TPK4186 - Advanced Tools for Performance Engineering Spring 2023

Assignment 3: Wafer production Line

Done by Christian G. Kartveit & Skjalg Nysaeter

NTNU
Kunnskap for en bedre verden

Introduction

To ensure you can run this program, you need to use the python virtual environment. This is done by using a terminal in the delivered folder.

To create a python virtual environment, use the following command:

```
Example: python -m venv /path/to/new/virtual/environment
```

```
Python -m venv venv
```

To activate use:

- Windows `source venv/Scripts/activate`
- Mac/linux `source venv/bin/activate`

Then run `pip install -r requirements.txt`

This will install the required packages to run the application

The Dockerfile provided may be ignored, as it was used to create a Docker Container to run the simulation on a faster computer remotely.

Assumptions

- You need to run all the scripts in the requirements file to run the application
- The time specified between each task is estimated to be minutes, and therefore total runtime is set to minutes.
- For task 7 we simulated the same batch size for all simulations to be able to get consistent results.
- The total runtime is specified in the Optimization.docx document at the bottom. This is a file created from the optimization() method. Remember that the total runtime for the program may be different on a weaker computer.

Structure

Classes

- **Batch:** Contains an ID and a size.
- **Buffer:** Contains an ID, a capacity and a list of its current batches.
- **Task:** Contains an ID, a processtime, a loadbuffer, an unloadbuffer, and a currently processing batch.
- **Unit:** Contains an ID, a list of its assigned tasks, and a currently processing task.
- **Event:** Contains an execution time, an event action, and an assigned unit.
- **ProductionLine:** Contains the buffers, tasks, and units.
- **Printer:** Contains an outputlocation.
- **Simulation:** Contains an eventqueue, a list of batches, and a current time. It also contains variables like intervals to manage the optimization.

Batch, Buffer, Task, and Unit:
Object oriented. Representing the components of the simulation.

ProductionLine:
Creating the components of the simulation except for the batches. We chose to do this manually because it makes it easier to change specific parts of the simulation.

Event:
The event object is what drives the simulation. Every time an event is executed a new eventobject is created and sorted into the eventqueue based on its completion time.

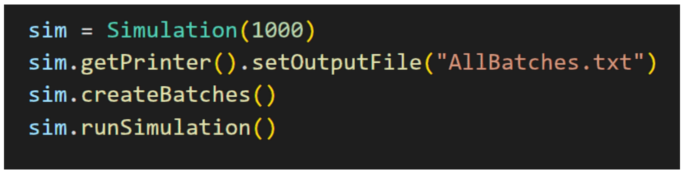
Printer:
The printer class consists of functions that print the current time, object IDs, event actions and everything else that we might use to get a better understanding of how the simulation is running. The functions are called at the preferred times in the simulation and printed to the preferred destination.

Simulation:
Responsible for the simulation.

Functionality

Start Simulation

The simulation is run by creating a new simulation with the desired wafers to process, setting the desired outputlocation with setOutputLocation(), creating the batches with createBatches(), and finally runSimulation() to start the simulation..



```
sim = Simulation(1000)
sim.getPrinter().setOutputFile("AllBatches.txt")
sim.createBatches()
sim.runSimulation()
```

Task 1: Handling the components of the simulation is done in the designated classes. Creating the components and assigning buffers and tasks are done in the productionLine.

Task 2: The printer makes simulation observation easy. It has all the necessary functions to keep track of batches, events and runtime. I also have basic print functions to check on the different components of the production line.

Task 3: The actions of the simulations are handled as “Events”. Event objects are made as soon as the batches are created. These event objects are placed in an eventqueue and sorted based on the events completion time. Each event is assigned a unit and an action. The action can be one of three: “loadBatchToSimulation”, “load”, or “unload”.
The simulation runs as long as there are events left in the eventqueue and there are batches left in the simulation.

Task 4: The task is completed running the Task_4() method. This runs three different simulations with 20, 150 and 1000 batches. The results are printed to the specified files using the printer. From the textfiles you can see that all the batches are run through the simulation properly and the end times for the simulation.

Task 5: The task is completed running the Task_5() method. This runs the methods worstCase() and reducingTimeBetweenBatches(). WorstCase() calculates the worst total time of processing 1000 batches, while reducingTimeBetweenBatches() runs a number of simulations with different time intervals between each load of the simulations. This makes us able to collect the result data and visualize it in the form of grapes which you can see in the experiment part. Results are found in the Optimization.docx file

Task 6: The task is executed på running the Task_6() method which calls upon the changeOrderingHeuristicAndLoadingtimes() method. This method makes simulations with batch size 20 for each of the 288 permutations. We do this for the loading times between 0 and 300. Results are found in the Optimization.docx file.

Task 7: Executed by Task7(). A loop that runs Task_6() for each of the batch sizes available between 20 and 50. Only the best results from each loop are saved, and presented in a plot. Results are shown in the Optimization.docx file.

Optimization.docx:
Contain all the results from task 5 to 7. This may produce different results for each run of ProductionLine.py, since some calculations are done with random batch sizes.
The method optimization() creates the document, plots and results from the optimization tasks!

Experiments

All our experiments are documented in the Optimization document.