University of Würzburg
Institute of Computer Science
Research Report Series

Using Simulation and Genetic Algorithms to Improve Cluster Tool Performance

Mathias A. Dümmler

Report No. 227 July 1999

University of Würzburg
Department of Computer Science
Am Hubland, D-97074 Würzburg, Germany
duemmler@informatik.uni-wuerzburg.de

Published in: 1999 Winter Simulation Conference, Phoenix, AZ, USA, December 5-8

Using Simulation and Genetic Algorithms to Improve Cluster Tool Performance

Mathias A. Dümmler
University of Würzburg
Department of Computer Science
Am Hubland, D-97074 Würzburg, Germany
duemmler@informatik.uni-wuerzburg.de

Abstract

In this paper, we present an approach to generate optimal processing sequences of lots at cluster tools. We consider the problem of sequencing n lots, where each lot can be processed by any of m available cluster tools. The proposed method combines simulation and a genetic algorithm to generate lot processing sequences. We show that our approach leads to a significant reduction of cycle times at cluster tools.

1 Introduction

Cluster tools have gained importance in the fabrication of semiconductor chips during the last decade. Especially in upcoming 300 mm fabs, they will be an integral part of the production process. In a cluster tool, several processing steps required to produce a semiconductor chip are integrated into a single piece of equipment. In this way, the probability of contamination of the wafers, the space required for equipment, and waiting and transport times are reduced.

Performance analysis of these tools is not straightforward and cannot be accomplished using simple analytic approaches like spreadsheets. Instead, a performance analysis tool that predicts cycle times of wafers in a cluster tool adequately has to take into account the effects of different wafer recipes, cluster tool control and architecture, wafer waiting times, and sequencing. Hence, cluster tools make for an excellent area for the application of simulation.

There is already a number of publications available that deal with the simulation and analysis of cluster tools. In [1], potential throughput advantages of cluster tools and the problem of shifting bottlenecks are discussed. [2] and [3] present simulation models of cluster tools that were developed using general purpose simulation languages. In [4] and [5], a general purpose simulation language has been used to build simulation models of different types of cluster tools. [6] addresses the problem of detecting and avoiding deadlocks when simulating cluster tools.

In [7] and [8], an analytical approach for computing the time required to process a lot of wafers is presented. However, their analytic method is restricted to very simple models of cluster tools. For example, process times are supposed to be identical in all chambers. In reality, however, process times vary significantly among different process chambers. Even for a single chamber process times can vary for different sequences. [9] use a similar approach to compute the steady state throughput of a single-wafer

cluster tool. They compare the throughput for a single-blade and a dual-blade robot. In [10] and [11], the throughput time of a cluster tool is derived from four parameters, namely the fixed throughput time, the lot size, the incremental throughput time and a correction term. [12] use timed Petri nets to model cluster tools. They also present a control method that increases the performance of a cluster tool.

Since several simulation tools for cluster tools are available, it is possible to assess the performance of individual cluster tools. However, these simulation tools can not be used for the performance analysis and control of a set of cluster tools in the context of a wafer fab. In this paper, we address a problem that arises in manufacturing practice, namely the sequencing of lots that wait for processing at a set of cluster tools. The sequence generated should minimize a certain cost function, e.g. it should lead to minimal cycle times for all lots. This task can, in general, not be solved without the support of a manufacturing execution system. However, the available systems usually do not take into account the special properties of cluster tools. For example, the cycle times of lots in a cluster tool running in parallel mode depend strongly on the types of lots that are processed in parallel. Therefore, a sequencing algorithm has to take into account the effect of lot combinations. To solve this problem, we propose an approach that combines simulation and genetic algorithms to generate sequences of lots at a set of cluster tools that lead to optimal or close–to–optimal cycle times.

This paper is structured as follows. In Section 2, the simulation program that we developed in order to model and simulate cluster tools is presented. In Section 3, we describe the cluster tool model that is used for our studies. Section 4 contains a simple approach to identify advantageous combinations of lot sequences. In Section 5, we present a genetic algorithm that generates processing sequences of lots that are optimal or close–to–optimal. Finally, in Section 6 we discuss our results and indicate possible extensions of our approach.

2 Simulation Engine

In order to perform the simulation studies described in the previous section, we developed a simulation engine for cluster tools in C++. The simulation model can consist of arbitrary many cluster tools, each of which can have an arbitrary number of process chambers, load locks and handlers. The most important parameters that can be specified in the simulation model are listed in Table 1.

For each of the components of a cluster tool, down times can be specified. However, for this study, down times were not incorporated in the model.

The simulation engine computes a number of output statistics after a simulation run, such as cycle times of wafers and utilization of the components of a cluster tool. For the studies described in this paper, the total completion time for a given sequence of lots is of special interest.

The modular concept of the simulation tool allows to exchange different parts of the model and test their impact on performance. For example, the module responsible for the control of the handler can currently be chosen from a set of four modules: a FIFO

Table 1: Model Parameters

Cluster tools:	- Number of process chambers - Number of load locks - Number of handlers
Handlers:	- Move time (with or without wafer) for every origin/ destination pair
Load locks:	- Pump/vent time
Sequences:	- Number of process steps
Process steps:	- Process chambers qualified for a step - Process time in these chambers
Lots:	- Number of wafers per lot
Wafers:	- Process sequence

based control, a Least Slack based control, a Critical Ratio based control and a module that uses backtracking to find optimal wafer moves.

3 Cluster Tool Model

The cluster tool model under investigation in this paper is depicted in Figure 1. It consists of two main modules to which the individual processing chambers are attached. Transportation of wafers in the upper module is done by the transfer robot, in the lower module the wafers are transported by the buffer robot. There are two load locks that allow to load lots into the cluster tool independently and process them in parallel.

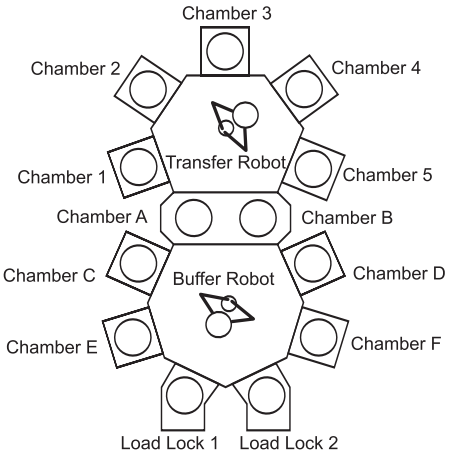

Figure 1: Cluster Tool Model

The model parameters are as follows. For both robots, we assume that it takes 20 seconds to move a wafer from any position (chamber or load lock) to another. Without

transporting a wafer, it takes the robots one second to move from one position to another. Pump and vent times for the load locks are zero, since they were not regarded in our study.

We have conducted simulation studies for this model for as many as 30 different process sequences. In this paper, we restrict the number of process sequences to four. The processing times in seconds at each chamber are listed in Table 2. A cell is empty if a wafer of the corresponding sequence does not visit the corresponding chamber. Note that in all process sequences, process time in chamber A is zero, since it is only used as a transfer to the upper main module.

Table 2: Processing Times in Chambers

Chamber	E|F	C	D	A	1	2	4	B
Seq. 1	80	60	40	0	70	40		30
Seq. 2	60			0	70			30
Seq. 3	80	60	40	0		90		30
Seq. 4				0			80	30

4 Lot Combinations

As the first step of our study, we tried to identify combinations of two lots that lead to low lot cycle times when processed in parallel. Therefore, we performed the following simulation experiment, using the cluster tool model presented in the previous section. Starting with an initially empty cluster tool, we simultaneously put a lot of process sequence $i \in \{1, \ldots, 4\}$ in load lock 1 and a lot of process sequence $j \in \{1, \ldots, 4\}$ in load lock 2. For all of the 16 possible combination of sequences, we measured the cycle time for both lots. For every combination i, j, we computed the ratios $T_{i,j}/T_i$, where $T_{i,j}$ is the cycle time of a lot of sequence i when processed together with a lot of sequence j. T_i is the cycle time of a lot of sequence i when processed exclusively. The resulting ratios are displayed in Table 3.

Table 3: Cycle Time Ratios for Combinations

Sequences	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 1$	1.86	1.85	1.85	1.35
$i = 2$	1.66	1.78	1.29	1.34
$i = 3$	1.50	1.38	1.88	1.39
$i = 4$	1.25	1.53	1.76	1.72

Obviously, there are combinations that lead to more favorable cycle times for both lots than other combinations. For example, combining sequence 2 and 3 leads to an increase in cycle time of 29% for sequence 2 and of 38% for sequence 3. Hence, this is a more favorable combination than, e.g., sequence 1 and 3, which leads to an increase

in cycle time of 85% for sequence 1 and of 50% for sequence 3. Combining process sequences 2 and 3 leads to shorter cycle times since sequence 2 does not make use of chamber 2, which is the bottleneck chamber of sequence 3 and vice versa. On the other hand, when combining sequences 1 and 3, wafers of sequence 1 visit chamber 2, the bottleneck resource for sequence 3, causing higher waiting times for both sequences.

Table 3 can be used as a guideline for operators to choose combinations of process sequences that lead to short cycle times. However, when a large number of lots has to be sequenced on several cluster tools, this task can no longer be performed manually.

5 Lot Sequencing

The next step is to automate the sequencing of lots at cluster tools. It can be shown that for n lots and m cluster tools, there are $(n+m-1)!/(m-1)!$ ways to distribute the lots over the cluster tools. It is obvious that even for small values of n and m it is not feasible to test all possible sequences. For example, for $n = 8$ lots and $m = 2$ cluster tools, 362,880 simulations would have to be run to do an exhaustive search over all possible sequences.

To solve this problem, we implemented a heuristic method based on a genetic algorithm (GA) to generate the lot sequences. For an introduction to genetic algorithms, the reader is referred to [13] and [14]. For the implementation of the genetic algorithm we used the programming library "GAlib" [15]. Since GAlib is written in C++, it could be easily integrated into our existing simulation tool.

The basic idea behind a genetic algorithm is to imitate an evolutionary process: The best individuals or *genomes* of a generation survive and reproduce to pass on their genetic material to the next generation. Roughly spoken, the GA needs three pieces of input data: A data structure to represent the genomes, operators on this data structure that allow the genetic algorithm to create new solutions and an objective function to evaluate the *fitness* of a genome.

In our case, the genomes of the GA are represented as follows. If n lots, numbered $1, \ldots, n$, have to be scheduled on m cluster tools, numbered $1, \ldots, m$, a genome consists of a list of integer numbers $l_k \in \{1, \ldots, n\}$, $k = 1, \ldots, n$ and an array of integer numbers $a_k \in \{1, \ldots, m\}$, $k = 1, \ldots, n,$. The list l_k represents the processing sequence of the lots and the array entries a_k denote the cluster tool on which lot k is scheduled for processing. We use the default operators on lists and arrays that are implemented and documented in the GAlib library to generate new genomes. As the objective function, we use the time required for processing all lots according to the sequence that the genetic algorithm suggests. This time is derived using the simulation model of the cluster tools. The GA uses this objective function to evaluate a genome and to decide whether it is "fit" enough to survive and reproduce.

Three problem instances have been used to test the genetic algorithm. In the first problem, four lots, one of each process sequence, have to be sequenced for processing at a single cluster tool. The optimal sequence can be found in this case by simulating all $4! = 24$ lot sequences. This sequence has a makespan of 10031 seconds. The GA was run five times for this problem. The parameters of the GA can be found in Table 4.

Table 4: Parameters for Problem 1

Population size	5
Number of generations	5
Probability of crossover	0.6
Probability of mutation	0.1
Number of replacements	2

The results of the GA are displayed in Table 5. For each of the five test runs, the best lot sequence that the GA found is displayed. In the following columns, the makespan for the best sequence, the number of sequences tested to generate the sequence and the total run time of the algorithm are given. The runs were performed on a Pentium II 266 processor.

Table 5: Results for Problem 1

Run	Best Lot Sequence	Make-span	Sequen-ces tested	Run Time (sec.)
1	4 1 2 3	10036	13	9
2	1 4 3 2	10052	9	6
3	3 2 4 1	10031	14	9
4	2 3 4 1	10142	10	6
5	2 3 4 1	10142	14	10

The GA found the optimal sequence in one of the five runs, the results for the other runs differed not more than one percent from the shortest makespan. However, instead of testing all 24 sequences, the GA needed to test only 14 sequences to find its best solution.

In the second problem, two lots of each process sequence are sequenced for processing on one cluster tool. The parameters of the GA are given in Table 6.

Table 6: Parameters for Problem 2 and 3

Population size	20
Number of generations	10
Probability of crossover	0.6
Probability of mutation	0.1
Number of replacements	8

Five test runs have been performed. The results are displayed in Table 7. For each run, the makespan of the best sequence that the GA found is displayed. To the author's knowledge, no algorithm is available that finds the optimal sequence of lots for this problem in adequate time. Therefore we compare the makespan of the best sequence to the average makespan of 20 randomly generated sequences. The reduction in makespan

is given in the third column. Finally, the number of sequences generated to find the optimum and the run time of the GA are displayed.

Table 7: Results for Problem 2

Run	Best Make-span	% Improved	Sequen-ces tested	Run Time (sec.)
1	19562	12.5	114	160
2	19840	11.3	113	157
3	19601	12.3	117	161
4	20067	10.3	105	149
5	19691	11.9	115	163

In all five runs, the GA produced a sequence that leads to more than ten percent reduction in makespan compared to the randomly generated sequences.

Finally, in the third problem three lots of each process sequence are sequenced for processing on two cluster tools. The parameters used in this case are the same as in Table 6. The results of five test runs are displayed in Table 8. Again, the improvement in makespan for the best sequence is compared to the average makespan of 20 randomly generated sequences. The random sequences were generated by evenly distributing the lots over the cluster tools.

Table 8: Results for Problem 3

Run	Best Make-span	% Improved	Sequen-ces tested	Run Time (sec.)
1	14418	10.8	111	213
2	14182	13.0	100	197
3	13979	13.7	89	165
4	14059	14.9	111	213
5	13860	14.9	118	228

It can be noted in general, that the GA generated sequences that lead to an optimal or near–to–optimal makespan. It is expected that applying the presented approach on the manufacturing floor will lead to a significant reduction of cycle times.

The run time of the GA is small enough to make it useful in the actual dispatching of cluster tools. Sequences can be re–optimized in only a few minutes if the set of lots waiting for processing changes. Another advantage of the genetic algorithm is that it is an *anytime* algorithm. This means, that if a result is needed before the genetic algorithm has terminated, the computation can be stopped and the genetic algorithm will respond with the best current solution.

6 Conclusion

In this paper we presented an approach that uses a simulation model of a set of cluster tools and a genetic algorithm (GA) to find optimal processing sequences of lots at these cluster tools. Several sample applications showed that the proposed method can be used to produce optimal or close–to–optimal sequences in short time. When applied on the production floor, this algorithm can lead to a significant reduction in cycle times.

Several improvements to the approach are possible. For example, a GA-within-GA might lead to better performance. In a first step, this algorithm distributes the lots over the cluster tools and then, as a second step, tries to find optimal sequences for each individual cluster tool.

ACKNOWLEDGEMENTS

The author would like to thank Matthias Schmid, Markus Bohr and Andreas Reifert for their programming efforts.

References

[1] R. W. Atherton, F. T. Turner, L. F. Atherton, and M. A. Pool, "Performance analysis of multi-process semiconductor manufacturing equipment," in *Proceedings of IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 131–136, 1990.

[2] N. G. Pierce and M. J. Drevna, "Development of generic simulation models to evaluate wafer fabrication cluster tools," in *Proceedings of Winter Simulation Conference*, pp. 874–878, 1992.

[3] T. H. LeBaron and M. Pool, "The simulation of cluster tools: A new semiconductor manufacturing technology," in *Proceedings of the Winter Simulation Conference*, pp. 907–912, 1994.

[4] J. L. Mauer and R. E. Schelasin, "The simulation of integrated tool performance in semiconductor manufacturing," in *Proceedings of Winter Simulation Conference*, pp. 814–818, 1993.

[5] J. Mauer and R. Schelasin, "Using simulation to analyze integrated tool performance in semiconductor manufacturing," *Microelectronic Engineering*, vol. 25, no. 2/4, pp. 139–146, 1994.

[6] L. W. Schruben, "Deadlock detection and avoidance in cluster tools," in *Proceedings of the 1999 International Conference on Semiconductor Manufacturing Operational Modeling and Simulation*, pp. 31–35, 1999.

[7] T. L. Perkinson, P. K. McLarty, and R. S. Gyurcsik, "Single-wafer cluster tool performance: An analysis of throughput," *IEEE Transactions on Semiconductor Manufacturing*, vol. 7, pp. 369–373, Aug. 1994.

[8] T. L. Perkinson and R. S. Gyurcsik, "Single-wafer cluster tool performance: An analysis of the effects of redundant chambers and revisitation sequences on throughput," *IEEE Transactions on Semiconductor Manufacturing*, vol. 9, pp. 384–400, Aug. 1996.

[9] S. Venkatesh, R. Davenport, P. Foxhoven, and J. Nulman, "A steady-state throughput analysis of cluster tools: Dual-blade versus single-blade robots," *IEEE Transactions on Semiconductor Manufacturing*, vol. 10, pp. 418–424, Nov. 1997.

[10] S. C. Wood, S. Tripathi, and F. Moghadam, "A generic model for cluster tool throughput time and capacity," in *Proceedings of IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 194–199, 1994.

[11] S. C. Wood, "Simple performance models for integrated processing tools," *IEEE Transactions on Semiconductor Manufacturing*, vol. 9, pp. 320–328, Aug. 1996.

[12] Y.-H. Shin and T.-E. Lee, "Performance modeling of cluster tools using timed petri nets," in *Proceedings of the 1999 International Conference on Semiconductor Manufacturing Operational Modeling and Simulation*, pp. 36–41, 1999.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[14] "Encore (The EvolutioNary COmputation REpository network)." ftp://ftp.krl.caltech.edu/pub/EC/Welcome.html, 1997.

[15] M. Wall, "GAlib. A C++ library of genetic algorithm components." http://lancet.mit.edu/ga/, 1995.