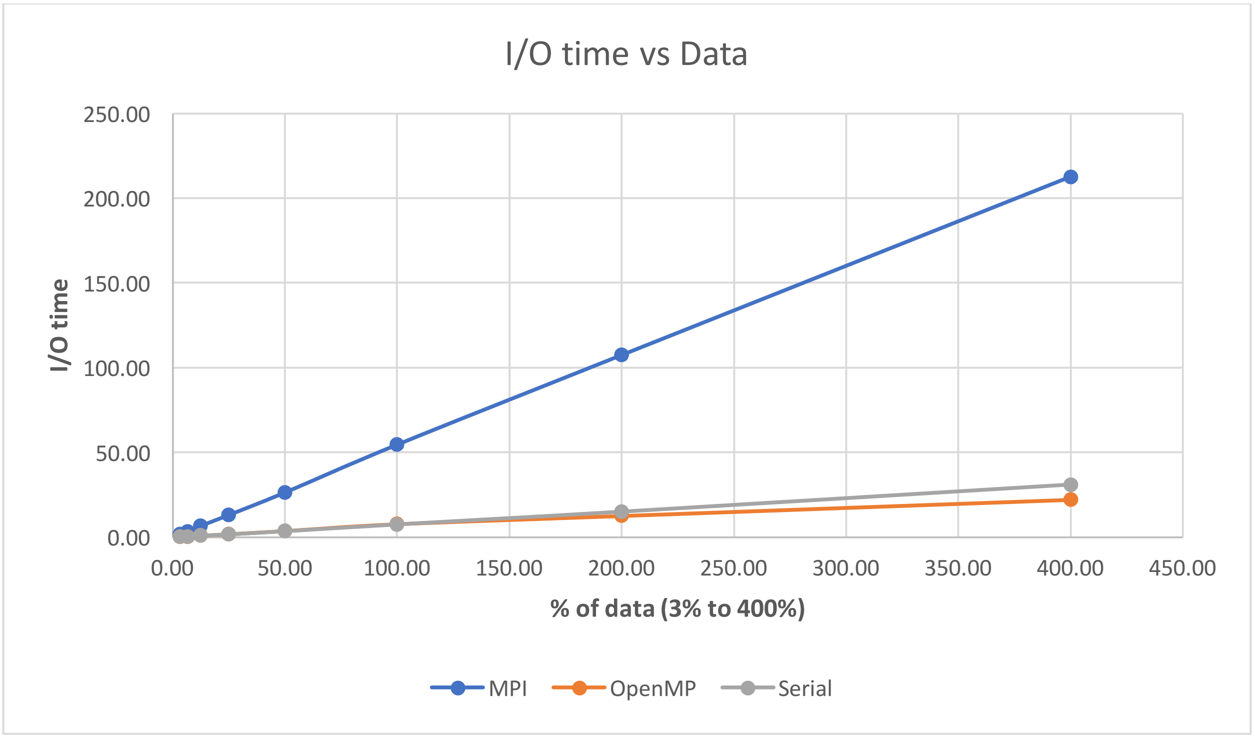

Fig4. I/O time for different Implementations

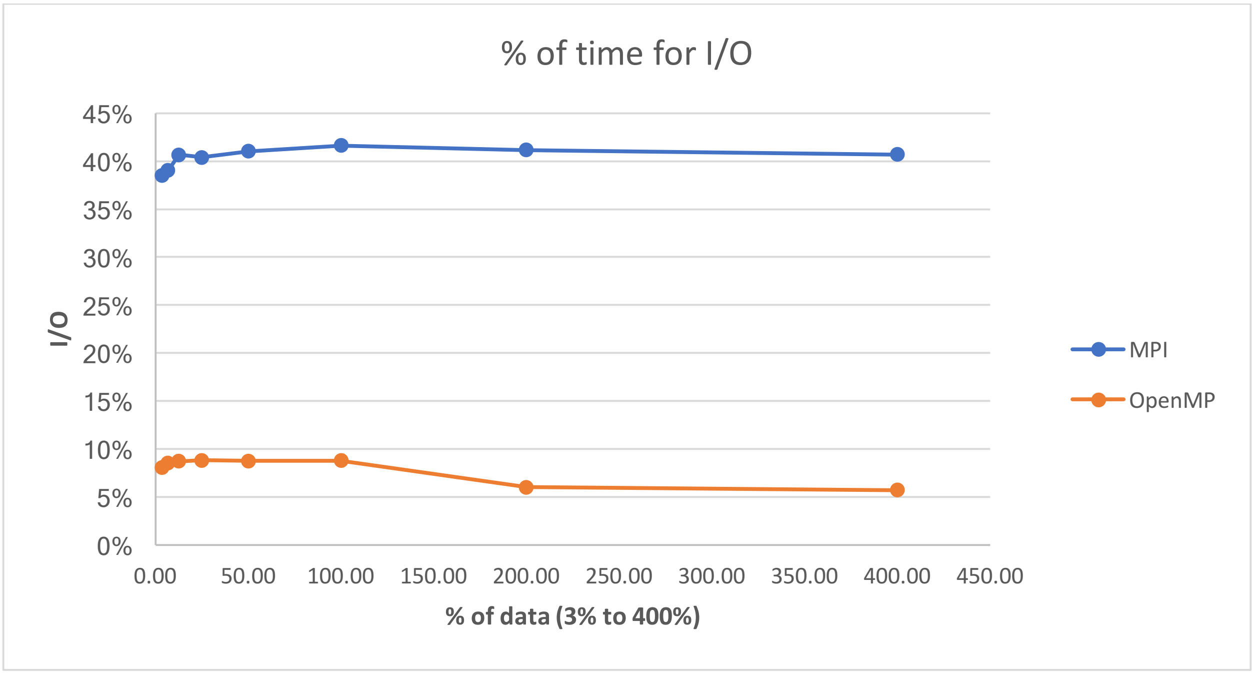

Fig5. Percentage of time for I/O

6. Analysis

- Based on Fig3, there is a *considerable performance gain* in parallelizing the BLEU score computation. The data-sets we have chosen are on the smaller end of spectrum for Machine Translation tasks and as the benefits scale linearly with data size, for huge datasets the performance gain is very high.

- Among the shared and distributed paradigms, we see that *OpenMP implementation gives better performance than MPI* for the same number of parallel execution units. So the nature of this problem is better suited to a shared memory framework. This is because of two reasons:

    - Parallelizing the computation among #sentences ensures that very little data (or no data) needs to shared among multiple processing units. So there is little need to access shared data and hence multi-threading gives optimum efficiency
    - Parallelizing over multiple nodes (MPI) needs data to be sent over the network which proves to be a major bottleneck (See next point).

- **Fig4** and **Fig5** showed us interesting results. When we just plotted the I/O time (includes initial file reading by process 0 and sending appropriate data to other processes) , we see that MPI I/O proves to be a major bottleneck. This is because process 0 needs to read the file and send (*over the network*) the required data to all other processes. This hogs up considerable execution time for packing and communicating the data over the network.
- This answers the problem of bottleneck in our thesis proposal that in parallelizing BLEU score I/O proves to be a major bottleneck. This we think is the classic case of **startup costs** as a factor against parallelism and this is the principle in Parallel Computing that this project explores.

7. Conclusion

We show that parallelizing BLEU score gives considerable benefits that scale up with the data size. Also we see that the nature of the problem is more suited to a shared memory parallelism over distributed memory. This is because of I/O communication over the MPI network proving to be a major bottleneck. One of the possible solutions is to use MPIIO which enables multiple MPI processes to access the data files in parallel. This would greatly speedup the MPI code and would also enable us to use a hybrid approach (MPI+OpenMP) to extract more parallelism from the underlying hardware.

8. Work done by team members:

Both of us have worked together on developing the pseudo-code for the different implementations. Sachith has worked on MPI implementation and Sankalp has written the Serial and OpenMP versions. The writeup was also written by discussing the questions together and collaborating on drive. It is an equal participation of both of us.

References

1. Papineni, Kishore, et al. "BLEU: a method for automatic evaluation of machine translation." Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2002.
2. https://en.wikipedia.org/wiki/BLEU
3. http://opus.lingfil.uu.se/