OpenMP to parallelise LSTM|

Abisek N
CS22B1092

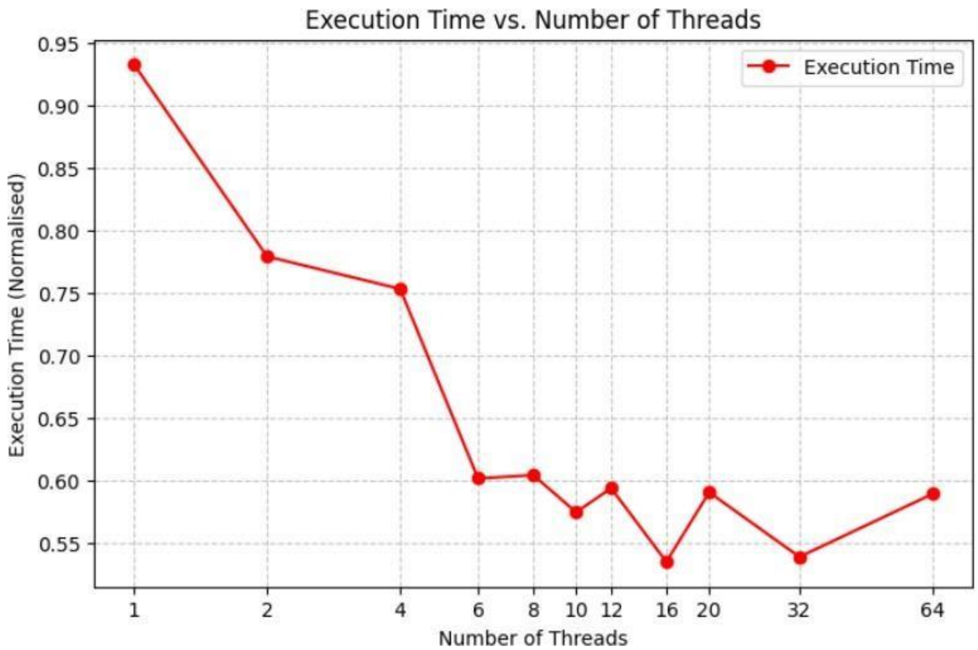
Showing only parallelized part of the code:

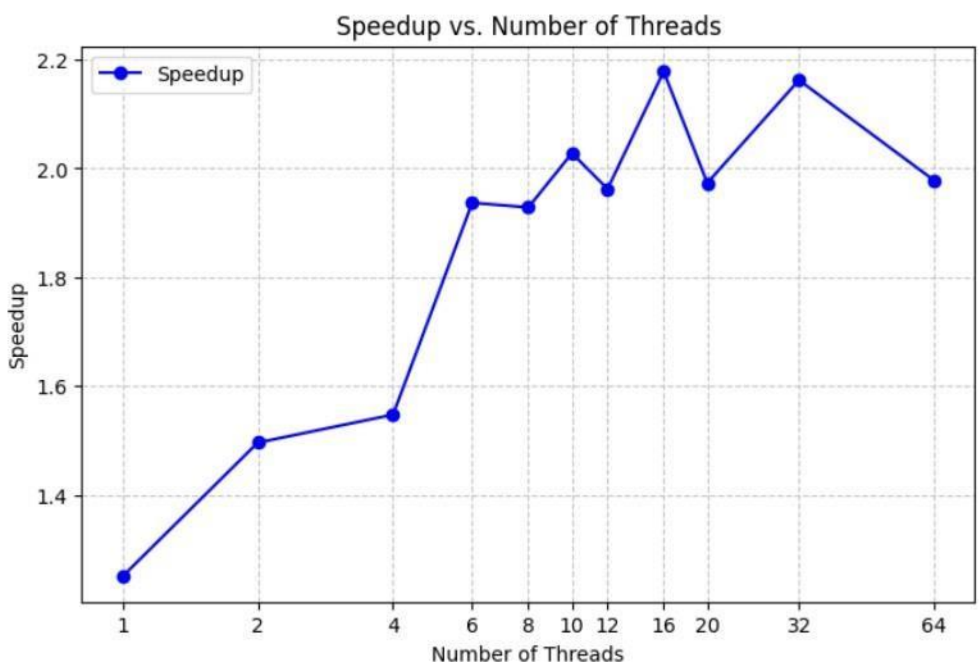
```
root@DESKTOP-8P9HTVN:/mnt/c/Users/LENOVO/Desktop/recurrent-neural-net-master/src# gcc -o main *.c -lm
root@DESKTOP-8P9HTVN:/mnt/c/Users/LENOVO/Desktop/recurrent-neural-net-master/src# ./main input.txt
```

```
void vectors_add(double* A, double* B, int L) {
    #pragma omp parallel for
    for (int l = 0; l < L; l++) {
        A[l] += B[l];
    }
}
```

```
vererYc.
mJfk9rhmya agoagIdmdM
-mtos 2ro
aa frrr fhao ge ind ther wJat
uiwte yo iehinid onliid mitd.
 9onUm aintedT Ceawsydsrsras dd bat vtoree aen eEficlte fhderdtRpste
=====================================================
18:09:43 Iteration: 600 (epoch: 14), Loss: 2.567078, record: 2.567078 (iteration: 600), LR: 0.001000
=====================================================
Iire wairn  band tesire
wnde ,en kuintodh e
vererYc.
 9onUm aintedT Ceawsydsrsras dd bat vtoree aen eEficlte fhderdtRpste
=====================================================
18:09:43 Iteration: 600 (epoch: 14), Loss: 2.567078, record: 2.567078 (iteration: 600), LR: 0.001000
=====================================================
Iire wairn  band tesire
wnde ,en kuintodh e
vererYc.
in bhirgiinty yigmere, ausat anc. o8erthne Irdu in
wnde ,en kuintodh e
vererYc.
in bhirgiinty yigmere, ausat anc. o8erthne Irdu in
in bhirgiinty yigmere, ausat anc. o8erthne Irdu in
r aude aud eand ficihe th.
ino dit iomlgochangI
=====================================================
18:41:26 Iteration: 700 (epoch: 17), Loss: 2.446119, record: 2.442453 (iteration: 695), LR: 0.001000
18:41:26 Iteration: 700 (epoch: 17), Loss: 2.446119, record: 2.442453 (iteration: 695), LR: 0.001000
=====================================================
d0V U?ua8gvett erou4II)gonte aad eamani
mwte inym24
mwte inym24
.oaam
.oaam
ailgas in5 hov.. , top oud. I 4 fthe. veecindas atales soe il irais uth al
peot filbe tetre
ailgas in5 hov.. , top oud. I 4 fthe. veecindas atales soe il irais uth al
peot filbe tetre
wases rakocr,coutg poneoH as .ferehh an atdem
=====================================================
19:13:11 Iteration: 800 (epoch: 19), Loss: 2.347986, record: 2.347986 (iteration: 800), LR: 0.001000
wases rakocr,coutg poneoH as .ferehh an atdem
=====================================================
19:13:11 Iteration: 800 (epoch: 19), Loss: 2.347986, record: 2.347986 (iteration: 800), LR: 0.001000
=====================================================
=====================================================
gin)
Fn1Ho vey lishm wose to hhin site ante ous waW :hhe, he ak the fol
wio thm ir ako Id iatn wfwim te caren sa
 rot hir,tig hine file at he ente s
=====================================================
19:45:00 Iteration: 900 (epoch: 21), Loss: 2.251413, record: 2.251413 (iteration: 900), LR: 0.001000
=====================================================
N?,eygdth Wtoit the wfxlee Yecacdecin.
mI hdes ntsom ead pafi as do tlos ind ha teal tou far in tha me tof lon lpan weitg. the ciuDto lo cands wenedM
=====================================================
```

PLOTS :





Inference on Normalized Execution Times for Parallelizing LSTM using OpenMP

Inference on Normalized Execution Times for Parallelizing LSTM using OpenMP

1.1 Initial Performance Gains (1 to 6 threads)

- Execution time **reduces significantly** from **0.9325 (1 thread) to 0.6017 (6 threads)**.
- This indicates that the **parallelization is effective in this range**, efficiently distributing computations across multiple threads.

1.2 Diminishing Returns (8 to 12 threads)

- Execution time remains almost **constant** between **6 and 12 threads**, with values fluctuating around **0.5747 to 0.5939**.
- This suggests that **parallel efficiency is decreasing**, and increasing threads further does not provide substantial benefits.
- Possible reasons:
    - **Synchronization overhead**
    - **Memory bandwidth bottlenecks**

1.3 Fluctuations & Saturation (16+ threads)

- At **16 threads**, execution time drops to **0.5350**, showing a slight improvement.
- However, beyond 16 threads (**20, 32, and 64 threads**), execution time fluctuates, increasing instead of decreasing.
    - **20 threads: 0.5906** (increase)
    - **32 threads: 0.5390** (small drop)

- **64 threads: 0.5893** (increase again)

- This irregular pattern suggests that **overhead from excessive thread creation and memory contention limits performance gains.**

- At higher thread counts, **the cost of managing parallel execution outweighs the benefits of additional cores.**

Parallelization Fraction

Calculation Using Amdahl's Law:

S=Tseq/Tparallel

f=(1−(1/S))/(1−(1/p))

The parallelization fraction is approximately **57.7%**, for 16 threads.

Interpretation of Parallelization Fraction

- **57.7% of the workload is parallelizable**, while **42.3% remains sequential**.
- This suggests that the LSTM implementation has **significant sequential components** that limit performance scaling.
- **Beyond 16 threads, performance does not improve significantly** due to:

  - **Thread synchronization overhead**
  - **Memory contention**
  - **Load imbalance across threads**

Conclusion

- **LSTM parallelization provides noticeable speedup up to 6-8 threads.**
- **Performance gains diminish beyond 12 threads**, with fluctuations due to synchronization and memory access bottlenecks.
- The **parallelization fraction is 57.7%**, indicating that a significant portion of the workload remains sequential.
- **Further optimizations are required** to improve parallel efficiency, particularly in memory access and thread management.