Parallel and Distributed Computing on Nerves

Instructores:
Juan Jose Cordoba,
Laura Isabella Martinez Galindo,
Juan Andrés Mendez Galvis

Abstract

The objective of this assigment is to developt a couple of parallel programs to run on the raspberry pi cluster that we created with the help of nerves.
The first program is a simple parallel program that calculates the number of repated words in an array of random words. Where as the second program is going to rotate an image using the parallel processing power of the raspberry pi cluster.

1 Task 1

1.1 Analysis of the parallelism possibilities

The potential for parallelism in this program is substantial, given its simplicity of counting repeated words in an array of random words.

We considered the following parallelism strategies:

- **Fork-Join:** Divides the array into smaller subarrays, counts repeated words in each subarray in parallel, and then joins the results to obtain the final count.
- **Map-Reduce:** Maps the array into smaller subarrays, processes each subarray in parallel, and then reduces the intermediate results to produce the final count.
- **Master-Worker:** Splits the array into smaller subarrays, assigns each subarray to a worker process to count repeated words, and aggregates the results at the master process.

We ultimately chose the fork-join model for its simplicity and efficiency. Our implementation involves counting repeated words in individual segments of the input array and aggregating the results from all segments. Additionally, we introduced a parallel component that spawns tasks, each handling approximately 1000 words, optimizing parallelization according to best practices.

1.2 Speedup/Slowdown Analysis

With this approach, we achieved a speedup of 5 times compared to the sequential version of the program. Here are the results of the program running 10^6 words on the raspberry pi cluster:

Table 1: Count Repeated Words on Raspberry Pi Cluster

Nodes	**Time (ms)**
1	3008
4	1812

The results in 1 show that the program scales well with the number of nodes, achieving a speedup of 1.66 times with 4 nodes compared to 1 node.

We also tested the program on a workstation here are the specs of the workstation:

Table 2: Workstation Specks

Processor	Amd Ryzen 9 7950X3D
Cores	16
Threads	32
RAM	64 GB

Here are the results of the program running 10^6 words on the workstation:

Table 3: Count Repeated Words on Workstation

Nodes	**Time (ms)**
1	210
4	126

As we can see in 3 the program scales well with the number of nodes, achieving a speedup of 1.66 times with 4 nodes compared to 1 node. The other thing that we can see is tha the speed of the workstation is much faster than the raspberry pi cluster. The speed can also be aluded to the conection that the raspberry pi cluster has, which is wifi 2.4 GHz. Instead of the workstation that the nodes are running on the same machine.

1.3 Conclusion

The fork-join model is an effective parallelism strategy for counting repeated words in an array of random words. Our implementation demonstrates that the program scales well with the number of nodes, achieving a speedup of 1.66 times with 4 nodes compared to 1 node. The program also runs faster on a workstation compared to the raspberry pi cluster, which can be attributed to the workstation's superior processing power and network connection. One of the strategies that we can use to improve the performance on the raspberry pi cluster is to use a wired connection instead of a wireless connection.

2 Task 2

2.1 Analysis of the parallelism possibilities

We utilized the fork-join model for its simplicity and efficiency. By converting the image into an RGB matrix, we can split it into smaller submatrices and rotate each in parallel. After rotating, we join the submatrices to form the final image, an addition operation that can also be parallelized.

2.2 Speedup/Slowdown Analysis

On a single core a 360x360 image took 18 seconds to rotate, while on the best strategy that we used took 4 seconds to rotate the same image. Here are the results of the program running on the raspberry pi cluster:

The first approach was to do the same that we did on the first task, where we split the image into smaller submatrices until the size of the submatrix was near 1000. This where the results:

Table 4: Rotate Image on Workstation multiple task approach

Nodes	**Time (ms)**
1	43846
4	11882

This approach was a slowdown compared to the sequential version of the program. The reason for this is that the overhead of splitting the image into smaller submatrices was too high. In this case if we take into consideration that the smallest submatrix that we have is going to have just 3 rows that means that we are going to create 100+ task in a cpu that on has 16 cores and maybe 12 aviable because the Os is going to use some of them. This is going to create a lot of overhead that is going to slow down the program. However we managed to find a speed/up when we distributed this task into 4 nodes but it was minimal. This program was son ineficent that the raspberry cluster wasn't able to run it.

The second approach was to set a number of tasks that we are going to create. With this approach we managed to find a better speedup. Here are the results of the program running on both the raspberry pi cluster and the workstation:

Table 5: Rotate Image - 2 Tasks per Node Approach

Table 6: Raspberry Pi Cluster

Nodes	**Time (ms)**
1	160320
4	67200

Table 7: Workstation

Nodes	**Time (ms)**
1	10743
4	4603

As we can see in 5 the program scales well with the number of nodes, achieving a speedup of 2.38 times with 4 nodes compared to 1 node on the raspberry pi cluster. The program also scales well with the number of nodes on the workstation, achieving a speedup of 2.33 times with 4 nodes compared to 1 node. Eventhough we managed to get a good speedup for the program the implemenatation of the rotate function might need some refinment because on the raspberry pi is running extremly slow. The reason for this is that the raspberry pi has a much slower cpu and also is a quad core instead of a 16 core like the workstation. This is why the program is running slower on the raspberry pi cluster.

We did try to incremental increase the number of tasks that we are going to create but the improvement was minimal being 2 task per node the sweet spot for this program.

2.3 Conclusion

The fork-join model is an effective parallelism strategy for rotating an image using the raspberry pi cluster. Our implementation demonstrates that the program scales well with the number of nodes, achieving a speedup of 2.38 times with 4 nodes compared to 1 node. The program also scales well with the number of nodes on the workstation, achieving a speedup of 2.33 times with 4 nodes compared to 1 node. The program is running slower on the raspberry pi cluster compared to the workstation, which can be attributed to the raspberry pi's slower processing power and network connection. One of the strategies that we can use to improve the performance on the raspberry pi cluster is to use a wired connection instead of a wireless connection.