# CMPE 300 – Project 2 Report

Ahmet Fırat Gamsız
Ali Tarık Şahin
Group No: 40

## 1. Design

We implemented two scripts for this project. First script is called master.py. Master corresponds to control room of the factory. Also, master manages general program tasks, it reads, parses, and saves the input.

Master uses COMM_WORLD to initialize an intra-communicator and using this we spawn new processes. Specifically, we spawn the second script, called slave.py. This script corresponds to machines in the factory, so we are initializing our machines and connect them to communicator. Then master sends the input to relevant machine (nodes spawned will be mentioned as machine instead of slave throughout the report). Also, master sends the info of parent and children machine ids to machine and forms a tree structure between the machines. Master's final task is receiving logs and final products. Master uses iprobe to check the logs in non-blocking manner. Logs are always sent with tag 0 to discriminate them from final products. Master does this check all the time while program working. Same is true for the final output. Master uses iprobe to check whether terminal completed a production cycle and sent its product. If so, master receives it and prints it to the output file. However, logs are first buffered and printed to output after all production cycles are completed.

Initialized machines do the actual task in this project. They have two communicators. First one is an inter-communicator that they get from their parents. This communicator only communicates with the master. Second one is an intra-communicator, which is COMM_WORLD. This communicator enables machines to communicate with each other. We limit this communication as from child to parent. After initializing the communicators machine receives its initial data. There are three different machine types: First is terminal machine which is root of the tree. Second one is machine with even id and third one is with odd id. Second and third have same code structure and only difference is that they have alter between different operations.

Terminal machine doesn't alter between different operations. It only performs add operation. Since add operation doesn't have wear cost, terminal machine never needs maintenance. This machine waits for its children' messages in a blocking manner using receive through intracom. All the children hold in sorted array so by placing receive in a for loop we receive the messages in correct order. After that the add operation is performed and the product is sent to master with tag 1 through intercom. This completes a production cycle, and the machine turns back to the start of the loop.

Other machines alter their operation after each cycle. If the machine is not a leaf node, then it waits for its children the same way terminal does and perform add operation. If the machine is a leaf, it performs add on its given input. Then the machine performs its current operation on the

result of the add operation. These operations are simple string operations and doesn't require further explanation. However, these operations cause wearing out. Each function implementing the operation increases the accumulated wear by its wear factor. After operation is completed, the product is sent to the parent using the intracom with tag of its machine id. Tagging is done to prevent mixing up the message retrieval between machines. Then the machine checks its maintenance. If the accumulated wear is over the threshold, by using the given formula in the description machine calculates its cost of maintenance. This is logged directly to the master using intercom and accumulated wear is set to 0. Machine changes the current operation to next operation and completes its production cycle. The loop returns to start. Machines loop for exactly the given production cycle amount.

After all production cycles are completed and logs are written to the file communicators get disconnected and program terminates itself. Sometimes after completing its execution, program (probably MPI since python doesn't often does this) gives segmentation error. We are not sure this is a bug of our program or an OS related issue, however; this does not constitute any disruption in the execution of the program. Program correctly executes, prints its output, and finalizes all its instructions. This error is produced when program is terminating so it can be ignored.

## 2. Advantages of Parallel Programming

This project includes multiple machines working synchronously and the sibling processes doesn't depend on each other to complete their tasks. Using parallel programming for this task is the most natural way. While sibling processes executes synchronously, their parents wait for their output. Thanks to message passing interface each child sends its output to correct parent without mixing up the communication or halting other parents. Also, while child is processing in the second production cycle, parent can process in the first cycle. So by using parallel programming we reduced the waiting overhead that we would normally have in sequential implementations.

## 3. Mock Input

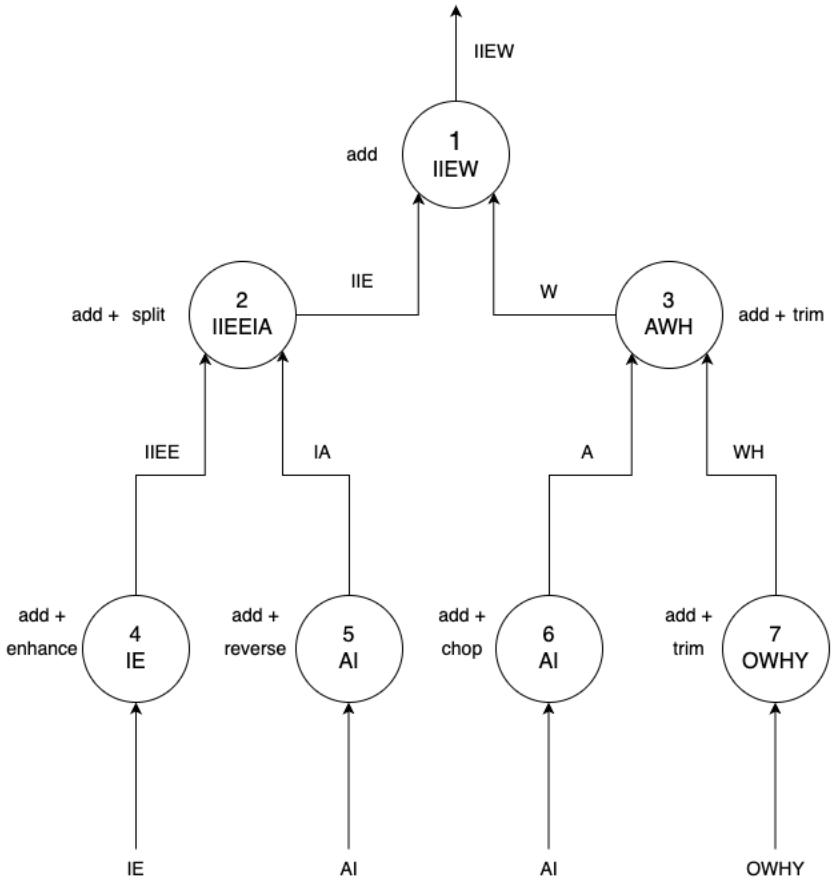
### Input

```
7
5
4 3 1 2 3
13
2 1 split
3 1 trim
4 2 enhance
5 2 reverse
6 3 chop
7 3 trim
IE
AI
```

AI
OWHY

Output
IIEW
IAOWHYIIAA
IIIAAW
IIEOWHYA
IIAIIW
6-4-5
5-3-5
4-9-5

First Cycle Graph




## 4. Industry 4.0

Any simulation of real-world systems in the digital world would have many flaws regarding the system itself. In this Industry 4.0 digital twin implementation, there are also many points overlooked. For instance, this simulation offers us perfectly synchronized flawless machines, which is far from the reality. Not only machines but also products will have differences in reality.

Unlike the uniform and standardized products in the MPI Project, consisting of ‘reliable’ strings, there will be several diverse product types and they will require different algorithms rather than one algorithm for all products. That’s why the real digital twin implementation would require additional data about each type of products and machines. This would also lead to complex and sophisticated algorithms and yield non-standardized data, which makes maintenance of the system challenging. On the other hand, communication between machines may also yield different results in the real system. More specifically, some properties like time taken and integrity of the product may suffer from external factors. Finally, the digital twin must be adaptable to different scales of the factory growth. It must yield accurate results with these additional layers. Therefore, there are many flaws to remedy in the simulation of real-world factory design.