Scheduling Approach for Microfactories with Setup Times

Mathias Coqblin, Dominique Gendreau, Philippe Lutz, Jean-Marc Nicod, Laurent Philippe and Veronika Rehn-Sonigo
FEMTO-ST Institute, UMR CNRS / UFC / ENSMM / UTBM, Besancon, France
# Corresponding Author E-mail: mathias.coqblin@femto-st.fr, TEL: +33-3-8166-2065, FAX: +33-3-8166-6450

KEYWORDS: Multicell Microfactory, Scheduling, Setup Time, Buffer

In this paper we consider microfactories for manipulation and assembly. These microfactories are composed of several cells containing microrobotic systems capable of a high level of repeatability. The assembly plan of the production is a pipeline of tasks that are performed by the cells. Our aim is to manage the production flow in the case where the cells can be reconfigured to perform different task types. Each cell is in charge of several consecutive tasks. A setup time is necessary to switch from the processing of one task type to another, and multiple intermediate results may be stored temporarily in storage areas to avoid switching the task type after the processing of each product. In this context we assess the optimized use of these storage areas, called buffers, and its impact on the production throughput.

1 Introduction

Microfactories are small production systems designed to manufacture microproducts [11]. In our work, we focus on the microfactories for manipulation and assembly. The manipulation or assembly tasks are performed by systems which have to be adapted to the microworld context. The considered systems are microrobotic systems whose architecture and control system permit to obtain high performance as high level of repeatability, resolution or even speed [7]. To perform complex manipulation and assembly tasks, the microrobots are grouped inside cells. Microrobotic cells often have a great number of degree of freedom (more than 10 for example). A microfactory could be a unique cell (most of the time in case of semi-automatic microfactories) but has to consist in several associated cells to allow a good management of a workflow. This flow has to be optimized to obtain the best global performance characterized by the throughput, the reliability, and the setup time.

Among the different properties that can be those of microfactories, we consider that we are able to reconfigure the cells. Different task types can be performed by the same cell. Moreover, the number of available cells is generally small compared to the number of tasks. Consequently, we assume that each cell in charge of several consecutive tasks as defined in an assembly plan. Switching from a task type to another requires to reconfigure the system, i.e. to change the associated tools, which induces an unavailability time called setup time [1]. Hence the whole production time for one piece includes the process time of each task as well as the setup times needed for cell reconfiguration. To organize the production flow, the storage areas described by their size and position are of great importance to define the cell organization of a microfactory. We name “buffers” the storage areas which can be containers or wafers.

Our work is motivated by the high cost – and the low availability – of reconfigurable cells in microfactories. The targeted production process is defined as a pipeline composed of several tasks, or steps. So before being completed a product has to undergo each of the tasks of the pipeline, as illustrated on Figure 3.1. Our goal is to maximize the overall throughput of the production. We consider that, due to their cost, the number of available stations is much lower than the number of tasks to be performed in the pipeline, hence the need to assign multiple tasks to a single production cell. As the number of machines is limited and the reconfiguration times are significant, using buffers is needed to greatly improve the throughput. If for instance setup times are at least the same as processing times, we are able to almost half the average period. On standard factories and assembly lines however, the use of buffers would not be as easy because the costs of keeping intermediate results usually exceed the benefits of using buffers.

The problem can be split in two sub-problems: first, find the correct allocation of tasks to production cells then find the optimal schedule of tasks within a cell (inner schedule). For the first problem we use an assignment called Interval Mapping, which means that a subset of consecutive tasks is assigned to the same cell [9]. We then concentrate on the second problem.

The scheduling issue is here to find a schedule (i.e. define an order to perform the tasks) inside a cell in the case where cells require a setup time to change from one task to another and where we consider the production of a batch of the *same* product. As stated before, we assume that intermediate productions can be temporarily stored in a dedicated space called buffer. Using buffers allow to perform several times the same task on different products and thus avoid to reconfigure the cell each time a new product arrives. In this context, as the goal of our work is to maximize the production throughput, this implies to minimize the cell unavailability and thus making the maximum use of the buffers is the key to reduce setup times and maximize the overall throughput. So the issues related to the global problem depend on the properties of the microfactory, in particular their buffer sizes, and on the properties of the tasks, their execution times and their setup times.

Considering that buffers may be of different sizes and differently used also involves scheduling and configuration issues. We must carefully choose the sizes of the buffers, which must be of reasonable size to avoid spending a long time filling them up or to meet a deadline condition. They also have to be consistent with each other to avoid unnecessary space allocation, and ensure the correct execution of any scheduling algorithm applied on the cells. Using buffers does not guarantee the overall optimization of the schedule: the schedule may be optimal for on a given machine, but outputting pieces by batches may delay the work of the next machine, hence reduce the overall throughput.

In the paper we tackle the problem of mapping tasks on cells taking setup times into account. The presented contribution is theoretical results on the complexity of mapping algorithms. For homogeneous production cells – cells with the same performance and storage capability – we propose a greedy scheduling algorithm that computes an optimal schedule. Then, for the case where the storage capacity differs from one cell to another, we show that the problem becomes considerably more difficult, indeed the mapping problem is strongly NP-Hard. On homogeneous platforms, when setup times are sequence-dependent [1] – they depend on the current task as well as on the previous one – the problem is also NP-Hard, but can be modeled as a Traveling Salesman Problem (TSP) [8].

The organization of the paper is as follows: first, we present the framework model in Section 3. In Section 4 we present our solution for single machine scheduling and interval mapping. In Section 5 we show the resulting execution of our solution on an example application. Then we conclude our work in Section 6.

2 Related work

Most of the research works involving reconfigurations focus on the ability of machines to process batches of pieces from a specific family, then to be reconfigured – or recalibrated – to process batches from another family. In other words, a single machine or a series of machines follow an assembly plan to create a product, then have to be reconfigured to follow another assembly plan that is totally unrelated, or at least require a recalibration. In this context, the problem of reducing the impact of setup times has been covered several times, mainly in semiconductor manufacturing. For instance, Zhang and Goldberg [12] focus on wafer-handling robots and propose a solution of eliminate costly manual re-calibration during component replacements. Li et al. [5, 6] study the problem of batch processing of incompatible lot families by reducing the total weighted tardiness. Jing and Li [4] provide a linear programming solution to minimize the total completion time in semiconductor factories.

Becker and Scholl [2] covered the problem of mapping tasks on machines in form of assembly line balancing problems (ALBP). In these kinds of problems typically one or more types of models have to be produced, and thus a precedence graph is mapped onto a linear assembly line. The setup part of the problems, however, focuses on the decision which type of piece has to be produced at a time, and when to reconfigure for another type of piece. The specific Interval Mapping problem has also been studied [3], and solutions are offered to map sequences of tasks on a lesser number of machines. These works however do not involve any reconfiguration in the process.

3 Framework model

Our study includes a theoretical contribution on task mapping algorithms. So before explaining how these algorithms are designed we first formally set the context of the work that will define its range. The production model relays on tasks that are performed by cells. We consider that a batch of the same product has to be realized, i.e., the same set of tasks is performed on each product in the same order. So when a product enters the production line it has to be processed by the whole set of tasks before being completed.

As the tasks are performed one after the other, always in the same order, their set can be modeled as a pipeline (Figure 3.1). A pipeline is made of a set $\mathscr{T}$ of n **tasks**: $\mathscr{T} = \{T_1, \dots, T_n\}$. The output of task T_i is the input of the next task T_{i+1}. Each task T_i requires an operating time w_i to be performed on the current product. As the aim of this production line is to output a huge amount of products out of the pipeline we concentrate on the steady state behavior of the line.

To perform tasks we use a multi-cell microfactory. We assume that these cells are interconnected by a transport system that can convey the products from one cell to every other cell. So we just take processing times into account and we do not take any transporting issue into consideration as the transport time can be neglected compared to the processing times. So the target platform is modeled as a set $\mathscr{M}$ of p machines: $\mathscr{M} = \{M_1, \dots, M_p\}$ interconnected as a clique. A processing speed v_u is associated to each cell M_u.

To execute a given application pipeline on a given platform, tasks are mapped onto machines considering consecutive tasks, then a inner schedule has to be planned. Each machine is indeed able to perform sequentially its allocated tasks. However, to switch from the processing of one task T_i to another task P_j ($i \neq j$), the machine M_u has to be reconfigured. This induces a setup time of $st_{i,j,u}$ time units. On the other hand it is possible to perform several times the same task on multiple input pieces without setup. This allows eventually to save setup times. However, the number of task repetitions is limited; each task T_i mapped onto M_u has an input buffer B_i, where the output parts from the previous task are stocked. Given this context, different versions of the model may be considered depending on the framework's heterogeneity in terms of setup times and buffers.

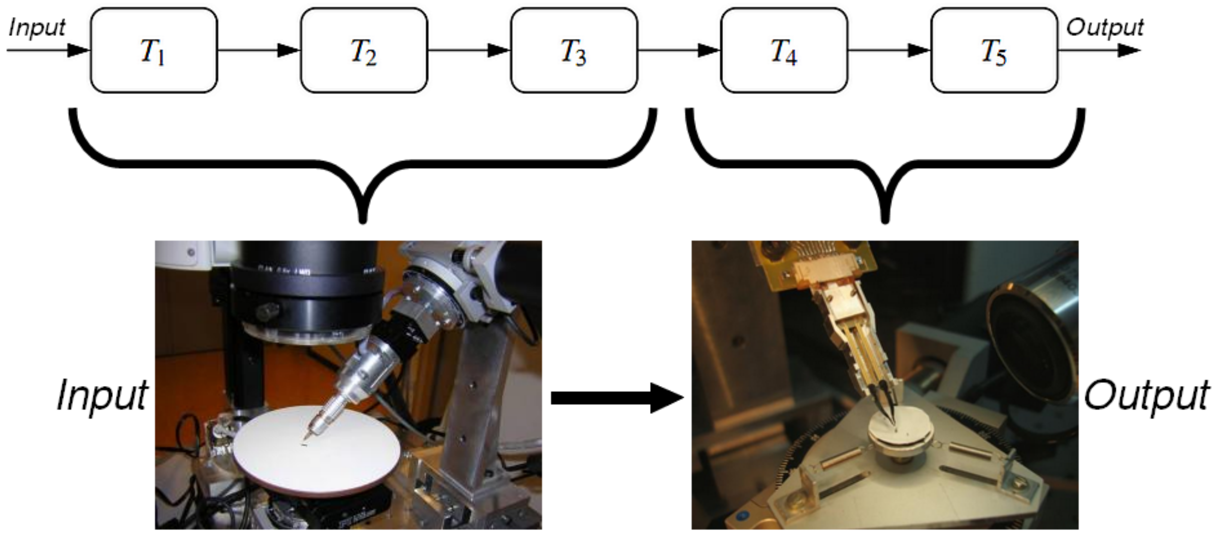

Figure 3.1: Example of tasks mapping on a microfactory: tasks 1 to 3 are mapped on the first cell, while tasks 4 and 5 are performed on the second cell

Setup times Considering the setup times, if the application is fully homogeneous, the setup times may be the same on every machine, and for every task. This will be referred to as **st**. On the other hand, on most applications, setup times will depend on the task we want to setup to. These times may be *sequence-independent* ($\mathbf{st_i}$), or *sequence-dependent* ($\mathbf{st_{i,j}}$). Sequence-dependent setup times mean that the setup time depends on both the task the machine is already configured for and the next task to be performed. Sequence-independent setup times usually occur where changeovers have minor influence, such as gripper calibration, or when both tasks are from the same type. Sequence-dependent setup times occur when there is a diversity amongst the tasks, such as going from handling and assembling parts to gluing them together.

The schedule problem with sequence-dependent setup times is NP-Hard and can be modeled as a Traveling Salesman Problem (TSP). It is not studied in the scope of this paper.

Buffers The problem of allocating available space into buffers may result in four variants:
• **B** All space is allocated evenly, so that every single task's buffer has the same capacity.
• $\mathbf{B_u}$ Likewise, if the available space is linked to a machine, buffers may be allocated to have the same capacity within a machine, but not the same amongst machines.
• $\mathbf{B_i}$ **(fixed sizes)** If the available space is not related to any machine, buffer sizes may have been fixed before any mapping is done. This is the most constrained variant we have to deal with.
• **(allocable sizes)** The most general problem, where the whole space is available within a machine, and we may freely choose the capacity of each buffer.

In the rest of this paper, we use the notation B_i to denote the name of the input buffer of task T_i, while b_i is the capacity of buffer B_i. When we are under the context of homogeneous buffer capacities (**B** and $\mathbf{B_u}$), it is implied that for all tasks b_i has the same value b (respectively b_u).

In this context, our objective function is to maximize the throughput of the production line: $\mathscr{T} = \frac{1}{\mathscr{P}}$ where $\mathscr{P}$ is the average period of time between two outputted products.

4 Scheduling with setup times and buffers

Our contributions cover several problems. First, since there are less cells available than there are tasks to perform, cells will have to perform several tasks. To find a solution that maximizes the throughput of the assembly line, we have to perform load balancing when assigning tasks on machines, so that we minimize the impact of the bottleneck cell.

Each machine then has a set of consecutive tasks to perform, which must be scheduled to avoid any unnecessary setup time involved. The latter is done by using buffers to stock intermediate productions at each step before treating them, allowing batch processing of the pieces.

Thus, our study will focus on three main operations:

- For a given machine with a given set of tasks to perform, finding an optimal schedule on a machine to maximize its throughput.
- Mapping tasks on machines as interval mapping – each machine has a set of consecutive tasks to perform. Assuming that on each machine a schedule is found that maximizes the throughput, the mapping has to be done in a way that allows to maximize the throughput of the machine with the lowest throughput - the bottleneck cell.
- Allocating the available space as input buffers for each task.

The combination of these three operations will allow to determine if our solution is optimal. The solution is optimal if:

- The schedule within a machine, as well as buffers allocation, perform the lowest period possible (or highest throughput).

- The bottleneck station on the pipeline – the one with the highest period – has no idle time while in steady state.

  Thus the whole pipeline has the same period as this bottleneck machine.

As a result, assuming we found an optimal solution, we know that the slowest machine will have a throughput as high as possible, and is never slowed down within the process. Since the overall execution of the pipeline is limited by the bottleneck station, we will be able to assess that the throughput of the application is maximum. However, even if the slowest machine has idle times, the solution may still be optimal if there is no way to meet this "no idle time" criteria.

We split this contribution section in two parts: the *single machine scheduling* section is dedicated to the schedule within a single given machine, and the *multi machine scheduling* section treats the mapping of tasks on the machines. We will not cover the buffer allocation part in the scope of this article, as most of the time the space can be evenly distributed amongst buffers, and other specific situations (such as a remainder of space that could be used to raise the size of some buffers only) are to be studied on a case-by-case basis. Thus, in the following sections we assume that the capacities of buffers are already fixed. That is, in regard to our framework model, only buffer models $\mathbf{B}$, $\mathbf{B_u}$ and $\mathbf{B_i}$ are considered, while *allocable sizes* are not.

4.1 Single Machine Scheduling

The approach for inner schedules is not directly linked to model variants. At machine level, the problem with buffers $\mathbf{B_u}$ is the same as $\mathbf{B}$, as they are both homogeneous buffer capacities within a machine; for all task T_i, the value of b_i is b. The problem with heterogeneous buffer capacities ($\mathbf{B_i}$) is a little more complex. Likewise, the heterogeneity of setup times (st or st_i) has absolutely no influence: some setups may take longer than others, but our algorithms will minimize the amount of setups the machine has to perform for each task, regardless of the time each will take.

Homogeneous buffer capacities We developed a greedy scheduling algorithm – GREEDY-B – that minimizes the period for homogeneous buffer capacities. The schedule is as follow: for buffers of size b, perform the first task all available pieces, that is b pieces. This will empty the first buffer and fill the next. Then we setup to the next task, where its input buffer is now full. The algorithm will continue to treat as many pieces as it can on each task before performing a setup to the next task. Since all buffers are the same, the progression is linear and after the last task the machine will setup back to the first task.

Heterogeneous buffer capacities Handling different buffer capacities is harder. This cannot be done the same way we did it with homogeneous buffers, except if set the limit of pieces processed on each task to the lowest buffer. The would however not be optimal, as better solution exist, that make better use of all buffers.

It is not possible to express a specific period for any random buffer sizes. However, it is possible to have a control over the behavior of the algorithm if all adjacent buffers are multiples to each others. More formally: $\forall i \in [1,..,n], \min(b_i, b_{i+1}) | \max(b_i, b_{i+1})$. In this configuration, we know that any buffer will be either x times larger or smaller than its predecessor, or its successor.

Based on that knowledge, we improved our GREEDY-B algorithm into GREEDY-BI. The idea is that once the cell is reconfigured for a specific task, the maximum amount of pieces that can be processed before a new setup is needed is limited by either the input buffer (the amount of pieces available) or the next buffer (the space available to store them). The minimum of both will be a hard limit; to stay active the cell will have to reconfigure to perform another task.

Keep in mind that by processing as many pieces as possible on a single task T_i, we minimize the impact of the setup time st_i on that task, as the ratio of st_i and the time needed to process all pieces on T_i is the lowest possible. Thus, when selecting the next task the cell will reconfigure to, we restrain this choice to maximize the use of buffers on the task. On GREEDY-BI, a reconfiguration for task T_i is done on task when on of the following conditions holds true:

- $b_i \geq b_{i+1}$, b_{i+1} is empty, and we can process enough pieces to totally fill b_{i+1}.
- $b_i \leq b_{i+1}$, b_i is full, and we can process enough pieces to totally empty b_i.

The conditions of the algorithm allow to fill perfectly the buffers when they are multiple to each others. If this is not the case, the behavior of the algorithm is undefined and depends on the actual implementation. We are then unable to work out an expression for the period, and cannot consider GREEDY-BI optimal for all configurations other than buffers to each others.

We however developed heuristics that aim to truncate the size of some buffers in order to have them multiple to each other. The resulting pipeline is a pipeline on which the execution of GREEDY-BI is well defined and optimal. However since we truncated some buffer, we may have lost setup time reduction potential.

4.2 Multi Machine Scheduling

Subhlok and Vondran [9, 10] addressed the problem of Interval Mapping on fully homogeneous platforms *without* setup times. They developed a dynamic programming algorithm to find an optimal mapping solution in polynomial time. This algorithm was later slightly adapted by Benoit and Robert [3] to find the optimal period.

The aim of the algorithm is to find a mapping that will eventually maximize the throughput of the application, of minimize its period, when running a schedule on each machine. Remind also that the period is determined by the period of the bottleneck machine. Through binary search, the algorithm will try to find a mapping that minimizes the period of the slowest machine; that is to say, it will try to minimize the value of the maximum period amongst the period of all machines. Every time the algorithm has a mapping to test on a machine, it will calculate its period and determine if it is better than the previous one.

We proceeded to adapt this algorithm to take setup times into account. The solution is the same algorithm as before, but the period calculation on a machine has been. This period is calculated according to the execution of the scheduling algorithm we would use.

Assuming an optimal inner schedule is found for each machine, this mapping solution has been proved optimal for any setup times (**st**, $\mathbf{st_i}$), and for **B** and $\mathbf{B_u}$. However, experimentations has shown that idle times may appear on the slowest machine when fully heterogeneous buffers capacities ($\mathbf{B_i}$) are used. This is due to the behavior of the schedule: before any new batch of n pieces is outputted, the machine must go through the process of all its tasks for n pieces. When buffers are heterogeneous, the size of those batches do not reflect the period of the machine – a faster machine may take too much time to output by small batches, while the slowest machine is waiting for its first big buffer to be filled before proceeding.

5 Results

In order to test the behavior of all our algorithms and heuristics, we have simulated them using the distributed system simulation tool SimGrid. With the simulator, we are able to test on platforms that reflect several real case scenarios, and compare the results with other (worse) solutions we come up with. We can also assess the execution behavior and the performance of the algorithms on different configurations.

In the following we give an example of execution for a system consisting of eight tasks and four cells, as shown on Figure 4.1. A total of 100 pieces are processed in the simulation. The configuration of the application is as follows:

- Tasks are mapped according to Figure 4.1, namely they are distributed evenly: two tasks per machine.
- Homogeneous setup times, fixed at $st = 2$ time units.
- Homogeneous buffers within machines ($\mathbf{B_u}$). As show on the figure, the buffer capacities are $b_1 = b_2 = 4$ on M_1, $b_3 = b_4 = 5$ on M_2, $b_5 = b_6 = 3$ on M_3, and $b_7 = b_8 = 5$ on M_4.
- All tasks take the same time to process a piece: 2 time units.

On this configuration, the slowest machine is M_3: as all machines have to perform the same amount of tasks and the process times are homogeneous, having buffer capacities lower than the other machines implies that the total time spent on reconfiguration is higher.

Figures 5.1 and 5.2 show two extracts from the Gantt chart obtained executing this configuration. The times are given in time units (compatible with any unit and coefficient that may suit a problem). The first sample – on Figure 5.1 – goes from $t = 219$ to $t = 309$, and the second sample – on Figure 5.2 – goes from $t = 441$ to $t = 531$. Each of the four lines corresponds to the activity of a machine (going from M_1 to M_4). Each black rectangle represents a setup while colored rectangles are the processing of a piece. The colors on the chart identify a single piece to track its location (colors are looped every 40 pieces to keep them distinguishable), and the values inside rectangles are the name of the task being processed. For instance, as illustrated on Figure 5.1, on M_1 (line 1) a yellow piece being processed on T_1 (a yellow rectangle marked as T_1) will later be found, after a setup, as a yellow rectangle marked as T_2 on the same line. It will then be found on the second line still as a yellow rectangle, marked as T_3 then T_4, and so on.

The output of the execution shows that, once the system has reached a steady state, the slowest machine M_3 has no idle time. On the Gantt chart the line corresponding to M_3 always shows a full activity, while other lines have time slots with no activity.

Both samples from the chart show that M_4 regularly has idle times. The machine is periodically in a state of starvation, as M_3 cannot deliver new pieces fast enough (for instance, on Figure 5.1 it is waiting for purple then red pieces to arrive from M_3). On Figure 5.2, we can observe that both M_1 and M_2 have inactivity. As the machines are located before M_3 on the pipeline, this is a case of saturation: M_3 cannot process pieces from its buffers fast enough, therefore M_2 has to wait for some space available on M_3 before proceeding. Then, by propagation, as M_2 is slowed down by M_3, M_1 is also slowed down by waiting for M_2 to empty its buffers.

As we can see, the throughput of the application is limited and thus determined by the throughput of M_3, the slowest machine. Assuming the mapping on which this execution is tested is optimal, M_3 has the highest throughput possible and we maximize the throughput of the application.

6 Conclusion

In this paper we have presented theoretical results on the problem of mapping tasks on the cells of a microfactory. These tasks are organized as a pipeline in the production process and in the case where several tasks can be mapped on the same cell our work relays on interval mapping which assumes that two consecutive tasks are mapped on the same cell. In this context we tackle the optimization of the production throughput depending on setup times, i.e. the time needed to switch from a task to another. We define a global model for the problem and propose some polynomial time heuristic to solve the problem.

As shown in Section 4 several problems are identified depending on the buffers and setup times properties. Most of them seems to be NP-Hard. So in our future works will concentrate on giving proofs for their complexities and defining efficient heuristics.

Acknowledgments

This work is part of the MUSINE2 project, supported by the Région de Franche-Comté.

Bibliography

[1] A. Allahverdi, C. Ng, T. Cheng, and M. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[2] C. Becker and A. Scholl. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3):694 – 715, 2006. ISSN 0377-2217. doi: 10.1016/j.ejor.2004.07.023. URL `http://www.sciencedirect.com/science/article/`

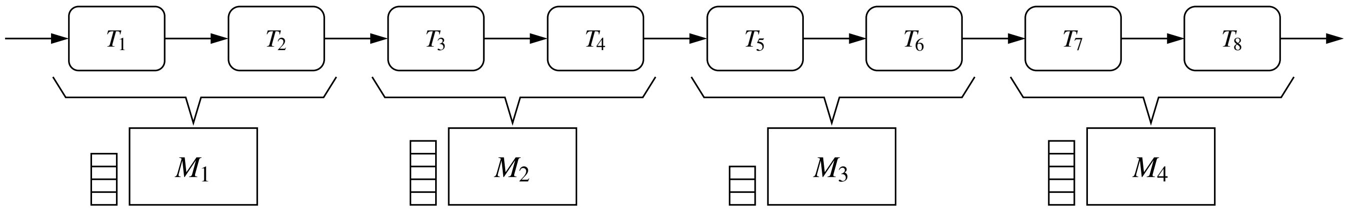

Figure 4.1: Example of execution configuration

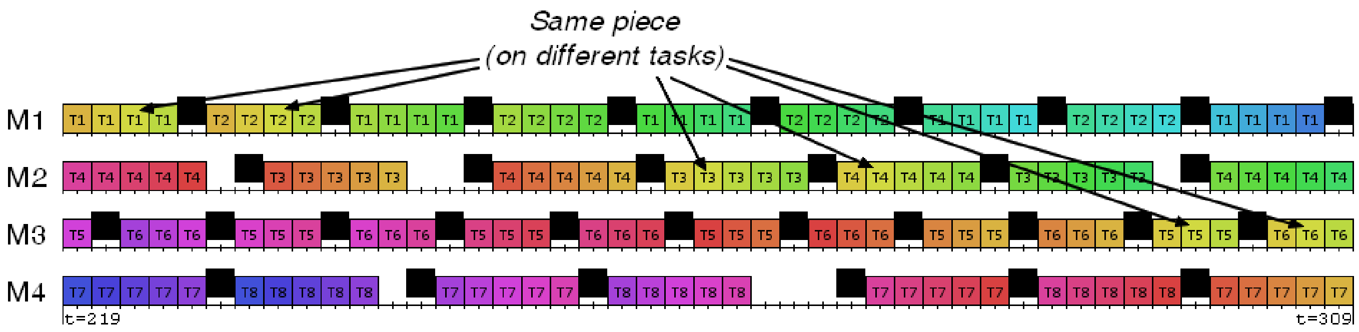

Figure 5.1: Example of execution (sample 1)

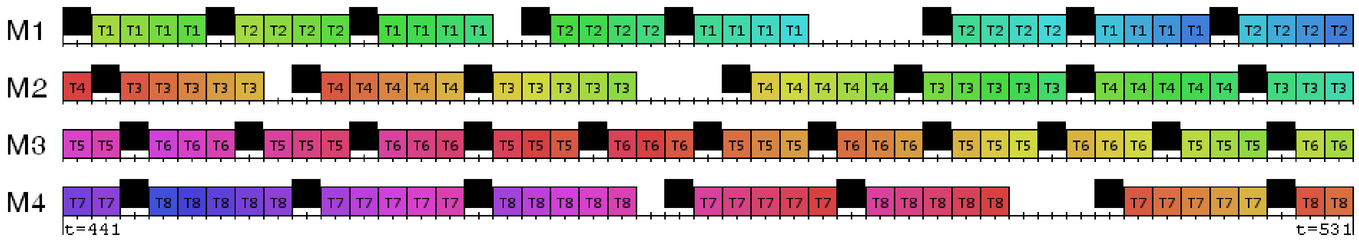

Figure 5.2: Example of execution (sample 2)

pii/S0377221704004801. <ce:title>Balancing Assembly and Transfer lines</ce:title>.

[3] A. Benoit and Y. Robert. Mapping pipeline skeletons onto heterogeneous platforms. *J. Parallel and Distributed Computing*, 68(6):790–808, 2008.

[4] X. Jing and Z. Li. A milp-based batch scheduling for two-stage hybrid flowshop with sequence-dependent setups in semiconductor assembly and test manufacturing. In *Automation Science and Engineering (CASE), 2010 IEEE Conference on*, pages 87–92. IEEE, 2010.

[5] L. Li and F. Qiao. Aco-based scheduling for a single batch processing machine in semiconductor manufacturing. In *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, pages 85–90. IEEE, 2008.

[6] L. Li, F. Qiao, and Q. Wu. Aco-based scheduling of parallel batch processing machines to minimize the total weighted tardiness. In *Automation Science and Engineering, 2009. CASE 2009. IEEE International Conference on*, pages 280–285. IEEE, 2009.

[7] M. Rakotondrabe, Y. Haddab, and P. Lutz. Development, modelling and control of micro/nano positioning 2 dof stick-slip device. *IEEE/ASME Transactions on Mechatronics, IEEE/ASME TMech/ *14* (6)*, pages 733–745, 2009.

[8] B. Srikar and S. Ghosh. A milp model for the n-job, m-stage flowshop with sequence dependent set-up times. *International Journal of Production Research*, 24(6):1459–1474, 1986.

[9] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *ACM SIGPLAN Notices*, volume 30, pages 134–143. ACM, 1995.

[10] J. Subhlok and G. Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, page 71. ACM, 1996.

[11] M. Tanaka. Development of desktop machining micro-factory. *Journal RIKEN Rev*, 34:46–49, April 2001. ISSN:0919-3405.

[12] M. Zhang and K. Goldberg. Calibration of wafer handling robots: A fixturing approach. In *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*, pages 255–260. IEEE, 2007.