HPC ASSIGNMENT-7 REPORT

CS22B2014 Navaneethakrishnan R

The parallel code for Graph Neural Network (GNN) computation with parallelism optimization is provided in main.c. The plotting script is provided in analysis.py. After execution, the results are stored in a txt file, which will be used by Python to plot. The setup includes 10 nodes, each with 1 to 10 neighbors, and 200 features per node.

The input data was generated using Python, and the thread counts used for execution are {1, 2, 4, 6, 8, 10, 12, 16, 20, 32, and 64}.

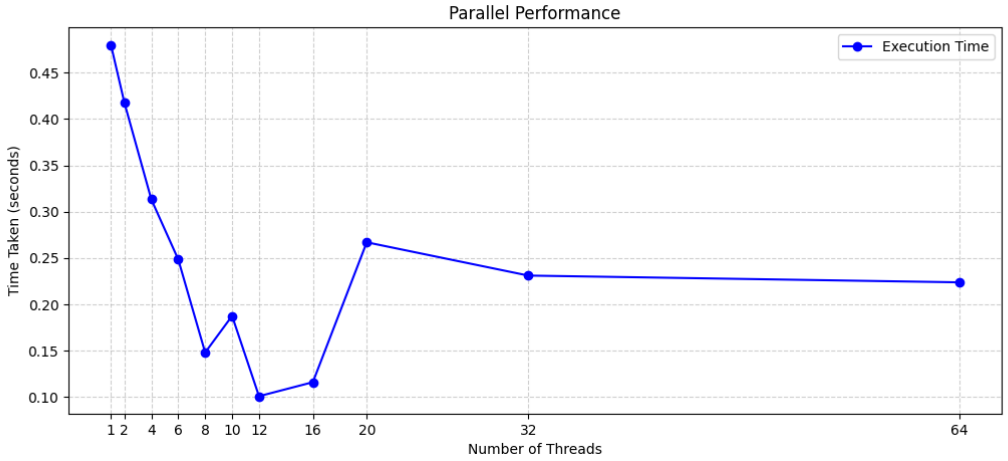
Code segment for Parallel Execution

```
#pragma omp parallel for
for (int i = 0; i < NUM_NODES; i++) {
    float* original_features = (float*)malloc(NUM_FEATURES * sizeof(float));
    memcpy(original_features, node_features[i], NUM_FEATURES * sizeof(float));

    if (graph[i].num_neighbors > 0) {
        float* scores = (float*)malloc(graph[i].num_neighbors * sizeof(float));
        float* aggregated = (float*)calloc(NUM_FEATURES, sizeof(float));

        // Calculate attention scores
        for (int j = 0; j < graph[i].num_neighbors; j++) {
            int neighbor = graph[i].neighbors[j];
            scores[j] = attention_score(original_features, node_features[neighbor], params);
        }

        // Apply softmax
        softmax(scores, graph[i].num_neighbors);
```

Here I am parallelizing with respect to the number of available nodes for this example I have taken 10 nodes as example to demonstrate parallelism.

Analysis of Thread vs Time Taken

Processor used has 2 cores and 4 Logical Processors



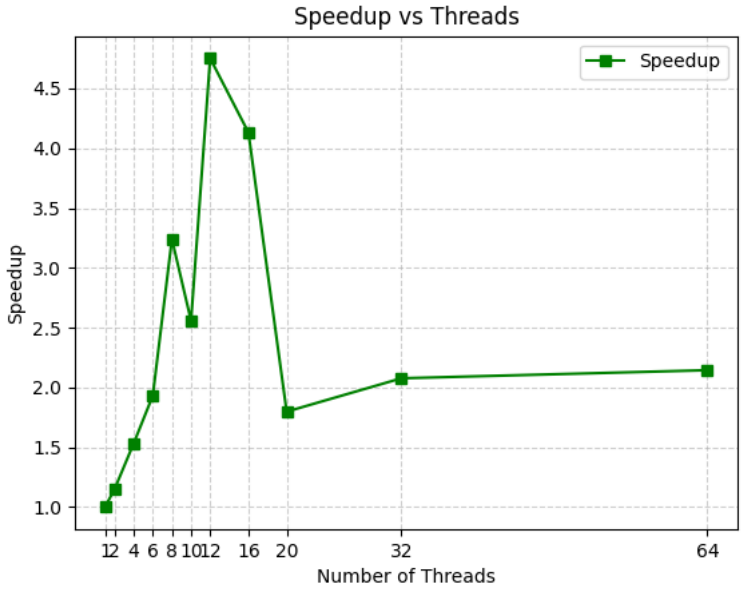
The parallelization factor improves significantly as the number of threads increases up to 10-12, reaching a peak value close to 0.85.

After reaching the peak, the parallelization factor declines sharply beyond 12 threads, indicating inefficiencies likely caused by synchronization overhead or workload imbalance.

At 32 and 64 threads, the parallelization factor remains constant around 0.5, suggesting diminishing returns due to excessive thread contention.

Speedup vs Processor

Here for the calculation of speed up every time is taken relative to the sequential time T1. Speed of Thread t = Time Taken at T=1 / Time Taken at T=t



The speedup follows a trend similar to the parallelization factor, reaching a maximum of approximately 4.5× at 10-12 threads.

After 12 threads, the speedup decreases significantly, suggesting overheads such as communication costs, context switching.

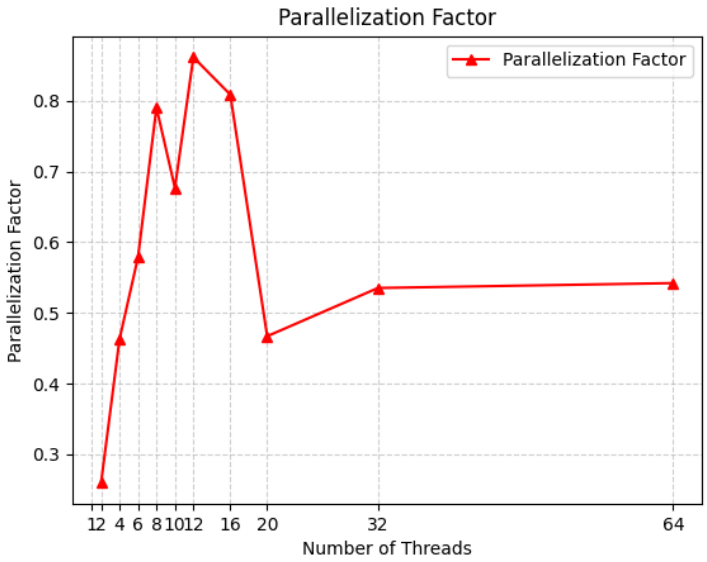
The speedup stabilizes around 2× for 32 and 64 threads, indicating that adding more threads does not significantly improve performance.

Parallelization Fraction and Inference

Here parallelization Fraction is calculated using Amdahl's law.

P = ((1/speedup) – 1) / ((1 / number of threads) – 1)

1 > P > 0 – suggestion that the code is good for parallelization whereas P<0 suggests that sequential code runs more optimized compared to parallel code.



The parallelization factor improves significantly as the number of threads increases up to 10-12, reaching a peak value close to 0.85.

After reaching the peak, the parallelization factor declines sharply beyond 12 threads, indicating inefficiencies likely caused by synchronization overhead or workload imbalance.

At 32 and 64 threads, the parallelization factor remains constant around 0.5, suggesting diminishing returns due to excessive thread contention.

Outputs:

Thread	Time	Speedup	Parallelization Factor
1	0.480128	1.0	
2	0.417736	1.1493574889403835	0.2598973607038124
4	0.313361	1.5321881153047126	0.4631181684884031
6	0.248602	1.9313118961231206	0.578660690482538
8	0.148166	3.2404735229404857	0.7901750009521271
10	0.187674	2.558308556326396	0.6767963728191
12	0.101029	4.752378030070574	0.8613589406461307
16	0.116171	4.132941956254142	0.8085777126099707
20	0.267069	1.7977676181061824	0.46711008643309154
32	0.231176	2.0768937952036546	0.5352379150506102
64	0.223887	2.144510400335884	0.5421644175309864

Conclusion:

The performance scales well up to 10-12 threads, after which the benefits diminish due to thread overheads.

Beyond 12 threads, excessive threading introduces inefficiencies, leading to reduced speedup and a declining parallelization factor.

The ideal number of threads for maximum efficiency appears to be around 10-12, balancing execution time and speedup while minimizing overheads.