NTNU

Kunnskap for en bedre verden

Assignment 3: Wafer Production Line

TPK4186 - Advanced Tools for Performance Engineering

Done by

Group 10

Georg Hove Zimmer & Fredrik Faanes Slagsvold

Table of content

Introduction

This documentation provides a detailed account of our thought process in tackling this assignment. We have developed six classes, namely Batch.py, Buffer.py, Task.py, Unit.py, ProductionLine.py, Event.py, Scheduler.py, Printer.py, and Simulator.py.

The code is organized into several classes that represent different components of a manufacturing system. The Batch class represents a group of wafers, while the Buffer class manages the movement of batches between tasks. The Task class processes batches of wafers, and the Unit class represents a processing unit that can perform specific tasks. The ProductionLine class brings all these components together to simulate a production line. The Scheduler class manages the scheduling of events related to units, tasks, and batches. The

Printer class prints the scheduled events to an output file, and the Simulator class provides a high-level interface for simulating a production line, managing the different components involved in the simulation, and producing simulation results.

In the code, you can look through comments to run different aspects of the task.

Task 2.1 Production Line

Task 1 & 2

The production line structure is made up of the Batch, Buffer, Task, Unit and Production Line classes. We can run the following code in ProductionLine.py:

```
>>> pl = ProductionLine()
>>> batch_size = random.randint(20,50)
>>> first_input_buffer = pl.buffers[0]

>>> for i in range(3):
>>>    batch = Batch.Batch(i+1,batch_size)
>>>    first_buffer.load(batch)

>>> pl.print_productline()
```

```
Task 1:
  Input buffer: [Batch 1: 33, Batch 2: 33, Batch 3: 33]
  Output buffer: []
Task 2:
  Input buffer: []
  Output buffer: []
Task 3:
  Input buffer: []
  Output buffer: []
Task 4:
  Input buffer: []
  Output buffer: []
Task 5:
  Input buffer: []
  Output buffer: []
Task 6:
  Input buffer: []
  Output buffer: []
Task 7:
  Input buffer: []
  Output buffer: []
Task 8:
  Input buffer: []
  Output buffer: []
Task 9:
  Input buffer: []
  Output buffer: []
 Buffer 1:
   No from task
   To task: Task 1
 Buffer 2:
   From task: Task 1
   To task: Task 2
 Buffer 3:
   From task: Task 2
   To task: Task 3
 Buffer 4:
   From task: Task 3
   To task: Task 4
 Buffer 5:
   From task: Task 4
   To task: Task 5
 Buffer 6:
   From task: Task 5
   To task: Task 6
 Buffer 7:
   From task: Task 6
   To task: Task 7
 Buffer 8:
   From task: Task 7
   To task: Task 8
 Buffer 9:
   From task: Task 8
   To task: Task 9
 Buffer 10:
   From task: Task 9
   No to task
 Unit 1:
   Task 1
   Task 3
   Task 6
   Task 9
 Unit 2:
   Task 2
   Task 5
   Task 7
 Unit 3:
   Task 4
   Task 8
```

Figure 1: Production line

In order to visualize the production line in the terminal with three batches in the first buffer, refer to Figure 1, for the output. Each Task is connected to an input buffer and an output buffer, with the output buffer being the same object as the input buffer of the next task. This method is one of the ways the system is interconnected. This connection is also evident when we print each buffer along with its associated tasks, where the "to task" of a buffer matches the "from task" of the next buffer. Additionally, at the bottom of the output, we can see the tasks assigned to each unit.

Upon loading and unloading the buffers with batches, it becomes evident that the batches transition from one list to

another throughout the system. All the necessary functions to facilitate this process can be located within the Buffer.py, Task.py, and Units.py files.

Task 2.2 Simulation

Task 3

For the simulation part we thought that a discrete event simulation would be the best approach to solve the problem on hand. Our main idea is that one event will trigger the next event and the simulation loop will run as long as there are events in the scheduler. Therefore the first event has to be created manually.

The simulation starts by initializing the production line and scheduler, and creating a list of batches based on the total number of wafers to be processed. During the simulation loop, the scheduler iterates through events, performing actions depending on the event type. These actions include loading a batch to a unit, processing a batch, and loading a batch to the output buffer. Once a unit has completed processing a batch, the simulation attempts to load another task onto the unit.

Task 4

One batch

See the tsv file named *“one batch”* in the program.

A few batches

See the tsv file named *“two batches”* in the program.

See the tsv file named *“three batches”* in the program.

1000 wafers

See the tsv file named *“20 batches with 50 before optimization.tsv”* and all the files starting with “Shortest….” and “Longest…”

Task 2.3 Optimization

Task 5

Loading time

We found it reasonable that the most efficient way the system would run is if the batches get sent into a unit as soon as another unit has been removed. Therefore we decided that the system was already optimized on this front.

Task 6

Heuristic

To optimize the heuristics, we explored various alternatives, including Longest Processing Time, Shortest Processing Time, and First Come, First Served. We opted to implement the Shortest Processing Time heuristic for task prioritization within each unit, as it ranks tasks according to their processing duration. This heuristic aims to minimize both the total processing time and average job completion time, aligning with our goal of minimizing completion time. As a result, we embraced the Shortest Processing Time heuristic as our strategy for achieving this objective.
The modification in the code occurred within the get_next_task() function of the Unit class. Initially, the function simply iterated through all tasks in a unit, checking if the associated buffers had capacity and whether a batch was available in the input buffer. It then returned the first task meeting these criteria, resulting in an approximately chronological order. With the new implementation, we compile a list of all available tasks, compare their completion times, and return the most favorable option. This adjustment produced a significant improvement in our results.
To provide an experimental comparison, we also tested the Longest Processing Time heuristic, which operates similarly but prioritizes tasks with the longest completion time. The table below displays the results for all three heuristic variants.

Heuristic method	**Completion time 1000 wafers (50 batch size)**
First Come, First Served	6244 time units
Longest Processing Time	6187
Shortest Processing Time	5742

Task 7

Batch size

When we did this task we decided to look away from the fact that each batch must be between 20 and 50 wafers. We considered making it so that batch_sizes that did not match exactly 1000 wafers in total had to create more batches, but considering the fact that we should optimize based on the notion that exactly 1000 wafers was produced we decided to go with the former option. We created a function create_batches_from_thousand_wafers() that adds the remainder between 1000 and the total amount of wafers. This remainder is added to the final batch.
To find the optimal batch size we created two functions. The first one is try_different_batch_sizes() which runs simulations with 20 to 50 batches and adds the batch sizes and times into two lists. The second one is find_optimal_batch_size() which creates a dictionary from the two lists returned in the try_different_batch_sizes()-function and finds the key corresponding to the lowest value. Before we optimized the production we found the ideal batch size to be 41. After the optimization (shortest processing time) we found the ideal batch size to be 37. See the “Batch sizes vs. Time used.png” to view the different batch sizes and their corresponding times.

In the file *“Shortest Processing Time (37 batches).tsv”* you can find our best completion time. This is heuristic optimization combined with batch size optimization.

The time was 5663.7 time units.