CMPE300 PROJECT 2

MPI PROJECT

CEYHUN SONYÜREK - 2020400258

AHMET ERTUĞRUL HACIOĞLU - 2020400132

Introduction

Parallel programming is a process of splitting computations of an algorit hm into smaller parts in order to handle them synchronously by taking the advantage of hardware resources as far as possible. It enables faster and more efficient processing of tasks which can be divided into smaller parts that are able to operate simultaneously. Parallel programming techniques offer flexibility in managing complex simulations. In the code, the ability to handle different machine operations concurrently demonstrates the adaptability of parallelism. It allows for dynamic adjustments to the production cycle based on varying conditions, showcasing the flexibility inherent in parallel programming paradigms. In this project, we benefit from these properties of parallel programming using MPI, which is an application program interface defining a model of parallel computing where each parallel process has its own local memory. Passing messages to share data between processes is also a beneficial feature of it that we used in the project.

In short, in this project, we aimed to replicate a factory environment, utilizing parallelism to boost overall productivity. Using the mpi4py library, we established a parallel structure among machines to execute specific string operations based on the provided inputs.

Design Choices

Parallel Programming Selection: MPI (Message Passing Interface) was chosen due to its ability to facilitate communication among different nodes efficiently. It allows for distributed processing, enabling the simulation of numerous machines simultaneously, mirroring a real production setting.

Data Structure: The simulation employs a dictionary-based data structure to represent machines and their attributes. Each machine is characterized by its ID, parent, operation, children, factors, and product, allowing easy traversal and manipulation during the simulation.

Algorithmic Approach: The algorithm operates in cycles, which are production cycles. Each machine executes its operation based on the defined factors and performs communication with parent and child machines accordingly. Parallelism is incorporated by handling multiple machines' operations simultaneously.

Execution of the Code

The main part of the code is “control_room.py” and created child proccesses are running in “worker.py”. Firstly “control_room.py” takes arguments for the files will be read and written. After master process (control_room.py) read file and initialized all values, It spawns worker processes with the following code:

```
comm = MPI.COMM_SELF.Spawn(command="python3", args=["worker.py"],
maxprocs=machine_num)
```

Then, in a for loop iterating over the number of machines, it sends all the input values in a list (including strings, integers and a dictionary) to the spawned slave proccesses according to their id (id – 1 = rank of the slave process) by send() (blocking communacion).

For each slave process spawned, “worker.py” receives the message according to their id via comm. Then, they stpre the values in the list in the variables. After that, they perform their operations according to their id (being odd or even, and also terminal process’ rank is 0) and do log operations if necessary with isend() (non-blocking) communaciton function. If a process has a parent, it sends the product to the parent after the operation with send() function via node_comm. In terminal process, final product is sended to master process.

Finally, master process does necessary print operations after it receives final products and wear out logs.

Mock Input

İnput:

9

31

3 1 4 1 5

13

2 1 chop

3 2 reverse

4 2 split

5 2 trim

6 3 split

7 4 trim

8 4 chop

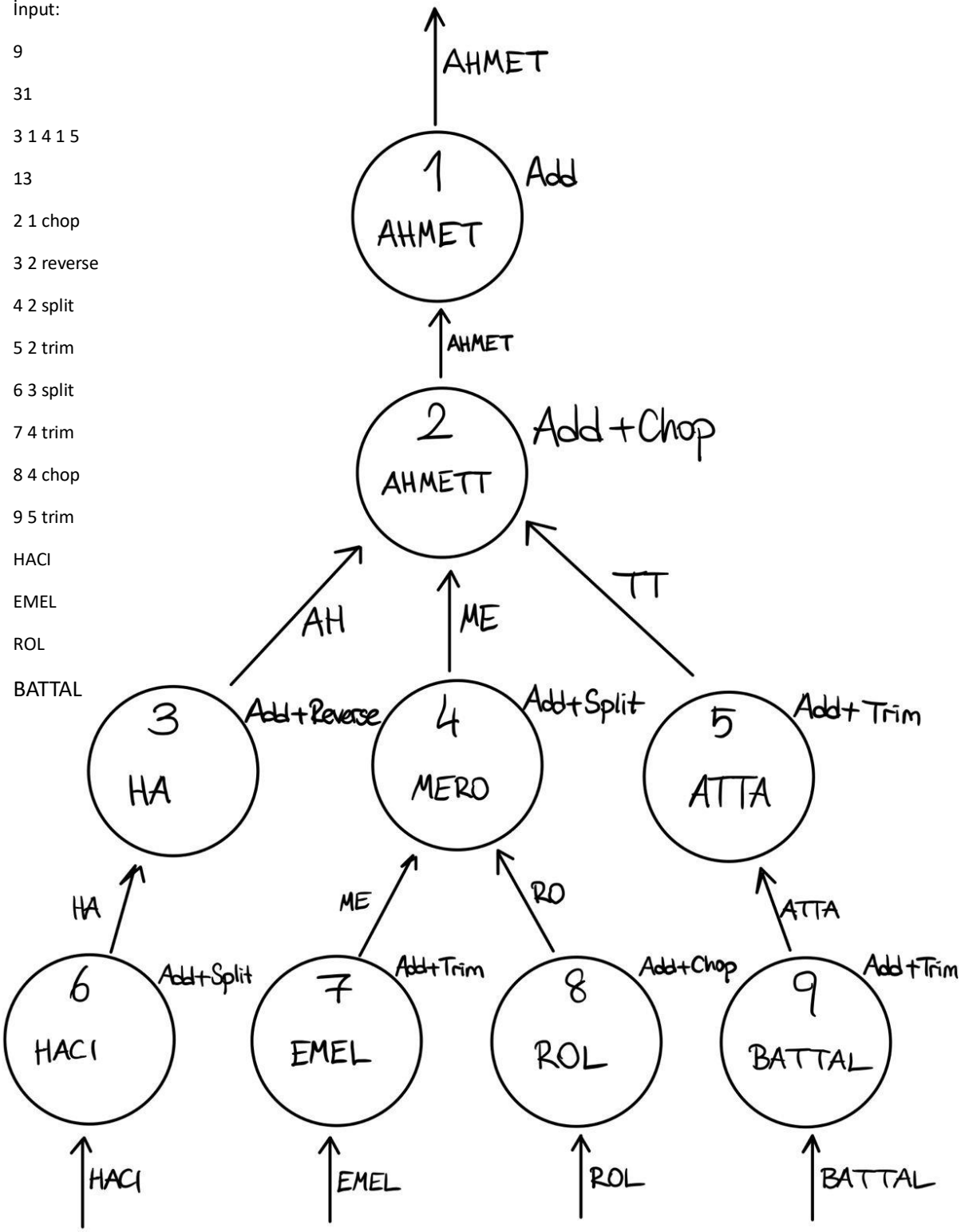
9 5 trim

HACI

EMEL

ROL

BATTAL



Bonus Part

Implementing an Industry 4.0 digital twin, like creating a mirror image of a factory, involves challenges. Integrating data from various sources, akin to piecing together a puzzle, requires accuracy and timely delivery. Any delays or errors in data can disrupt operations, impacting efficiency.Accurately simulating a complex factory environment within the digital twin demands a thorough understanding of machine behavior. Predicting how machines will respond in different scenarios is challenging, especially when unexpected issues arise.Ensuring the digital twin's security against cyber threats is critical. Security concerns might prompt temporary pauses in operations to resolve potential risks and protect sensitive data.Adapting the digital twin to changes within the factory landscape requires adjustments. These modifications might briefly interrupt its operations during the recalibration process.Getting the workforce accustomed to the digital twin might take time. Staff training and adoption might temporarily slow down processes until everyone becomes familiar with the system.Demonstrating the tangible benefits and cost-effectiveness of the digital twin is essential for gaining acceptance. Unclear benefits might lead to cautious adoption, impacting the pace of implementation.Adhering to regulations and adapting to new compliance standards might briefly pause operations for necessary adjustments within the digital twin.In reality, challenges such as delays due to data waiting times, pauses for resolving unexpected issues, or halts for system updates can occur. Successfully addressing these challenges is crucial for the digital twin to become an invaluable tool in enhancing factory operations.