Computing optimal performance in a clustertool

A.T. Hofkamp

$Id: desc.tex 8 2009-01-20 15:01:26Z hat $
compiled January 20, 2009

1 Introduction

In the semi-conductor industry, performance of the machines is of critical importance. One area of attention is to optimize transport of wafers between the machines (*processing cells*). Recently, a new modular approach has been introduced where several processing cells are located closely in a so-called cluster tool. As an example of such a tool (inspired by [1, 4]), see Figure 1. The processing cells P1…P9 are all located around the wafer handling system formed by the robots R1…R4.

Each robot has an arm that can pick and place a single wafer from its left, right, up, or down neighbor. Between two robots is a buffer that can contain a single wafer. Each processing cell P_i takes a wafer, processes it, and returns the processed wafer to the robot. At the left, a wafer supply system WS delivers unprocessed wafers to the cluster tool, and takes processed wafers. With arrows, the flow of the wafers is shown in the cluster tool.

While in real semi-conductor manufacturing systems cluster tools as shown in Figure 1 are normal, the assignment is about computing performance in a smaller[1] cluster tool, namely with a wafer supply, one robot, and three processing cells. The system is shown in Figure 2. It consists of a single robot R1, and three processing cells. Each wafer from the wafer supply WS must first visit processing cell P1, then cell P3, and finally cell P2. Afterwards, the wafer is returned to the wafer supply where it is stored until an operator removes the processed wafers from the cluster tool.

[1]The general approach of computing performance is the same in both cases, for real-world cluster tools the amount of data to examine is just a **lot** bigger.

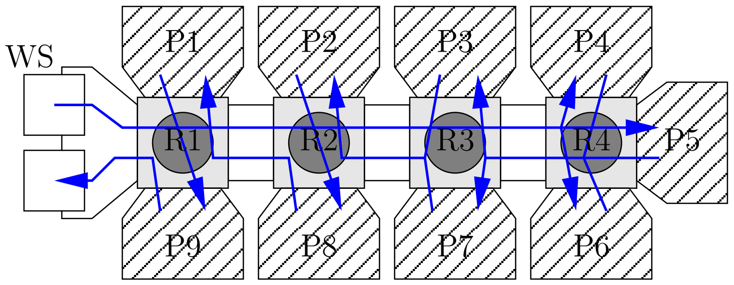


Figure 1: Example cluster tool with four robots and nine processing cells.

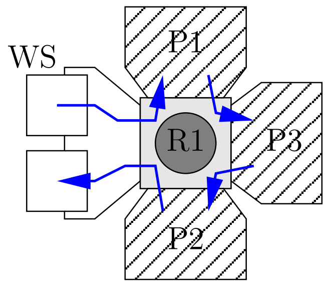


Figure 2: Cluster tool as used in the assignment.

2 Tool operations

The goal of the assignment is to compute what actions the robot and the processing cells in the single cluster tool should perform (and in what order) to get optimal performance of the cluster tool as a whole. In this case, optimal performance means that a pre-defined number of wafers should be processed in the smallest possible amount of time.

Before this performance question can be answered, first the actions performed in the cluster tool to process a wafer must be defined.

2.1 Robot actions

When you look careful at the robot, it becomes clear that its arm can be in one of four positions:

- In front of the wafer supply WS (position `10`).
- In front of processing cell P1 (position `11`).
- In front of processing cell P3 (position `13`).
- In front of processing cell P2 (position `12`).

Also, it can perform the following actions:

- Rotate clockwise 90 degrees (action `R1_turn`).
- Rotate counter clockwise 90 degrees (action `R1_turn_CCW`).
- Take a wafer from a position onto the robot by extending and retracting the arm (actions `pick_10`, `pick_11`, `pick_12`, and `pick_13`).
- Place a wafer from the robot onto a position by extending and retracting the arm (actions `place_10`, `place_11`, `place_12`, and `place_13`).

The positions and actions of the robot can also be drawn as a so-called state diagram, for example like Figure 3. Each position that the robot can be in is shown as a *state*, a circle. To differentiate between different states, each of them has a unique number. In this case, there are four states, one for each position of the arm.

The actions are shown as arrows that jump from one state to another (or they arrive at same state again). Each action has a label that states what action is being performed. For example,

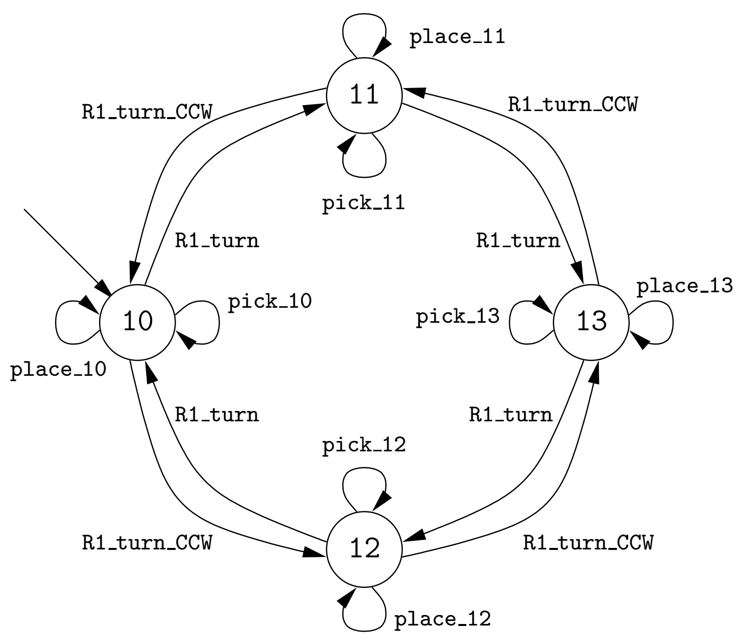


Figure 3: Robot R1 in a state diagram.

the arrow from position `11` to position `10` is labeled `R1_turn_CCW`, which means that the robot turns a quarter circle counter-clockwise to go from cell P1 to the wafer supply WS. At state `11`, it can also perform action `pick_11` and arrive again at state `11`, which represents picking up a wafer from cell P1 without rotating the arm. The initial position is defined with an unlabeled arrow pointing to the first state, in this case to state `10`.

As you can see in the state diagram, the robot can still do 'impossible' actions, for example perform two place actions after each other. This is however prevented by the state diagram of the machines, discussed next.

2.2 Processing cell and wafer supply actions

A processing cell does not change position in the physical sense of the word. It does however have three logical positions:

- Empty.
- Containing an unprocessed wafer.
- Containing a processed wafer.

. The state diagram of each processing cell is the same, except for the action labels. As an example, the diagram of P1 (at position `11`) is shown in Figure 4. The states are simply numbered consecutively, starting from 0. The first action it can do is a `place_11` together with the robot. Then it can perform a `process_11` action to process the wafer, and finally, the robot empties the processing cell with a `pick_11` action.

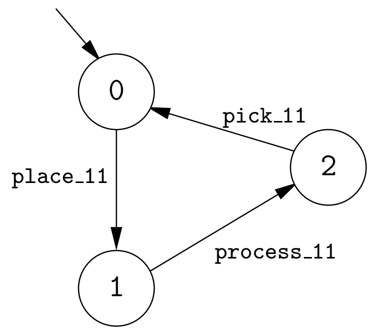


Figure 4: Processing cell P1 in a state diagram.

2.3 System states

With the same diagrams, you can also show how the whole system can progress, by merging the diagrams of all components. In Figure 5 you can see a simplified state diagram of the robot, wafer supply, and processing cell P1. It shows how the robot brings a wafer to the cell, the cell processes it, and the robot brings the processed cell back to the wafer supply. To show what state means 'task done', state `11` is a so-called marker state, shown by an outgoing arrow without label.

The simplified diagram of Figure 5 is quite easy to follow, because all states that obviously do not contribute to progress of the task have been removed. For example, in state `0`, the robot could first do a number of rotations before picking up the wafer (that is, in reality, state `0` should be replaced with four states, one for each position of the robot, with double arrows between them labeled with robot rotations). The same should be done with states `5`, `6`, and `11`. The rectangles formed by states `1`, `2`, `3`, and `4`, and by states `7`, `8`, `9`, and `10` should also have arrows in the reverse direction (the robot may also turn in the opposite direction a number of times). Last but not least, the `process_11` action is also possible after the robot turned a few times, leading to three more arrows from all expanded states of `5` to the equivalent expanded robot state of `6`. You can find the resulting state diagram in Appendix A.

As you can imagine, the number of states grows very rapidly with a larger number of processing cells or when there are more wafers to process. The computation also gets more complicated. For example, after delivery of a wafer in P1, the robot may not go back to the wafer supply and pick another fresh wafer, since it would have no place to bring the wafer to. Instead, it must first pick the processed wafer from P1, and bring it to P3.

Wonham has developed a supervisor synthesis theory [3] for computing such state diagrams for large systems. At our group, research is being done to further develop these techniques.

3 Performance

A part of the research in our group is about using state diagrams like above to compute performance. The question in this case is what actions have to be performed by the system (and in which order) such that all wafers are processed as fast as possible.

Performance is computed with a theory called Heaps of pieces by Viennot [2]. In this theory, a system is seen as a collection of components (called *resources* in the theory), that can perform

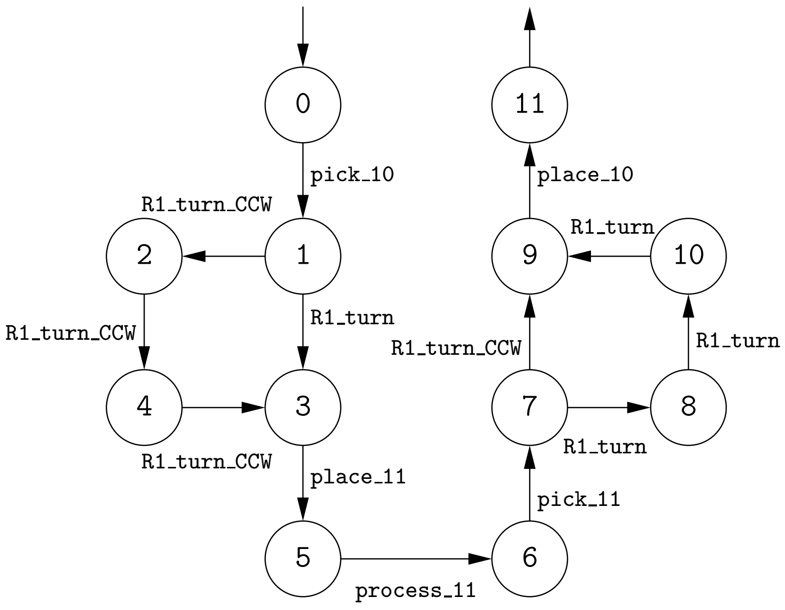


Figure 5: Simplified system state.

actions. Usually, only a few components participate in an action.

To reason about performance, each action gets a cost, in this case its duration. The following table shows the duration and the components involved for each action of the cluster tool used in the assignment:

Action	**Duration**	`Components`
R1_turn	1	R1
R1_turn_CCW	1	R1
pick_10	1	R1, WS
pick_11	1	R1, P1
pick_12	1	R1, P2
pick_13	1	R1, P3
place_10	1	R1, WS
place_11	1	R1, P1
place_12	1	R1, P2
place_13	1	R1, P3
process_11	11	P1
process_12	12	P2
process_13	13	P3

For each path in the system state diagram from start to end, all actions performed on that path are stacked on top of each other in a tetris-like grid. Horizontally, each component gets a column. Vertically, cumulative cost is shown.

The path starts at the starting state with an empty grid. Each time an action is traversed to reach the next state, a block is added to the grid. The height of the block represents the cost

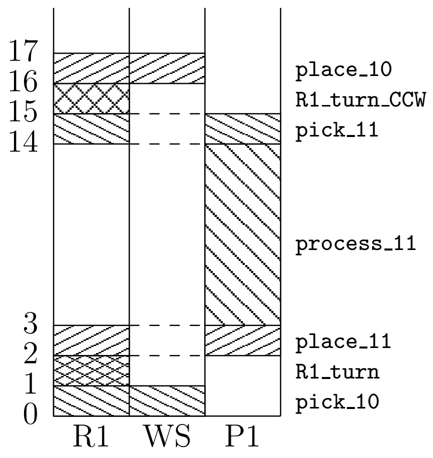


Figure 6: Costs of a traversal in the simplified system state diagram.

of the action (duration of the action in this case). Horizontally, the block is put in all columns of participating components. This stacking continues, until the marked end state is reached.

At the end state, all the stacks from all paths are compared with each other, and the one with the lowest cumulative costs is considered to have optimal performance.

As an example, consider the system state diagram of Figure 5. The tetris-like grid belonging to the path `0`, `1`, `3`, `5` `6`, `7`, `9`, and `11` is shown in Figure 6. At time 0, a `pick_10` can be performed both by R1 and by WS during 1 time unit. After the pick has finished, the robot can perform the `R1_turn` action, also taking 1 time unit. The next action is the `place_11` action. The processing cell P1 can start at time 0, but the robot R1 participates as well, so the earliest starting time of this action is at time 2. After the place action, the cell starts its processing with the `process_11` action. In a system where more wafers get processed at the same time, the robot could do other actions while P1 is processing. In the example however, the robot cannot do anything, and has to wait until time 14 before the next action can be performed. At that time, the three actions `pick_11`, `R1_turn_CCW`, and `place_10` can be done in sequence, arriving at marked state `11` at time 17. It is quite easy to see that this cost is the best you can do in the diagram with the given durations, so the path is optimal.

If the durations of the actions are changed however, the optimal path may change as well. Try for example to find the best performance when the duration of the `R1_turn_CCW` is 4 instead of 1.

4 Assignment

The objective of the assignment is to compute the best performance of processing a number of wafers in the cluster tool as shown in Figure 2. To this end, a C program has to be designed, implemented, and documented that reads ADS files containing the system state diagram, and outputs lines describing the actions performed at the optimal path.

There are four ADS files available for use: uncon_onelink1.ads, uncon_onelink3.ads, uncon_onelink10.ads, and uncon_onelink25.ads. The number in the filename indicates the number of wafers that must be processed. More wafers means a bigger system state diagram. Further details about the contents of the ADS files can be found in Appendix B.

The output of the program should be lines of the form

```
<component> <action> <start-time> <end-time>
```

As an example, the lines below would be the output of the path shown in Figure 6:

```
R1 pick_10 0 1
WS pick_10 0 1
R1 R1_turn 1 2
R1 place_11 2 3
P1 place_11 2 3
P1 process_11 3 14
R1 pick_11 14 15
P1 pick_11 14 15
R1 R1_turn_CCW 15 16
R1 place_10 16 17
WS place_10 16 17
```

The order of the lines is not relevant.

4.1 Questions

As a guide, some questions you may ponder about while designing the program:

- State diagram data structures, file loading, operations:
  - What data do you need to store from the ADS file?
  - If you have a state what information should you get (and how?) to make a step in your walk?
- Tetris grid:
  - What data do you need to store in the grid?
  - How do you add a block?
  - How do you compare two grids and find the best one?
- Walking (difficult!):
  - What do you need to save with each state?
  - How do you walk all possible paths?
  - What about cycles? (a sequence of steps that ends in the same state as where you started)
  - How do you know you are done?
- Scaling:
  - How fast is the program with larger number of system states?
  - How much memory does it use with larger number of system states?
  - How can you improve speed and/or reduce memory use?

Note there is a lot of hidden complexity in the 'walking' part, so make sure you have enough time for that part.

References

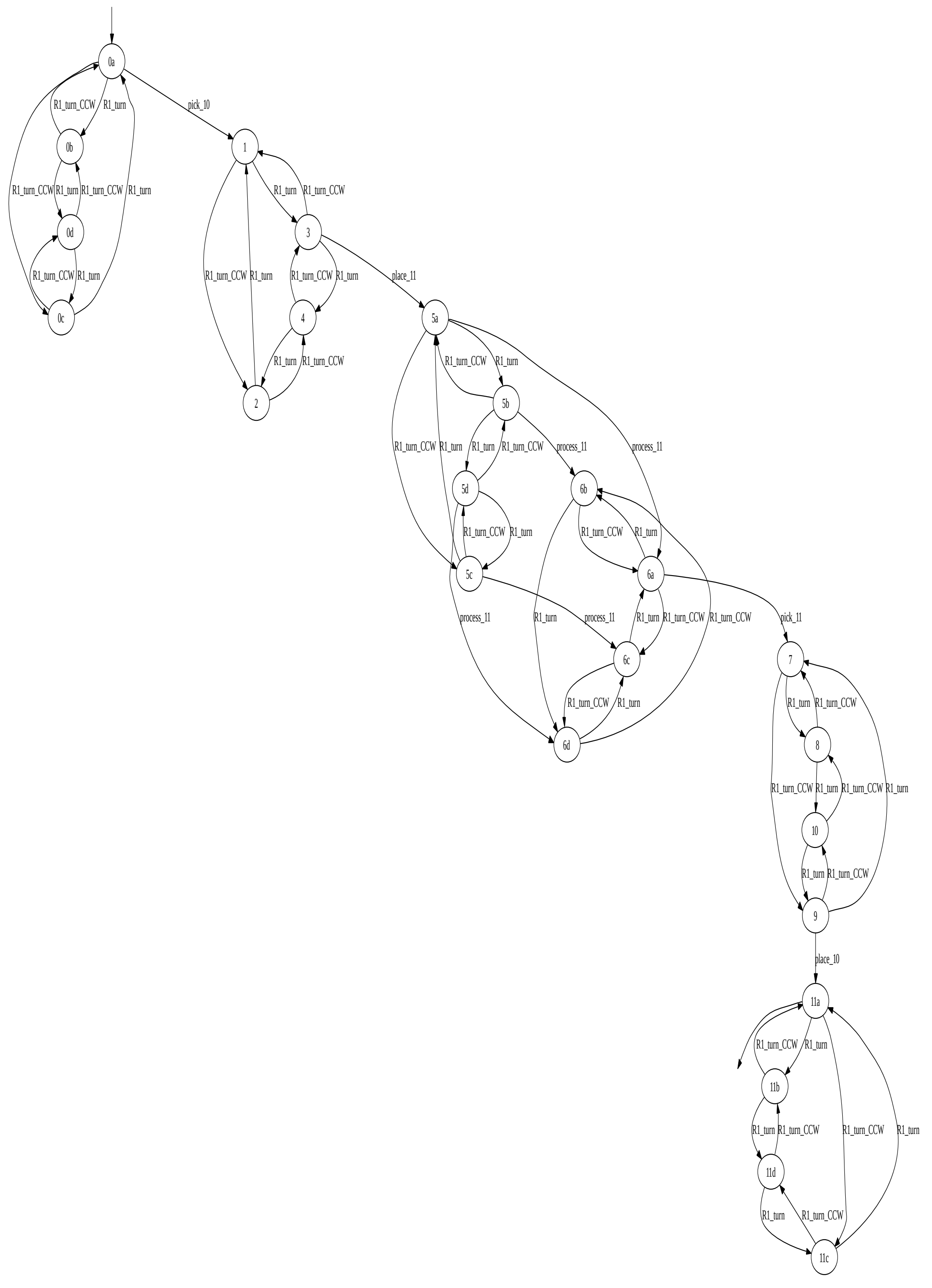
[1] P. Van Der Meulen. Linear semiconductor manufacturing logistics and the impact on cycle time. *Advanced Semiconductor Manufacturing Conference, 2007. ASMC 2007. IEEE/SEMI*, pages 111–116, June 2007.

[2] G.X. Viennot. Heaps of pieces. I: Basic definitions and combinatorial lemmas. In G. Labelle and P. Leroux, editors, *Combinatoire Énumérative*, pages 321–350. Springer-Verlag, Berlin, 1986.

[3] W. M. Wonham. Supervisory control of discrete-event systems. Technical Report ECE 1636F/1637S 2006-07, University of Toronto, 2006. Lecture notes.

[4] Jingang Yi, Shengwei Ding, M.T. Zhang, and P. van der Meulen. Throughput analysis of linear cluster tools. *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*, pages 1063–1068, Sept. 2007.

A Full system state diagram.



B Format of ADS files

An ADS file contains the system states of a cluster tool for processing a pre-defined number of wafers.

There are four ADS files available for use: uncon_onelink1.ads, uncon_onelink3.ads, uncon_onelink10.ads, and uncon_onelink25.ads. The number in the filename indicates the number of wafers that must be processed.

The file format is text, and uses numbers for everything. Below are the first 30 lines of the uncon_onelink1.ads:

```
# CTCT ADS auto-generated

uncon_onelink1

State size (State set will be (0,1....,size-1)):
# <-- Enter state size, in range 0 to 2000000, on line below.
48

Marker states:
# <-- Enter marker states, one per line.
# To mark all states, enter *.
# If no marker states, leave line blank.
# End marker list with blank line.
44

Vocal states:
# <-- Enter vocal output states, one per line.
# Format: State  Vocal_Output.  Vocal_Output in range 10 to 99.
# Example: 0 10
# If no vocal states, leave line blank.
# End vocal list with blank line.

Transitions:
# <-- Enter transition triple, one per line.
# Format: Exit_(Source)_State  Transition_Label  Entrance_(Target)_State.
# Transition_Label in range 0 to 999.
# Example: 2 0 1 (for transition labeled 0 from state 2 to state 1).
0     1   1
0     3   2
0     5   3
```

The number of states is 48 (at line 7), numbered from 0 to 47. The marked final state has number 44. The remaining 114 lines all have the same structure as the last three. Each line represents moving from one state to another while performing an action. The mapping of transition labels to actions is as follows:

action	**Transition label**
R1_turn	1
R1_turn_CCW	3
pick_10	5
pick_11	7
pick_12	9
pick_13	11
place_10	13
place_11	15
place_12	17
place_13	19
process_11	10
process_12	12
process_13	14

For example, the second line '`0 3 2`' means that action `R1_turn_CCW` is performed while going from state 0 to state 2.