Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

AUTOMATED GENERATION AND PARAMETERIZATION OF THROUGHPUT MODELS FOR SEMICONDUCTOR TOOLS

Jan Lange
Kilian Schmidt
Roy Börner

AMD Saxony LLC & Co. KG
Wilschdorfer Landstrasse 101
Dresden, 01109, GERMANY

Oliver Rose

Institute of Applied Computer Science
Dresden University of Technology
Dresden, 01062, GERMANY

ABSTRACT

Cluster tools play an important role in modern semiconductor fabs. Due to their complexity in configuration and their varying material flow, the creation of accurate throughput models for cluster tools is a demanding task. Proposed analytical approaches are either quite intricate and require manual maintenance process, or are inexact due to reliance on lot-level events. This paper presents an approach for the automated generation and parameterization of detailed cluster tool models based on bottleneck analysis. Equipment configuration as well as all throughput model base data is extracted from recent equipment reporting data.

1 INTRODUCTION

Cluster tools are very flexible, since they combine multiple process steps in a single mainframe. Because of this flexibility cluster tools are important for modern semiconductor manufacturing. To analyze the throughput capability of cluster tools and their respective process recipes, it is necessary to build throughput models.

Two types of throughput models are used in the industry. One is the generation of detailed models by spreadsheets. Through their complexity, those models are created and maintained manually. Consequently, these tasks are very time consuming and fault-prone. In addition, to include recent data it is necessary to synchronize the actual tool data to the model. The other type is the automated generation and calculation of cluster tool throughput models. This type relies on simplified models which are based on lot-level events. In these models, cluster tools are regarded as black boxes. More detailed and realistic analytical models,which are generated automatically, often are not feasible because of the high effort the modeling of such complex tools takes and the lack of necessary base data. Moreover, the simple black box models do not provide any information about how the tool is working inside and how the process inside the tool could be optimized. With a black box model it is not possible to generate useful data for planned future recipe variants with new material flows. To overcome this problem, a detailed model is necessary, which can represent each of the cluster tool's entities. In the following these entities are also called components or sublocations.

This paper presents an automated generic creation approach for detailed cluster tool models which allow to determine the throughput behavior of cluster tools.

In the following sections we present the approach and the implementation of the automated generation of throughput models. Section 2 introduces the model, which contains all tool data relevant to the throughput calculation. Section 3 shows how the tool model is filled with data from available source data. Section 4 describes how the throughput is calculated and how it is implemented. Section 5, provides an overview of possible applications. The paper closes with a conclusion in Section 7.

2 CLUSTER TOOL MODEL

In this section we present a flexible model for cluster tools, which is the basis for successional throughput calculation.

2.1 Tool Configuration

An exemplary structure of a cluster tool is depicted in Figure 1. In this simple example the cluster tool comprises two load locks (LL), a robot (R) and three process chambers (CH). A piece enters and leaves the tool via the two load locks and is processed in process chambers. The robot carries pieces between load locks and process chambers. Every component of the tool has model-relevant characteristics. A component's piece capacity is one of them. Other model-relevant characteristics are component-specific times. In the following, these times are called static times. Static times are

- Entity Pre-Wait Time (ET_{pre}),
- Entity Post-Wait Time (ET_{post}) and

978-1-4244-2708-6/08/$25.00 ©2008 IEEE

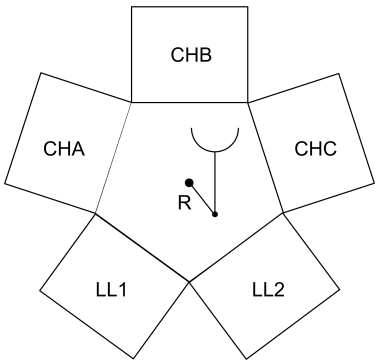


Figure 1: Example of a Cluster Tool

- Entity Swap Time *(ET_{swap})*.

ET_{pre} is the time an entity needs before processing. ET_{post} is the time after processing. Finally, ET_{swap} is the minimum time between the end of ET_{post} of one piece to the start of ET_{pre} of a following piece.

2.2 Material Flow

Cluster tools can process pieces on different routes and recipes. A flow is the processing route a piece takes through the tool. Thereby, the *flow* comprises a sequence of steps representing process actions on the entities. This can be a processing step in a chamber or a movement on the robot, for example. The time a piece is processed on a step is called *Entity Process Time (ET_p)*. This time depends on the flow's step and also on the process recipe. Figure 2 shows an exemplary flow with seven steps. In the example the piece processing on step five can be done either in CHB or in CHC.

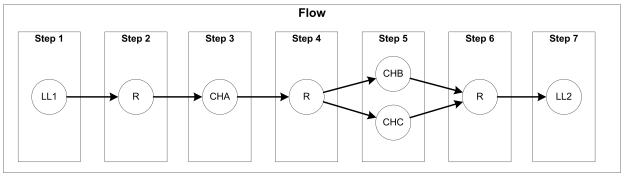


Figure 2: Exemplary Flow

2.3 Model

The model shown in Figure 3 merges the cluster tool configuration and the material flows. Each tool contains a number of entities and also a number of flows. Whereby a flow comprises a number of steps. The component *Process Times* carries the values for ET_p. The component combines the contexts of step, entity and the specific recipe. Beside the entity capacities, all static times are stored in a component attached to the according entity.

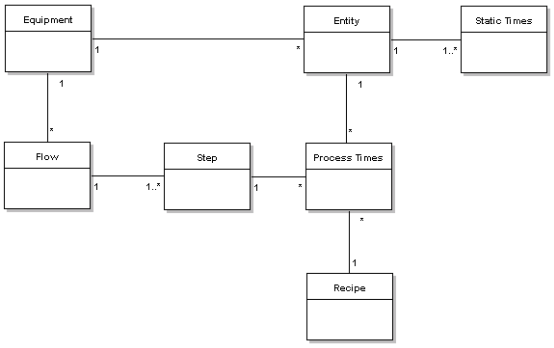


Figure 3: Cluster Tool Data Model

For throughput calculation we use the bottleneck analysis approach. Bottleneck analysis is a very simple and powerful method to calculate the throughput of a process line. Also a flow inside a cluster tool can be regarded as a process line. According to (Hopp and Spearman 2001), '...the rate of a line is ultimately determined by the bottleneck, or slowest, process.' However, in general, bottleneck analysis is not regarded as appropriate for cluster tools. In (Hopp and Spearman 2001) it is stated '..that few manufacturers can identify their bottleneck process with any degree of confidence [...] Most systems involve multiple products with different processing times. As a result, the bottleneck machine for one product may not be the bottleneck for the other product. This can cause the bottleneck to "float" depending on the product mix.'

Also (Dümmler 2004) states 'Whereas this approach can produce satisfactory results for simple cluster tools with fixed routing, it is not applicable in the case of flexible-sequence tools with changing recipe mix [...] Therefore, identifying the bottleneck is very difficult or not possible at all.'

These objections give rise to a serious challenge, but a solution is feasible. Our tool will provide fine-grained reporting data with acceptable quality. Also, it is possible to separate product mixes to create a throughput model for each single recipe. Consequently, the two basic tasks to realize this model are flow separation and data evaluation.

In contrast to other approaches, e.g. (Perkinson et al. 1994) and (Wood et al. 1994) we do not distinguish between transport-bound and process-bound schedules. This means, all entities are treated the same.

3 DETERMINE MODEL DATA

The model data consisting of a tool configuration and entity times, is extracted from simple reporting events *(baseline events)*. Baseline events are generated by tool interfaces and carry the following information:

- Tool Identifier
- Piece Identifier
- Recipe Identifier
- Event Name
- Timestamp

Static times as well as process times are determined by the difference of the timestamps of two events. However, since process times are flow-specific and static times are focused on a single entity, the calculation of process times and static times is strictly separated.

3.1 Checking Input Data

Although baseline events come as a sorted list, the correct order cannot be guaranteed. Missing events can be found by subsequent algorithms, but switched events can lead to serious calculation errors. However, it is guaranteed that only events with equal timestamps can appear in wrong order.

A special component checks the event sequence syntactically and corrects it if necessary. This component utilizes a freely configurable directional graph, which defines all possible syntactical event interrelations.

3.2 Flow Separation

The separation of flows is an essential part of our approach. A new candidate flow is evaluated against all known flows of the concerning tool. We utilize a decision tree with three stages for this purpose.

Stage One checks, if there is already a flow with the same length. If there is no such flow, the candidate flow is a new flow. In case there is such a flow, the candidate flow is sent to stage two.

Stage Two compares the candidate flow with flows of equal length. If the candidate flow fits into an existing flow, the values of the candidate flow are merged to the old flow. If not, the candidate is sent to stage three.

Stage Three compares the similarity of the candidate flow to any of the flows with equal length with a pattern matching algorithm. On the base of entity names, this algorithm compares every step from the candidate with its equivalent from the old flow and values their similarity. Akin flows are merged, this means the old flow is modified by adding recent occurred entities from the candidate flow. If the candidate is not equal to any of the old flows, it is characterized as a new flow.

3.3 Determining Process Times

At runtime, an algorithm processes a list of baseline events. The list is ordered by piece and time. This means, the flow of each piece is constructed and evaluated sequentially. The algorithm collects baseline events for creating virtual step containers, which are connected to a virtual flow container. In case of a new piece, both, virtual flow container and virtual step containers are transformed to the cluster tool data model as stated in Figure 3. Thereby, the flow is also evaluated (see Section 3.2).

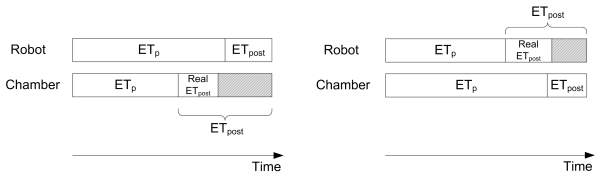
3.4 Determining Static Times

In contrast to the computation of process times, the algorithm for static times processes a list, which is ordered by entity and time. This means, every sublocation is processed sequentially. Although for static times it is not neccesarry to interpret the flow behavior, other issues have to be considered. Static times can be corrupted.

Figure 4 shows two small Gantt-Charts for two entities illustrating this corruption effect. In Figure 4a the chamber has to wait for the robot. This means, the robot is the limiting resource, whereas the chamber is the negatively affected resource. The period measured for ET_{post} at the chamber is corrupted by the additional waiting time. However, the time measured for the robot's ET_{post} is correct. In Figure 4b we show the case where a robot's ET_{post} is corrupted. A solution to this problem requires the knowledge of limiting resources, which has to be estimated from an already existing tool model. But, as the model itself has to be created, this obviously goes beyond the possibilities a simple list of baseline events can provide. Consequently, a much simpler solution is implemented. The approach benefits from the tool behavior during the run-in and run-out periods, when the limiting resource does not lead to additional waiting times. Figure 5 shows an example of a typical run's static time value distribution for a sublocation. Here, it is not relevant whether the static time is ET_{pre}, ET_{post} or ET_{swap}. To determine the actual value, the smallest reasonable value has to be found. But it has to be taken into account, that through missing baseline events and therefore erroneous parsing outliers may occur. To eliminate these dangerous outliers, we cut off the smallest values by using a percentile rule. The size of this percentile can be set in the configuration. However, a value of five percent proved to be useful.

4 THROUGHPUT CALCULATION

In (Wood et al. 1994) and (Wood 1996) Wood et al. introduced a generic model, which is based based on 'Fixed Throughput Time' T and 'Incremental Time' t. T is the base time, which is independent from lot size. Every piece increases the whole throughput time by t. The formula



(a) Robot Limited (b) Chamber Limited

Figure 4: Limiting Resources

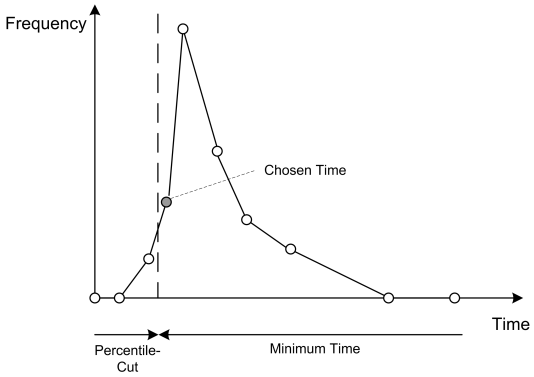


Figure 5: Select Representative Static Time Value

proposed by Wood et al. is $THP = min(\frac{n_L \times l}{T + l \times t}, \frac{1}{t})$, where l is the lot size in wafers and n_L is the number of lots. This approach distinguishes between handler bottleneck (robot bottleneck) and module bottleneck (chamber bottleneck) and also between serial and parallel configuration of the cluster tool. The approach we use is basically very similar to the approach of Wood et al. having a fixed time and an incremental time, but without using the distinction in the kind of bottleneck or tool configuration.

Figure 6 shows a Gantt chart with four pieces. We can see that a cluster tool can process multiple pieces simultaneously. The processing is done sequentially including pipelining effects. The time it takes to process a single piece is defined by the length of the whole process flow. This time is called *First Wafer Time* or PT_1. Any of the following pieces leaves the tool with an offset time to the preceeding piece. This Time is called *Process Interval Time* or *PI*. The Process Interval Time is defined by the bottleneck step of the flow or process line. In steady state, the throughput of a process line is simply defined by its slowest component. For this simple case we calculate the throughput by $THP = \frac{N}{T}$ or $THP = \frac{1}{PI}$, where N is the number of pieces, which are processed in a time T. Comprising the effect of the first wafer, we calculate the expected throughput with $THP = \frac{n \times l}{PT_1 + (n \times l - 1)PI}$. Consequently, only PT_1 and PI are relevant factors. The following subsections will provide formulas for PT_1 and *PI* lead from the simplest case of a flow to a general flow.

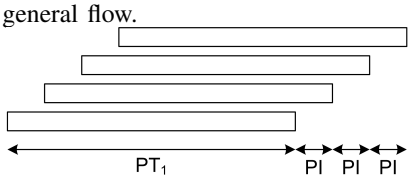


Figure 6: First Wafer Time (PT_1) and Process Interval (*PI*)

4.1 Simple Flow

For a simple flow, where the steps have only single entities, which do not re-occur. PT_1 and PI are

$$PT_1 = \sum_{s=steps} (ET_{pre_s} + ET_{proc_s} + ET_{post_s})$$

and

$$PI = \max_{s=steps}(T_s); T_s = \frac{ET_{pre_s} + ET_{proc_s} + ET_{post_s} + ET_{swap_s}}{WaferCapacity},$$

where the *Wafer Capacity* is the one from the entity of the step.

4.2 Flow with Parallel Entities

As steps can have parallel entities, these have to be taken into account. PT_1 is the time the first piece takes to run through the process flow. We have to find a representative value for parallel entities with differing processing times. Like every piece, the routing of the first piece is controlled by the tool in runtime. The tool routing strategy can depend on several unknown factors. Without any additional information, it is unclear, whether the chosen route is the fastest variant, the slowest variant or any in between. So for the whole flow, we simply calculate the average value of all entities of each step. The resulting error can be reduced with more knowledge about the specific tool's routing strategy. Moreover, the error is less significant for long wafer cascades, since the effect of PT_1 on the whole throughput decreases with an increasing number of wafers. We calculate

$$PT_1 = \sum_{s=steps} (\bar{ET}_{pre_s} + \bar{ET}_{proc_s} + \bar{ET}_{post_s})$$

for a flow with parallel entities.

For steps with parallel entities the *Interval Time of a Step* (T_s) is determined by all its entities. The equation for *PI* is

$$PI = \max_{s=steps}(T_s); \quad T_s = \frac{1}{\sum_{ent=Entities\ of\ s} F_{ent}},$$

where the *Entity Throughput Frequency* F_{ent} is

$$F_{ent} = \frac{WaferCapacity}{ET_{pres_{ent_{st}}} + ET_{proc_{s_{ent}}} + ET_{post_{s_{ent}}} + ET_{swap_{s_{ent}}}}.$$

4.3 Flow with Parallel Entities and Entity Revisiting

Real flows can also have entities, which occur in multiple steps of the flow. This revisiting of entities leads to a higher utilization of the concerning entity. This may create a significant influence on the throughput behavior. The following formulas for PT_1 and PI consider this issue and represent the final version of the calculation approach. We calculate PT_1 as in Section 4.1 by

$$PT_1 = \sum_{s=steps} (\bar{ET}_{pre_s} + \bar{ET}_{proc_s} + \bar{ET}_{post_s}).$$

For the calculation of T_s, the utilization time of every entity of step s is aggregated over the whole flow. Because there is no information about piece routing or preferring of specific entities, the approach assumes that the allocation of an entity in a flow is the same for each step. To obtain PI, we calculate

$$PI = \max_{s=steps}(T_s); \quad T_s = \frac{1}{\sum_{ent=Entities\ of\ s} F_{ent}}$$

where the *Entity Throughput Frequency* F_{ent} is

$$F_{ent} = \frac{WaferCapacity}{T_{ent}},$$

and where the time the entity is busy over all steps of the flow T_{ent} is

$$T_{ent} = \sum_{st=steps} (ET_{pres_{ent_{st}}} + ET_{proc_{s_{ent_{st}}}} + ET_{post_{s_{ent_{st}}}} + ET_{swap_{s_{ent_{st}}}}).$$

5 APPLICATIONS

An automated throughput calculation with recent tool data is a valuable base for decisions concerning how a cluster tool works or if another tool configuration has any influence to the tool's throughput behavior. A smart change or upgrade of the tool configuration may lead to a high performance enhancement at little cost. In addition, this model can be used to estimate future throughput behavior of cluster tools for new flows or new recipes. Consequently, a much higher utilization grade may be possible by planing based on this model. Furthermore, as a by-product new tool configurations are detected automatically. This can be of benefit for observing tool configuration.

By using the information about entity effects to the whole cluster tool behavior, we can significantly improve our availability calculation. This also means, we can estimate whether the tool is stopped by a chamber down event or if it is only slowed down.

6 IMPLEMENTATION

As the implementation is not focus of the paper, we only want to provide a brief outline.

We developed a prototype software based on Java EE 5 technologies. This software runs on a JBoss Enterprise Application Platform *(JBoss EAP)* and is designed as 3-tier architecture. The Java Persistence API *(*JPA) manages persistent data, which is stored on databases. The business logic is implemented with Enterprise Java Beans *(EJB 3.0)*. For the graphical user interface we decided to implement a web interface, which is based on Java Server Faces *(JSF)*. In addition, we used the JBoss Seam framework, which interconnects JPA, EJBs and JSF. Figure 7 shows the web interface for creating throughput scenarios.

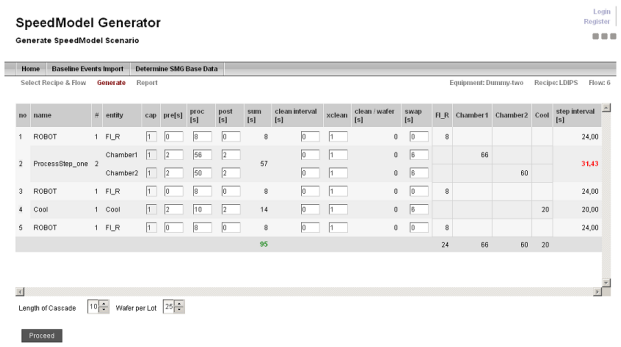


Figure 7: Throughput Model Scenario Screenshot

7 CONCLUSION

We presented an approach for the automated generation and parameterization of throughput models. We created a model for cluster tools, which encloses the tool configuration aspect as well as the material flow aspect. Furthermore, we presented methods to obtain data to populate the model and to calculate the specific throughput values. Further focus will be set on the automated cluster tool availability calculation utilizing cluster tool data and on combined tool throughput computation.

REFERENCES

Dümmler, M. 2004. *Modeling and optimization of cluster tools in semiconductor manufacturing*. Ph. D. thesis, University of Würzburg.

Hopp, W. J., and M. L. Spearman. 2001. *Factory physics: foundations of manufacturing management*. Boston: Addison-Wesley.

Perkinson, T. et al. 1994. Single-wafer cluster tool performance: An analysis of throughput. IEEE Transactions on Semiconductor Manufacturing 7(3): 369-373.

Wood et al. 1994. A generic model for cluster tool throughput time and capacity. In Proceedings of IEEE/SEMI Advanced Semiconductor Manufacturing Conference.

Wood, S. 1996. Simple performance models for integrated processing tools. IEEE Transactions on Semiconductor Manufacturing, 320–328.

AUTHOR BIOGRAPHIES

JAN LANGE is a Graduand in Information Systems Technology at the Department of Electrical Engineering at the Dresden University of Technology. He was previously a working student for AMD Saxony LLC & Co. KG in the department of Industrial Engineering. For this department he is now working for his diploma thesis on the topic of automated cluster tool throughput calculation. His email address is <jan.lange@amd.com>.

KILIAN SCHMIDT is a Senior Industrial Engineer at AMD Saxony LLC & Co. KG in Dresden, Germany, responsible for equipment capacity analysis and optimization as well as future manufacturing system approaches. He obtained a M.S. degree in mechanical engineering from the University of Stuttgart, Germany in 2003. In his part-time Ph.D. research at Dresden University of Technology he currently develops and assesses strategies for fast cycle time with modeling and simulation. He is the author of several papers assessing the potential of small lot size manufacturing in semiconductor front-end production. His email address is <kilian.schmidt@amd.com>.

ROY BÖRNER is a Software & Application Engineer at AMD Saxony LLC & Co. KG in Dresden, Germany. He received a M.Sc. degree in Computer Science from the University of Applied Sciences Dresden in 2005. Roy is working on several software projects regarding throughput controlling and reporting of semiconductor cluster tools. His email address is <roy.boerner@amd.com>.

OLIVER ROSE holds the Chair for Modeling and Simulation at the Institute of Applied Computer Science of the Dresden University of Technology, Germany. He received an M.S. degree in applied mathematics and a Ph.D. degree in computer science from Würzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities, in particular, semiconductor factories. He is a member of IEEE, INFORMS Simulation Society, ASIM, and GI. His web address is <www.simulation-dresden.com> and his email address is <oliver.rose@tu-dresden.de>.